Équipe d'ingénierie de l'Internet (IETF)
Request for Comments : 8881
RFC rendue obsolète : 5661
Catégorie : Sur la voie de la normalisation
ISSN: 2070-1721

D. Noveck, éditeur, NetApp
C. Lever, ORACLE
août 2020

Traduction Claude Brière de L'Isle

Protocole du système de fichier réseau (NFS) version 4.1

Résumé
Le présent document décrit le système de fichier réseau (NFS, *Network File System*) version 4, version mineure 1, incluant les caractéristiques conservées du protocole de base (NFS version 4 version mineure 0, qui est spécifiée dans la RFC 7530) et des extensions faites ultérieurement au protocole. La dernière version mineure de dépend pas de la version 4.0 de NFS , et est considérée comme un protocole distinct.

Ce document rend obsolète la RFC 5661. Il révise substantiellement le traitement des caractéristiques relatives à l'espace de noms de multi-serveurs, remplaçant la description de ces caractéristiques qui apparaît dans la RFC 5661.

Statut de ce mémoire
Ceci est un document de l'Internet sur la voie de la normalisation.

Le présent document a été produit par l'équipe d'ingénierie de l'Internet (IETF). Il représente le consensus de la communauté de l'IETF. Il a subi une révision publique et sa publication a été approuvée par le groupe de pilotage de l'ingénierie de l'Internet (IESG). Tous les documents approuvés par l'IESG ne sont pas candidats à devenir une norme de l'Internet ; voir la paragraphe 2 de la RFC5741.

Les informations sur le statut actuel du présent document, tout errata, et comment fournir des réactions sur lui peuvent être obtenues à http://www.rfc-editor.org/info/rfc8881

Notice de droits de reproduction
Copyright (c) 2020 IETF Trust et les personnes identifiées comme auteurs du document. Tous droits réservés.

Le présent document est soumis au BCP 78 et aux dispositions légales de l'IETF Trust qui se rapportent aux documents de l'IETF (http://trustee.ietf.org/license-info) en vigueur à la date de publication de ce document. Prière de revoir ces documents avec attention, car ils décrivent vos droits et obligations par rapport à ce document. Les composants de code extraits du présent document doivent inclure le texte de licence simplifié de BSD comme décrit au paragraphe 4.e des dispositions légales du Trust et sont fournis sans garantie comme décrit dans la licence de BSD simplifiée.

Le présent document peut contenir des matériaux provenant de documents de l'IETF ou de contributions à l'IETF publiées ou rendues disponibles au public avant le 10 novembre 2008. La ou les personnes qui ont le contrôle des droits de reproduction sur tout ou partie de ces matériaux peuvent n'avoir pas accordé à l'IETF Trust le droit de permettre des modifications de ces matériaux en dehors du processus de normalisation de l'IETF. Sans l'obtention d'une licence adéquate de la part de la ou des personnes qui ont le contrôle des droits de reproduction de ces matériaux, le présent document ne peut pas être modifié en dehors du processus de normalisation de l'IETF, et des travaux dérivés ne peuvent pas être créés en dehors du processus de normalisation de l'IETF, excepté pour le formater en vue de sa publication comme RFC ou pour le traduire dans une autre langue que l'anglais.

Table des matières

1. Introduction....5
1.1 Introduction à cette mise à jour....5
1.2 Protocole NFS version 4.1....6
1.3 Langage des exigences....7
1.4 Domaine d'application de ce document....7
1.5 Buts de NFSv4....7
1.6 Buts de NFSv4.1....7
1.7 Définitions générales....8
1.8 Vue d'ensemble des caractéristiques de NFSv4.1....9
1.9 Différences avec NFSv4.0....11
2. Infrastructure centrale....12

2.1 Introduction....12
2.2 RPC et XDR....12
2.3 COMPOUND et CB_COMPOUND....14
2.4 Identifiants de client et propriétaires de client....14
2.5 Propriétaires de serveur....17
2.6 Négociation de service de sécurité....18
2.7 Version mineure....21
2.8 Services de sécurité non fondés sur RPC....22
2.9 Couches de transport....23
2.10 Session....24
3. Constantes et types de données du protocole....50
3.1 Constantes de base....50
3.2 Types de données de base....51
3.3 Types de données structurées....52
4. Brides de fichier....57
4.1 Obtention de la première bride de fichier....57
4.2 Types de bride de fichier....58
4.3 Méthode de construction d'une bride de fichier volatile....60
4.4 Récupération du client de l'expiration de la bride de fichier....60
5. Attributs de fichier....61
5.1 Attributs exigés....61
5.2 Attributs recommandés....61
5.3 Attributs nommés....62
5.4 Classification des attributs....63
5.5 Attributs Set-Only et Get-Only....63
5.6 Attributs exigés - liste et références de définition....63
5.7 Attributs recommandés - liste et références de définition....64
5.8 Définition des attributs....65
5.9 Interprétation de owner et de owner_group....70
5.10 Attributs de casse de caractères....72
5.11 Attributs de notification de répertoire....72
5.12 Définitions d'attributs pNFS....72
5.13 Attributs de rétention....73
6. Attributs de contrôle d'accès....75
6.1 Buts....75
6.2 Discussion des attributs de fichier....76
6.3 Méthodes courantes....83
6.4 Exigences....85
7. Espace de noms d'un seul serveur....88
7.1 Exportations de serveur....88
7.2 Navigation dans les exportations....89
7.3 Système de pseudo fichiers de serveur....89
7.4 Racines multiples....89
7.5 Volatilité des brides de fichier....89
7.6 Racine exportée....90
7.7 Croisement de point culminant....90
7.8 Politique de sécurité et présentation d'espace de noms....90
8. Gestion d'état....91
8.1 Identifiants de client et de session....91
8.2 Définition de Stateid....92
8.3 Renouvellement de prêt....97
8.4 Récupération de panne....98
8.5 Révocation de verrous par le serveur....104
8.6 Prêts courts et longs....104
8.7 Horloges, délai de propagation, et calcul de l'expiration du prêt....105
8.8 Infrastructure de verrouillage obsolète de NFSv4.0....105
9. Verrouillage de fichier et réservations partagées....106
9.1 Verrouillages d'ouvertures et de gamme d'octets....106
9.2 Gammes de verrous....108
9.3 Mise à niveau et dégradation de verrous....108
9.4 Valeurs de seqid d'identifiant d'état et verrouillages de gammes d'octets....109

9.5 Questions de propriétaires d'ouvertures multiples....109
9.6 Verrous bloquants....109
9.7 Partage de réservations....110
9.8 Opérations OPEN/CLOSE....110
9.9 Mise à niveau et dégradation de OPEN....111
9.10 OPEN parallèles....111
9.11 Réclamation de verrous Open et de gamme d'octets....112
10. Mise en antémémoire côté client....112
10.1 Défis de performances pour la mise en antémémoire côté client....112
10.2 Délégation et rappels....113
10.3 Mise en antémémoire de données....115
10.4 Délégation ouverte....118
10.5 Mise en antémémoire de données et révocation....124
10.6 Mise en antémémoire d'attribut....125
10.7 Mise en antémémoire de données et métadonnées et fichiers transposés de la mémoire....126
10.8 Mise en antémémoire de nom et de répertoire sans délégation de répertoire....127
10.9 Délégations de répertoire....129
11. Espace de noms multi-serveurs....130
11.1 Terminologie....131
11.2 Attributs de localisation de système de fichiers....133
11.3 Présence ou absence de système de fichiers....134
11.4 Obtenir des attributs pour système de fichiers absent....134
11.5 Utilisations des informations de localisation de système de fichiers....135
11.6 Multiplexage sans information de localisation de système de fichiers....141
11.7 Utilisateurs et groupes dans un espace de noms multi-serveurs....141
11.8 Considérations supplémentaires sur le côté client....142
11.9 Vue d'ensemble des transitions d'accès au fichier....142
11.10 Effectuer des transitions de point d'extrémité réseau....142
11.11 Effectuer des transitions de système de fichiers....143
11.12 Transfert d'état sur migration....149
11.13 Responsabilités du client lors d'une transition d'accès....150
11.14 Responsabilités du serveur lors d'une migration....155
11.15 Réalisation des références de système de fichiers....158
11.16 Attribut fs_locations....161
11.17 Attribut fs_locations_info....163
11.18 Attribut fs_status....170
12. NFS parallèle (pNFS)....172
12.1 Introduction....172
12.2 Définitions de pNFS....173
12.3 Opérations de pNFS....177
12.4 Attributs de pNFS....177
12.5 Sémantique de disposition....177
12.6 Mécanique de pNFS....186
12.7 Récupération....187
12.8 Rôles d'appareil de métadonnées et de mémorisation....190
12.9 Considérations de sécurité pour pNFS....190
13. NFSv4.1 comme protocole de mémorisation dans pNFS : type de disposition de fichier....191
13.1 Identifiant de client et considérations de session....191
13.2 Définitions de dispositions de fichier....193
13.3 Types de données de disposition de fichier....193
13.4 Interprétation de la disposition de fichier....195
13.5 Serveur de données multi chemins....199
13.6 Opérations envoyées aux serveurs de données NFSv4.1....200
13.7 COMMIT sur serveur de métadonnées....201
13.8 Disposition iomode....202
13.9 Coordination d'état de serveur de métadonnées et de données....202
13.10 Taille de fichier composant au serveur de données....204
13.11 Révocation et barrage de disposition....204
13.12 Considérations de sécurité pour le type de disposition de fichier....205
14. Internationalisation....205
14.1 Profil Stringprep pour le type utf8str_cs....206

14.2 Profil stringprep pour le type utf8str_cis....207
14.3 Profil Stringprep pour le type utf8str_mixed....207
14.4 Capacités UTF-8....208
14.5 Erreurs relatives à UTF-8....208
15. Valeurs d'erreur....208
15.1 Définitions des erreurs....209
15.2 Opérations et leurs erreurs valides....222
15.3 Opérations de rappel et leurs erreurs valides....229
15.4 Erreurs et opérations qui les utilisent....230
16. Procédures de NFSv4.1....236
16.1 Procédure 0 : NULL - pas d'opération....236
16.2 Procédure 1 : COMPOUND - opérations Compound....237
17. Opérations EXIGÉES, RECOMMANDÉES, ou FACULTATIVES....243
18. Opérations NFSv4.1....245
18.1 Opération 3 : ACCESS - vérification des droits d'accès....245
18.2 Opération 4 : CLOSE - fermeture de fichier....248
18.3 Opération 5 : COMMIT - engagement des données en antémémoire....249
18.4 Opération 6 : CREATE - crée un objet de fichier non régulier....251
18.5 Opération 7 : DELEGPURGE - purge les délégations en attente de récupération....252
18.6 Opération 8 : DELEGRETURN - retourne la délégation....253
18.7 Opération 9 : GETATTR - obtient les attributs....253
18.8 Opération 10 : GETFH - obtient la bride de fichier courante....254
18.9 Opération 11 : LINK - crée une liaison à un fichier....255
18.10 Opération 12 : LOCK - crée un verrou....257
18.11 Opération 13 : LOCKT - vérification de verrou....259
18.12 Opération 14 : LOCKU - déverrouillage de fichier....260
18.13 Opération 15 : LOOKUP - recherche un nom de fichier....261
18.14 Opération 16 : LOOKUPP - recherche du répertoire parent....262
18.15 Opération 17 : NVERIFY - vérifie la différence des attributs....263
18.16 Opération 18 : OPEN - ouvre un fichier régulier....263
18.17 Opération 19 : OPENATTR - ouvre le répertoire d'attributs désigné....274
18.18 Opération 21 : OPEN_DOWNGRADE - réduit l'accès en ouverture de fichier....274
18.19 Opération 22 : PUTFH - établit la bride de fichier courante....275
18.20 Opération 23 : PUTPUBFH - établit une bride de fichier publique....276
18.21 Opération 24 : PUTROOTFH - établir la bride de fichier racine....277
18.22 Opération 25 : READ - lire à partir du fichier....277
18.23 Opération 26 : READDIR - lit le répertoire....279
18.24 Opération 27 : READLINK - lire le lien symbolique....281
18.25 Opération 28 : REMOVE - supprime l'objet de système de fichier....281
18.26 Opération 29 : RENAME - Renomme l'entrée de répertoire....283
18.27 Opération 31 : RESTOREFH - restaure la bride de fichier sauvegardée....285
18.28 Opération 32 : SAVEFH - sauvegarde la bride de fichier courante....286
18.29 Opération 33 : SECINFO - obtient la sécurité disponible....286
18.30 Opération 34 : SETATTR - établir les attributs....288
18.31 Opération 37 : VERIFY - vérifie les mêmes attributs....290
18.32 Opération 38 : WRITE - écrit sur le fichier....291
18.33 Opération 40 : BACKCHANNEL_CTL - contrôle le canal de retour....293
18.34 Opération 41 : BIND_CONN_TO_SESSION - associe la connexion à la session....294
18.35 Opération 42 : EXCHANGE_ID - instancie l'identifiant de client....296
18.36 Opération 43 : CREATE_SESSION - Crée une nouvelle session et confirme l'identifiant de client....305
18.37 Opération 44 : DESTROY_SESSION - détruit une session....311
18.38 Opération 45 : FREE_STATEID - libère l'identifiant d'état sans verrous....312
18.39 Opération 46 : GET_DIR_DELEGATION - obtient une délégation de répertoire....312
18.40 Opération 47 : GETDEVICEINFO - obtient les informations d'appareil....314
18.41 Opération 48 : GETDEVICELIST - obtient toutes les transpositions d'appareils pour un système de fichiers….316
18.42 Opération 49 : LAYOUTCOMMIT - engage les écritures faites en utilisant une disposition....317
18.43 Opération 50 : LAYOUTGET - obtient les informations de disposition....319
18.44 Opération 51 : LAYOUTRETURN - libère les informations de disposition....324
18.45 Opération 52 : SECINFO_NO_NAME - obtient la sécurité sur un objet non nommé....327
18.46 Opération 53 : SEQUENCE - fournit le séquençage et le contrôle par procédure....328
18.47 Opération 54 : SET_SSV - met à jour le SSV pour un identifiant de client....331

18.48 Opération 55 : TEST_STATEID - essai de validité des identifiants d'état....333
18.49 Opération 56 : WANT_DELEGATION - demande de délégation....334
18.50 Opération 57 : DESTROY_CLIENTID - détruit un identifiant de client....335
18.51 Opération 58 : RECLAIM_COMPLETE - indique la fin de la réclamation....336
18.52 Opération 10044 : ILLEGAL - opération illégale....338
19. Procédures de rappel NFSv4.1....338
19.1 Procédure 0 : CB_NULL - pas d'opération....338
19.2 Procédure 1 : CB_COMPOUND - opérations composées....339
20. Opérations de rappel NFSv4.1....341
20.1 Opération 3 : CB_GETATTR - obtient les attributs....341
20.2 Opération 4 : CB_RECALL - rappelle une délégation....342
20.3 Opération 5 : CB_LAYOUTRECALL - rappel de disposition du client....342
20.4 Opération 6 : CB_NOTIFY - notifie au client les changements de répertoire....344
20.5 Opération 7: CB_PUSH_DELEG - offre au client une délégation précédemment demandée....347
20.6 Opération 8 : CB_RECALL_ANY - garde tous les objets rappelables....347
20.7 Opération 9 : CB_RECALLABLE_OBJ_AVAIL - signale des ressources pour des objets rappelables....349
20.8 Opération 10 : CB_RECALL_SLOT - change les limites de contrôle de flux....350
20.9 Opération 11 : CB_SEQUENCE - fournit le séquençage et le contrôle du canal de retour....350
20.10 Opération 12 : CB_WANTS_CANCELLED - annule la délégation Wants en instance....352
20.11 Opération 13 : CB_NOTIFY_LOCK - notifie au client une possible disponibilité de verrou....352
20.12 Opération 14 : CB_NOTIFY_DEVICEID - notifie au client des changements d'identifiant d'appareil....353
20.13 Opération 10044 : CB_ILLEGAL - opération de rappel illégale....355
21. Considérations sur la sécurité....355
22. Considérations relatives à l'IANA....357
22.1 Actions de l'IANA....357
22.2 Définitions des attribut désignés....358
22.3 Notifications des identifiants d'appareil....358
22.4 Types de rappel d'objet....359
22.4.2 Mise à jour des enregistrements....360
22.5 Types de disposition....360
22.6 Définitions des variables de chemin....362
23. Références....364
23.1 Références normatives....364
23.2 Références pour information....365
Appendice A. Motifs de cette mise à jour....367
Appendice B. Changements apportés par cette mise à jour....368
B.1 Révisions à la Section 11 de la RFC 5661....368
B.2 Révisions des opérations de la RFC 5661....370
B.3 Révisions aux définitions d'erreur dans la RFC 5661....371
B.4 Autres révisions à la RFC 5661....372
Appendice C. Questions de sécurité qui doivent être traitées....372
Remerciements....373
Adresse des auteurs....375

1. Introduction

1.1 Introduction à cette mise à jour

Deux importantes caractéristiques précédemment définies dans la version mineure 0 mais jamais pleinement adressées dans la version mineure 1 sont le multiplexage, qui est l'utilisation simultanée de plusieurs connexions entre un client et un serveur, potentiellement à des adresses réseau différentes, et la migration d'état transparente, qui permet de transférer un système de fichiers entre serveurs d'une façon qui donne au client la capacité de maintenir son état de verrouillage existant à travers le transfert.

La description révisée du protocole NFS version 4 version mineure 1 (NFSv4.1) présentée dans cette mise à jour est nécessaire pour permettre une pleine utilisation de ces caractéristiques avec les autres caractéristiques d'espace de noms multi-serveurs. Le présent document est une forme de description mise à jour du protocole NFSv4.1 précédemment défini dans la [RFC5661]. La RFC 5661 est rendue obsolète par le présent document. Cependant, la mise à jour a une portée limitée et se concentre sur l'établissement de la pleine utilisation du multiplexage et de la migration d'état transparente. Le besoin de ces changements est discuté à l'Appendice A. L'Appendice B décrit les changements spécifiques faits pour arriver au texte actuel.

Cette mise à jour de portée limitée remplace la RFC NFSv4.1 actuelle dans l'intention de fournir une spécification complète et d'autorité, dont la motivation est discutée dans [REV], traitant les questions dans le domaine d'application de la mise à jour. Cependant, elle ne traite pas de problèmes connus mais qui sortent de cette portée limitée comme on pourrait l'attendre d'une mise à jour complète du protocole. On énumère ci-dessous certains domaines qui sont connus pour avoir besoin d'une mise à jour future du protocole :

* Du travail est nécessaire à l'égard de la [RFC8178], qui établit les règles des numéros de version de NFSv4. Comme la RFC 5661 n'est actuellement pas cohérente avec ce document, des changements sont nécessaires pour arriver à une situation dans laquelle il n'y aurait pas besoin d'une RFC 8178 pour mettre à jour la spécification NFSv4.1.

* Du travail est nécessaire à l'égard de la [RFC8434], qui établit les exigences pour les types de disposition de NFS parallèle (pNFS) qui ne sont pas définis de façon claire dans la RFC 5661. Quand ce travail sera fait et que les documents résultants seront approuvés, le nouveau document de spécification de NFSv4.1 fournira un ensemble clair d'exigences pour les types de dispositions et une description de type de disposition de fichier qui se conforme à ces exigences. Les autres types de disposition auront aussi leurs propres documents de spécification conformes à ces exigences.

* Du travail est nécessaire pour traiter les nombreux rapports d'erreur relatifs à la RFC 5661, autres que l'errata 2006 [ERR-2006], qui est traité dans le présent document. Il ne pouvait être différé de traiter ce rapport d'erreur à cause de l'interaction des changements qui y sont suggérés et du nouveau traitement décrit de migration d'état et de session.

Les rapports d'erreurs qui ont été différés et qui devront être traités dans un document ultérieur incluent des rapports auxquels ont été actuellement alloués une gamme de statuts dans le système de rapports d'errata, incluant des rapports marqués "Accepté" et ceux marqués "pour mise à jour du document" parce que le changement était trop minime pour avoir besoin d'une correction immédiate.

De plus, il y a un ensemble d'autres rapports, incluant au moins un dans l'état "Rejeté", qui vont devoir être traités dans un document ultérieur. Cela va impliquer de faire des changements aux décisions consensuelles reflétées dans la RFC 5661, dans des situations où le groupe de travail avait décidé que le traitement de la RFC 5661 est incorrect et doit être révisé pour refléter le nouveau consensus du groupe de travail et assurer la compatibilité avec les mises en œuvre existantes qui ne suivent pas le traitement décrit dans la RFC 5661.

Noter qu'on s'attend à ce que tous ces rapports d'erreur restent pertinents pour les développeurs et les auteurs d'une éventuelle rfc5661bis, en dépit du fait que le présent document rende obsolète la [RFC5661].

* Il y a besoin d'une nouvelle approche de la description de l'internationalisation car la section actuelle d'internationalisation (paragraphe 14) n'a jamais été mise en œuvre et ne satisfait pas les besoins du protocole NFSv4. Les solutions possibles sont de créer une nouvelle section d'internationalisation modélisée sur celle de la [RFC7530] ou de créer un nouveau document décrivant l'internationalisation pour toutes les versions mineures de NFSv4 et de référencer ce document dans les RFC qui définissent à la fois NFSv4.0 et NFSv4.1.

* Il y a besoin de réviser le traitement de la sécurité dans NFSv4.1. Les problèmes du traitement existant sont discutés à l'Appendice C.

Tant que les travaux ci-dessus n'auront pas été réalisés, il n'y aura pas un ensemble cohérent de documents qui fournissent une description du protocole NFSv4.1, et toute description complète impliquerait des documents qui mettent à jour d'autres documents au sein de la spécification. Les mises à jour appliquées par la [RFC8434] et la [RFC8178] à la RFC 5661 s'appliquent aussi à la présente spécification, et vont s'appliquer à toute spécification v4.1 suivante jusqu'à ce que ce travail soit terminé.

1.2 Protocole NFS version 4.1

Le protocole NFS version 4 version mineure 1 (NFSv4.1) est la seconde version mineure du protocole NFS version 4 (NFSv4). La première version mineure, NFSv4.0, est maintenant décrite dans la [RFC7530]. Elle suit généralement les lignes directrices pour les versions mineures données à la Section 10 de la [RFC3530]. Cependant, elle diverge des directives 11 ("un client et un serveur qui prennent en charge la version mineure X doivent prendre en charge les versions mineures 0 à X-1") et 12 ("aucune nouvelle caractéristique ne peut être introduite comme obligatoire dans une version mineure"). Ces divergences sont dues à l'introduction du modèle de sessions pour la gestion d'opérations non idempotentes et l'opération RECLAIM_COMPLETE. Ces deux nouvelles caractéristiques sont de nature infra structurelle et simplifient

la mise en œuvre des caractéristiques existantes et d'autres nouvelles. Faire d'elles autre chose que EXIGÉES ajouterait une complexité indue à la définition et à la mise en œuvre du protocole. NFSv4.1 met à jour en conséquence les directives sur les versions mineures (paragraphe 2.7).

Comme version mineure, NFSv4.1 est cohérent avec les buts globaux de NFSv4, mais étend le protocole afin de mieux satisfaire ces buts, sur la base de l'expérience de NFSv4.0. De plus, NFSv4.1 a adopté des buts supplémentaires, qui motivent certaines des extensions majeures dans NFSv4.1.

1.3 Langage des exigences

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

1.4 Domaine d'application de ce document

Le présent document décrit le protocole NFSv4.1. Par rapport à NFSv4.0, le présent document :
* ne décrit pas le protocole NFSv4.0, sauf pour le différencier de NFSv4.1 ;
* ne modifie pas les spécifications du protocole NFSv4.0 ;
* ne précise pas le protocole NFSv4.0.

1.5 Buts de NFSv4

Le protocole NFSv4 est une révision supplémentaire du protocole NFS défini déjà par NFSv3 [RFC1813]. Il conserve les caractéristiques essentielles des versions précédentes : récupération aisée, indépendance aux protocoles de transport, aux systèmes d'exploitation, et aux systèmes de fichiers, simplicité, et bonnes performances. NFSv4 a les buts suivants :

* Accès amélioré et bonnes performances sur l'Internet : le protocole est conçu pour traverser facilement les pare-feu, avoir de bonnes performances quand la latence est élevée et la bande passante faible, et s'adapte à de très grands nombres de clients par serveur.

* Forte sécurité avec négociation incorporée au protocole : le protocole s'appuie sur les travaux du groupe de travail ONCRPC en prenant en charge le protocole RPCSEC_GSS. De plus, le protocole NFSv4.1 fournit un mécanisme pour permettre aux clients et serveurs la capacité de négocier la sécurité et exige des clients et serveurs qu'ils prennent en charge un ensemble minimal de schémas de sécurité.

* Bonne interopérabilité inter plates-formes : le protocole caractérise un modèle de système de fichiers qui fournit un ensemble utile de caractéristiques communes qui ne favorise pas abusivement un système de fichier ou d'exploitation par rapport à un autre.

* Conçu pour des extensions de protocole : le protocole est conçu pour accepter des extensions standard au sein d'un cadre qui permet et encourage la rétro compatibilité.

1.6 Buts de NFSv4.1

NFSv4.1 a les buts suivants au sein du cadre établi par les buts NFSv4 globaux :
* Corriger des faiblesses structurelles significatives et des omissions découvertes dans le protocole de base,
* Apporter de la clarté et de la spécificité aux zones laissées non traitées ou traitées insuffisamment en détail dans le protocole de base. Cependant, comme déclaré au paragraphe 1.4, le but n'est pas de préciser le protocole NFSv4.0 dans la spécification NFSv4.1.
* Ajouter des caractéristiques spécifiques sur la base de l'expérience du protocole existant et des récents développements de l'industrie.
* Fournir à la prise en charge du protocole un moyen de tirer parti des déploiements de grappes de serveurs incluant la capacité de fournir un accès parallèle adaptable aux fichiers distribués sur de multiples serveurs.

1.7 Définitions générales

Les définitions suivantes fournissent un contexte approprié pour le lecteur.

Octet : dans ce document, un octet est un élément de données d'exactement 8 bits.

Client : le client est l'entité qui accède aux ressources du serveur NFS. Le client peut être une application qui contient la logique pour accéder directement au serveur NFS. Le client peut aussi être le client traditionnel de système d'exploitation qui fournit des services de système de fichier distant pour un ensemble d'applications. Un client est identifié de façon univoque par un propriétaire de client. En référence au verrouillage de gamme d'octets, le client est aussi l'entité qui maintient un ensemble de verrous au nom d'une ou plusieurs applications. Ce client est responsable de la récupération de panne ou défaillance pour les verrous qu'il gère. Noter que plusieurs clients peuvent partager le même transport et connexion et plusieurs clients peuvent exister sur le même nœud de réseau.

Identifiant de client : c'est une quantité de 64 bits utilisée comme référence abrégée unique à un vérificateur fourni par le client et le propriétaire du client. Le serveur est responsable de la fourniture de l'identifiant de client.

Propriétaire de client : c'est une chaîne unique, opaque au serveur, qui identifie un client. Plusieurs connexions réseaux et adresses réseau de source originaires de ces connexions peuvent partager un propriétaire de client. Le serveur est supposé traiter les demandes provenant de connexions avec le même propriétaire de client comme provenant du même client.

Système de fichiers : c'est une collection d'objets sur un serveur (comme identifié par l'identifiant majeur d'un propriétaire de serveur, qui est défini plus loin) qui partage le même attribut fsid *(identifiant de système de fichiers)* (voir le paragraphe 5.8.1.9).

Prêt : un prêt est un intervalle de temps défini par le serveur pour lequel des verrous irrévocables sont accordés au client. À la fin de la période de prêt, les verrous peuvent être révoqués si le prêt n'a pas été étendu. Un verrou doit être révoqué si un verrou en conflit a été accordé après l'intervalle de prêt. Un serveur accorde à un client un seul prêt pour tous les états.

Verrou : le terme "verrou" est utilisé pour se référer aux verrous de gamme d'octets (dans les environnements UNIX, aussi appelés des enregistrements) aux réservations, délégations, ou dispositions partagées sauf mention contraire spécifique.

Vérificateur d'état de secret (SSV, *Secret State Verifier*) : le SSV est une clé secrète unique partagée entre un client et un serveur. Le SSV sert de clé secrète pour un mécanisme interne (à NFSv4.1) de service de sécurité générique (GSS, *Generic Security Service*) (le mécanisme GSS SSV ; voir le paragraphe 2.10.9). Le mécanisme GSS SSV utilise le SSV pour calculer les jetons de code d'intégrité de message (MIC, *message integrity code*) et de Wrap. Voir au paragraphe 2.10.8.3 plus de détails sur comment NFSv4.1 utilise les mécanismes de SSV et de SSV GSS.

Serveur : le serveur est l'entité chargée de coordonner l'accès du client à un ensemble de systèmes de fichiers et est identifié par un propriétaire de serveur. Un serveur peut s'étaler sur plusieurs adresses réseau.

Propriétaire de serveur : le propriétaire de serveur identifie le serveur au client. Le propriétaire de serveur consiste en un identifiant majeur et un identifiant mineur. Quand le client a deux connexions chacune avec un homologue avec le même identifiant majeur, le client suppose que les deux homologues sont le même serveur (l'espace de noms de serveur est le même via chaque connexion) et que l'état de verrou est partageable à travers les deux connexions. Quand chaque homologue a à la fois le même identifiant majeur et mineur, le client suppose que chaque connexion pourrait être associable avec la même session.

Mémorisation stable : c'est une mémorisation à partir de laquelle les données mémorisées par un serveur NFSv4.1 peuvent être récupérées sans perte de données à partir de défaillances d'alimentation multiples (incluant des défaillances d'alimentation en cascade, c'est-à-dire, plusieurs défaillances d'alimentation en succession rapide) de défaillance du système d'exploitation, et/ou défaillance matérielle de composants autres que le support de mémorisation lui-même (comme un disque, une RAM non volatile, une mémoire flash, etc.).

Des exemples de mémorisation stable permise pour un serveur NFS incluent :

1. Engagement des données sur le support ; c'est-à-dire, les données modifiées ont été écrites avec succès sur le support de disque, par exemple, la platine de disque.
2. Un pilote de disque de réponse immédiate avec support de batterie, mémorisation sur pilote de mémorisation intermédiaire ou système d'alimentation électrique sans interruption (UPS, *uninterruptible power system*).
3. Engagement des données par le serveur avec mémorisation intermédiaire avec support de batterie et logiciel de récupération.

4. Engagement d'antémémoire avec système d'alimentation électrique sans interruption (UPS) et logiciel de récupération.

Identifiant d'état : un identifiant d'état (identifiant d'état) est une quantité de 128 bits retournée par un serveur, qui identifie de façon univoque les états ouvert et de verrouillage fournis par le serveur pour une paire spécifique propriétaire d'ouverture ou propriétaire de verrou/propriétaire d'ouverture pour un fichier spécifique et type de verrou.

Vérificateur : quantité de 64 bits générée par le client que le serveur peut utiliser pour déterminer si le client a redémarré et perdu tout l'état de verrou précédent.

1.8 Vue d'ensemble des caractéristiques de NFSv4.1

Les caractéristiques majeures du protocole NFSv4.1 vont être brièvement revues. Cela va être fait en fournissant un contexte approprié pour le lecteur qui est familiarisé avec les précédentes versions du protocole NFS et le lecteur qui est nouveau sur les protocoles NFS. Pour ces derniers, il y a quand même un ensemble de connaissances fondamentales attendu. Le lecteur devrait être familiarisé avec les protocoles de représentation de données externes (XDR, *External Data Representation*) et d'appel de procédure distante (RPC, *Remote Procedure Call*) décrits dans les [RFC4506] et [RFC5531]. Une connaissance basique des systèmes de fichiers et des systèmes de fichiers distribués est aussi attendue.

En général, cette spécification de NFSv4.1 ne va pas distinguer les caractéristiques ajoutées dans la version mineure 1 de celles présentes dans le protocole de base mais va traiter NFSv4.1 comme un tout. Voir au paragraphe 1.9 un sommaire des différences entre NFSv4.0 et NFSv4.1.

1.8.1 RPC et sécurité

Comme avec les précédentes versions de NFS, les mécanismes de représentation de données externes (XDR) et d'appel de procédure distante (RPC) utilisés pour le protocole NFSv4.1 sont ceux définis dans les [RFC4506] et [RFC5531]. Pour satisfaire les exigences de sécurité de bout en bout, le cadre RPCSEC_GSS [RFC2203] est utilisé pour étendre la sécurité RPC de base. Avec l'utilisation de RPCSEC_GSS, divers mécanismes peuvent être fournis pour offrir l'authentification, l'intégrité, et la confidentialité au protocole NFSv4. Kerberos V5 est utilisé comme décrit dans la [RFC4121] pour fournir un cadre de sécurité. Avec l'utilisation de RPCSEC_GSS, d'autres mécanismes peuvent aussi être spécifiés et utilisés pour la sécurité de NFSv4.1.

Pour permettre la négociation de sécurité dans la bande, le protocole NFSv4.1 a des opérations qui fournissent au client une méthode d'interrogation du serveur sur ses politiques concernant les mécanismes de sécurité qui doivent être utilisés pour l'accès aux ressources du système de fichiers du serveur. Avec cela, le client peut en toute sécurité correspondre aux mécanismes de sécurité qui satisfont les politiques spécifiées chez le client et le serveur.

NFSv4.1 introduit un accès parallèle (voir le paragraphe 1.8.2.2) qui est appelé pNFS. Le cadre de sécurité décrit dans ce paragraphe est significativement modifié par l'introduction de pNFS (voir le paragraphe 12.9) parce que l'accès aux données n'est parfois pas sur RPC. Le niveau de signification varie avec le protocole de mémorisation (voir le paragraphe 12.2.5) et peut être d'un impact nul (voir le paragraphe 13.12).

1.8.2 Structure du protocole

1.8.2.1 Protocole central

À la différence de NFSv3, qui utilise une série de protocoles traditionnels (par exemple, NLM, NSM (Network Status Monitor, *surveillant de l'état du réseau*), MOUNT) dans toutes les versions mineures de NFSv4 un seul protocole RPC est utilisé pour faire les demandes au serveur. Les facilités qui étaient des protocoles distincts, comme le verrouillage, sont maintenant intégrées dans un seul protocole unifié.

1.8.2.2 Accès en parallèle

La version mineure 1 prend en charge de hautes performances d'accès aux données à une mise en œuvre de serveur en grappe en permettant une séparation de l'accès aux métadonnées et de l'accès aux données, avec ce dernier fait sur plusieurs serveurs en parallèle.

Un tel accès en parallèle aux données est contrôlé par des objets rappelables appelés des "dispositions", qui sont intégrés au

modèle de verrouillage du protocole. Les clients dirigent les demandes d'accès aux données sur un ensemble de serveurs de données spécifiés par la disposition via un protocole de mémorisation de données qui peut être NFSv4.1 ou peut être un autre protocole.

Parce que les protocoles utilisés pour l'accès en parallèle aux données ne sont pas nécessairement fondés sur RPC, le modèle de sécurité fondé sur RPC (paragraphe 1.8.1) est évidemment impacté (voir le paragraphe 12.9). Le degré d'impact varie avec le protocole de mémorisation (voir le paragraphe 12.2.5) utilisé pour l'accès aux données, et peut être nul (voir le paragraphe 13.12).

1.8.3 Modèle de système de fichiers

Le modèle général de système de fichiers utilisé pour le protocole NFSv4.1 est le même que dans les précédentes versions. Le serveur de système de fichiers est hiérarchisé avec les fichiers réguliers contenus qui sont traités comme des flux d'octets opaques. Une petite différence est que les noms de fichiers et répertoires sont codés avec UTF-8 pour respecter les bases de l'internationalisation.

Le protocole NFSv4.1 n'exige pas un protocole séparé pour fournir la transposition initiale entre nom de chemin et bride de fichier *(file handle)*. Tous les systèmes de fichiers exportés par un serveur sont présentés comme une arborescence afin que tous les systèmes de fichiers soient accessibles à partir d'une bride de fichier racine globale particulière par serveur. Cela permet d'utiliser les opérations LOOKUP pour effectuer les fonctions précédemment fournies par le protocole MOUNT. Le serveur fournit tous les pseudo systèmes de fichiers nécessaires pour boucher les trous qui surviennent du fait de trous non exportés entre les systèmes de fichiers exportés.

1.8.3.1 Brides de fichier

Comme dans les versions précédentes du protocole NFS, des brides de fichier opaques sont utilisées pour identifier les fichiers et répertoires individuels. Les opérations de type de recherche et de création traduisent les noms de fichier et de répertoire en brides de fichier, qui sont alors utilisées pour identifier les objets dans les opérations suivantes.

Le protocole NFSv4.1 fournit la prise en charge de brides de fichier persistantes, dont la validité est garantie pour la durée de vie de l'objet de système de fichiers désigné. De plus, il fournit un soutien aux serveurs pour fournir des brides de fichier avec des garanties de validité plus limitées, appelées des brides de fichier volatiles.

1.8.3.2 Attributs de fichier

Le protocole NFSv4.1 a une structure d'attribut d'objet de fichier riche et extensible, qui est divisée en attributs EXIGÉS, RECOMMANDÉS, et nommés (voir la paragraphe 5).

Plusieurs (mais pas tous) des attributs EXIGÉS sont dérivés des attributs de NFSv3 (voir la définition du type de données fattr3 dans la [RFC1813]). Un exemple d'attribut EXIGÉ est le type d'objet de fichier (paragraphe 5.8.1.2) de sorte que les fichiers réguliers peuvent être distingués des répertoires (aussi appelés des dossiers dans certains environnements de fonctionnement) et autre types d'objets. Les attributs EXIGÉS sont discutés au paragraphe 5.1.

Un exemple de trois attributs RECOMMANDÉS est acl, sacl, et dacl. Ces attributs définissent une liste de contrôle d'accès (ACL, *Access Control List*) sur un objet de fichier (paragraphe 6). Une ACL fournit un contrôle d'accès aux répertoires et fichiers au delà du modèle utilisé dans NFSv3. La définition d'ACL permet la spécification d'ensembles spécifiques de permissions pour des utilisateurs individuels et des groupes. De plus, l'héritage d'ACL permet la propagation de permissions et restrictions d'accès le long d'une arborescence de répertoires lorsque des objets de système de fichiers sont créés. Les attributs RECOMMANDÉS sont discutés au paragraphe 5.2.

Un attribut nommé est un flux d'octets opaque qui est associé à un répertoire ou fichier et auquel on se réfère par un nom de chaîne. Les attributs nommés sont destinés à être utilisés par les applications clientes comme une méthode pour associer des données spécifiques d'application à un fichier ou répertoire régulier. NFSv4.1 modifie les attributs nommés relatifs à NFSv4.0 en resserrant les opérations permises afin d'empêcher le développement de mises en œuvre non interopérables. Les attributs nommés sont discutés au paragraphe 5.3.

1.8.3.3 Espace de noms multi-serveurs

NFSv4.1 contient un certain nombre de caractéristiques pour permettre la mise en œuvre d'espaces de noms qui traversent les frontières des serveurs et qui permettent et facilitent un transfert non perturbant de prise en charge de systèmes de fichiers individuels entre serveurs. Elles sont toutes fondées sur des attributs qui permettent à un système de fichiers de

spécifier des informations de localisation de remplacement, supplémentaires, et nouvelles qui spécifient comment le client peut accéder à ce système de fichiers.

Ces attributs peuvent être utilisés pour fournir des systèmes de fichiers individuels actifs :
* Des adresses réseau de remplacement pour accéder à l'instance actuelle de système de fichiers.
* Les localisations des instances de remplacement de système de fichiers ou des répliques à utiliser au cas où l'instance actuelle de système de fichiers deviendrait indisponible.

Ces attributs de localisation de système de fichiers peuvent être utilisés avec le concept de systèmes de fichiers absent, dans lequel une position dans l'espace de noms de serveur est associée à des localisations sur d'autres serveurs sans qu'il y ait d'instance de système de fichiers correspondante sur le serveur actuel. Par exemple,
* ces attributs peuvent être utilisés avec des systèmes de fichiers absents pour mettre en œuvre des références par lesquelles un serveur peut diriger le client sur un système de fichiers fourni par un autre serveur. Cela permet de construire des espaces de noms multi-serveurs extensifs.
* Ces attributs peuvent être fournis quand un système de fichiers précédemment présent devient absent. Cela permet une migration non perturbante de systèmes de fichiers sur des serveurs de remplacement.

1.8.4 Facilités de verrouillage

Comme mentionné précédemment, NFSv4.1 est un seul protocole qui inclut des facilités de verrouillage. Ces facilités de verrouillage incluent la prise en charge de nombreux types de verrous incluant un certain nombre de sortes de verrous rappelables. Les verrous rappelables comme des délégations permettent au client d'être assuré que certains événements ne vont pas se produire tant que ce verrou est tenu. Quand les circonstances changent, le verrou est rappelé via une demande de rappel. Les assurances fournies par les délégations permettent de faire une mise en antémémoire plus extensive en toute sécurité quand les circonstances le permettent.

Les types de verrous sont :
* Des partages de réservations comme établi par les opérations OPEN.
* Des verrous de gamme d'octets.
* Des délégations de fichier, qui sont des verrous rappelables qui assurent au détenteur que des ouvertures incohérentes et des changements de fichier ne peuvent pas se produire tant que la délégation est conservée.
* Des délégations de répertoire, qui sont des verrous rappelables qui assurent au détenteur que des modifications de répertoire incohérentes ne peuvent pas se produire tant que la délégation est conservée.
* Des dispositions, qui sont des objets rappelables qui assurent au détenteur qu'un accès direct aux données du fichier peut être effectué directement par le client et qu'aucun changement de la localisation des données incohérent avec cet accès ne peut être fait tant que la disposition est conservée.

Tous les verrous pour un client donné sont liés ensemble sous un seul prêt au niveau du client. Toutes les demandes faites sur les sessions associées au client renouvellent ce prêt. Quand le prêt du client n'est pas renouvelé rapidement, les verrous du client sont sujets à révocation. En cas de redémarrage du serveur, les clients ont l'opportunité de réclamer en toute sécurité leurs verrous dans une période de grâce spéciale.

1.9 Différences avec NFSv4.0

On résume ci-après les différences majeures entre la version mineure 1 et le protocole de base :
* Mise en œuvre du modèle de sessions (paragraphe 2.10).
* Accès parallèle aux données (Section 12).
* Ajout de l'opération RECLAIM_COMPLETE pour mieux structurer le processus de réclamation de verrou (paragraphe 18.51).
* Amélioration de la prise en charge de délégation comme suit :
  - Délégations sur des répertoires et autres types de fichiers en plus des fichiers réguliers (paragraphes 18.39, 18.49).
  - Opérations pour optimiser l'acquisition de délégations rappelées ou refusées (paragraphes 18.49, 20.5, 20.7).
  - Notifications des changements aux fichiers et répertoires (paragraphes 18.39, 20.4).
  - Méthode pour permettre à un serveur d'indiquer qu'il rappelle une ou plusieurs délégations pour des raisons de gestion de ressources, et donc une méthode pour permettre au client de prendre les délégations à retourner (paragraphe 20.6).
* Les attributs peuvent être réglés individuellement durant la création de fichier exclusive via l'opération OPEN (voir la nouvelle méthode de création EXCLUSIVE4_1 au paragraphe 18.16).
* Les fichiers ouverts peuvent être préservés si ils sont supprimés et le compte de liaisons dures *(hard link)* ("hard link"

est défini dans une norme Open Group [IEEE-1003.1]) tombe à zéro, supprimant donc le besoin que les clients renomment les fichiers supprimés en noms partiellement cachés -- appelés familièrement "renommage stupide" (voir le nouveau fanion de réponse OPEN4_RESULT_PRESERVE_UNLINKED au paragraphe 18.16).
* Compatibilité améliorée avec Microsoft Windows pour les listes de contrôle d'accès (paragraphes 6.2.3, 6.2.2, 6.4.3.2).
* Rétention des données (paragraphe 5.13).
* Identification de la mise en œuvre de client et serveur NFS (paragraphe 18.35).
* Prise en charge de la notification de disponibilité des verrous de gamme d'octets (voir le nouveau fanion de réponse OPEN4_RESULT_MAY_NOTIFY_LOCK au paragraphe 18.16 et voir le paragraphe 20.11).
* Dans NFSv4.1, LIPKEY et SPKM-3 ne sont pas des mécanismes de sécurité exigés [RFC2847].

2. Infrastructure centrale

2.1 Introduction

NFSv4.1 s'appuie sur un cœur d'infrastructure commun à presque chaque opération. Ce cœur d'infrastructure est décrit dans la suite de cette section.

2.2 RPC et XDR

Le protocole NFSv4.1 est une application d'appel de procédure distante (RPC, *Remote Procedure Call*) qui utilise RPC version 2 et la représentation de données externes (XDR, *eXternal Data Representation*) correspondante comme défini dans les [RFC5531] et [RFC4506].

2.2.1 Sécurité fondée sur RPC

Les versions précédentes de NFS avaient été conçues selon un modèle d'authentification fondé sur l'hôte, où le serveur NFS authentifie le client NFS, et fait confiance au client pour authentifier tous les utilisateurs. En fait, NFS a toujours dépendu de RPC pour l'authentification. Une des premières formes d'authentification RPC, AUTH_SYS, n'avait pas d'authentification forte et exigeait une approche d'authentification fondée sur l'hôte. NFSv4.1 dépend aussi de RPC pour les services de sécurité de base et rend obligatoire la prise en charge de RPC pour le modèle d'authentification fondé sur l'utilisateur. Le modèle d'authentification fondé sur l'utilisateur a les principaux utilisateurs authentifiés par un serveur, et le serveur est à son tour authentifié par les principaux utilisateurs. RPC fournit des services de sécurité de base qui sont utilisés par NFSv4.1.

2.2.1.1 Nuances de sécurité RPC

Comme décrit à la Section 7 "Authentification", de la [RFC5531], la sécurité de RPC est encapsulée dans l'en-tête RPC, via une nuance de sécurité ou d'authentification, et des informations spécifiques de la nuance de sécurité spécifiée. Chaque en-tête RPC porte les informations utilisées pour identifier et authentifier un client et un serveur. Comme discuté au paragraphe 2.2.1.1.1, certaines nuances de sécurité fournissent des services de sécurité supplémentaires.

Les clients et serveurs NFSv4.1 DOIVENT mettre en œuvre RPCSEC_GSS. (Cette exigence de mise en œuvre n'est pas l'exigence d'utiliser.) D'autres nuances, comme AUTH_NONE et AUTH_SYS, PEUVENT aussi être mises en œuvre.

2.2.1.1.1 RPCSEC_GSS et services de sécurité

RPCSEC_GSS [RFC2203] utilise la fonctionnalité de GSS-API [RFC2743]. Cela permet l'utilisation de divers mécanismes de sécurité par la couche RPC sans les frais généraux supplémentaires de mise en œuvre de l'ajout de nuances de sécurité RPC.

2.2.1.1.1.1 Identification, authentification, intégrité, confidentialité

Via la GSS-API, RPCSEC_GSS peut être utilisé pour identifier et authentifier aux serveurs les utilisateurs sur les clients, et les serveurs aux utilisateurs. Il peut aussi effectuer la vérification d'intégrité sur le message RPC entier, y compris l'en-tête RPC, et sur les arguments ou résultats. Finalement, la confidentialité, généralement via le chiffrement, est un service disponible avec RPCSEC_GSS. La confidentialité est effectuée sur les arguments et résultats. Noter que si la confidentialité est choisie, l'intégrité, l'authentification, et l'identification sont activées. Si la confidentialité n'est pas choisie, mais si l'intégrité est sélectionnée, l'authentification et l'identification sont activées. Si l'intégrité et la confidentialité ne sont pas retenues, mais si l'authentification est activée, l'identification est activée. RPCSEC_GSS ne fournit pas l'identification

comme service séparé.

Bien que GSS-API ait un service d'authentification distinct de ses services de confidentialité et d'intégrité, le service d'authentification de GSS-API n'est pas utilisé pour le service d'authentification de RPCSEC_GSS. À la place, chaque en-tête RPC de demande et réponse est protégé en intégrité avec le service d'intégrité de GSS-API, et cela permet à RPCSEC_GSS d'offrir l'authentification et l'identité par RPC. Voir plus d'informations dans la [RFC2203].

Les clients et serveurs NFSv4.1 DOIVENT prendre en charge le service d'intégrité et d'authentification de RPCSEC_GSS. Les serveurs NFSv4.1 DOIVENT prendre en charge le service de confidentialité de RPCSEC_GSS. Les clients NFSv4.1 DEVRAIENT prendre en charge le service de confidentialité de RPCSEC_GSS.

2.2.1.1.1.2 Mécanismes de sécurité pour NFSv4.1

RPCSEC_GSS, via GSS-API, normalise l'accès aux mécanismes qui fournissent des services de sécurité. Donc, les clients et serveurs NFSv4.1 DOIVENT prendre en charge le mécanisme de sécurité Kerberos V5.

L'utilisation de RPCSEC_GSS exige le choix d'un mécanisme, d'une qualité de protection (QOP) et de service (authentification, intégrité, confidentialité). Pour les mécanismes de sécurité obligatoires, NFSv4.1 spécifie qu'une QOP de zéro est utilisée, laissant au mécanisme ou à la configuration du mécanisme de transposer la QOP zéro en un niveau approprié de protection. Chaque mécanisme obligatoire spécifie un ensemble minimum d'algorithmes de chiffrement pour mettre en œuvre l'intégrité et la confidentialité. Les clients et serveurs NFSv4.1 DOIVENT être mis en œuvre sur des environnements de fonctionnement qui se conforment aux algorithmes de chiffrement EXIGÉS de chaque mécanisme EXIGÉ.

2.2.1.1.1.2.1 Kerberos V5

Le mécanisme GSS-API de Kerberos V5 comme décrit dans la [RFC4121] DOIT être mis en œuvre avec les services RPCSEC_GSS spécifiés dans le tableau suivant :

Descriptions des colonnes :
1 = numéro de la pseudo nuance
2 = nom de la pseudo nuance
3 = OID du mécanisme
4 = service RPCSEC_GSS
5 = les clients NFSv4.1 DOIVENT prendre en charge
6 = les serveurs NFSv4.1 DOIVENT prendre en charge

1	2	3	4	5	6
390003	krb5	1.2.840.113554.1.2.2	rpc_gss_svc_none	oui	oui
390004	krb5i	1.2.840.113554.1.2.2	rpc_gss_svc_integrity	oui	oui
390005	krb5p	1.2.840.113554.1.2.2	rpc_gss_svc_privacy	non	oui

Noter que le numéro et le nom de la pseudo nuance sont présentés ici comme une aide à la transposition pour la mise en œuvre. Parce que le protocole NFSv4.1 inclut une méthode pour négocier la sécurité et qu'il comprend le mécanisme GSS-API, la pseudo nuance n'est pas nécessaire. La pseudo nuance est nécessaire pour NFSv3 car la négociation de sécurité est faite via le protocole MOUNT comme décrit dans la [RFC2623].

Au moment de la spécification de NFSv4.1, la norme de chiffrement évolué (AES, *Advanced Encryption Standard*) avec HMAC-SHA1 était un algorithme EXIGÉ pour Kerberos V5. À l'opposé, quand NFSv4.0 a été spécifié, en ensemble d'algorithmes plus faibles était EXIGÉ pour Kerberos V5, et était EXIGÉ dans la spécification NFSv4.0, parce que la spécification de Kerberos V5 à ce moment ne spécifiait pas d'algorithmes plus forts. La spécification NFSv4.1 ne spécifie pas d'algorithmes EXIGÉS pour Kerberos V5, et à la place, la mise en œuvre est supposée suivre l'évolution de la norme Kerberos V5 si et quand des algorithmes plus forts sont spécifiés.

2.2.1.1.1.2.1.1 Considérations sur la sécurité des algorithmes cryptographiques dans Kerberos V5

Quand on déploie NFSv4.1, la force de la sécurité réalisée dépend de l'infrastructure Kerberos V5 existante. Les algorithmes de Kerberos V5 ne sont pas directement exposés ou choisis par le client ou le serveur, de sorte qu'une certaine diligence est exigée de l'utilisateur de NFSv4.1 pour s'assurer que la sécurité est acceptable lorsque nécessaire.

2.2.1.1.1.3 Principal serveur GSS

Sans considération du mécanisme de sécurité utilisé avec RPCSEC_GSS, le serveur NFS DOIT s'identifier dans GSS-API via un type de nom GSS_C_NT_HOSTBASED_SERVICE. Les noms GSS_C_NT_HOSTBASED_SERVICE sont de la forme "service@nom_d'hôte".

Pour NFS, l'élément "service" est nfs.

Les mises en œuvre de mécanismes de sécurité vont convertir nfs@nom_d'hôte en diverses formes différentes. Pour Kerberos V5, la forme suivante est RECOMMANDÉE : nfs/nom_d'hôte

2.3 COMPOUND et CB_COMPOUND

Une différence significative d'avec les versions du protocole NFS d'avant NFSv4 est l'introduction de la procédure COMPOUND. Pour le protocwole NFSv4, dans toutes les versions mineures, il y a exactement deux procédures RPC, NULL et COMPOUND. La procédure COMPOUND est définie comme une série d'opérations individuelles et ces opérations effectuent les sortes de fonctions effectuées par les procédures NFS traditionnelles.

Les opérations combinées au sein d'une demande COMPOUND sont évaluées dans l'ordre par le serveur, sans aucune garantie d'atomicité. Un ensemble limité de facilités existe pour passer les résultats d'une opération à l'autre. Une fois qu'une opération retourne un résultat d'échec, l'évaluation se termine et les résultats de toutes les opérations évaluées sont retournés au client.

Avec l'utilisation de la procédure COMPOUND, le client est capable de construire des demandes simples ou complexes. Ces demandes COMPOUND permettent une réduction du nombre de RPC nécessaires pour les opérations logiques de système de fichiers. Par exemple, des demandes de recherche multi-composants peuvent être construites en combinant plusieurs opérations LOOKUP. Celles-ci peuvent être encore combinées avec des opérations telles que GETATTR, READDIR, ou OPEN plus READ pour faire des ensembles d'opérations plus compliqués sans subir de latence supplémentaire.

NFSv4.1 contient aussi un ensemble considérable d'opérations de rappel dans lesquelles le serveur fait un RPC dirigé sur le client. Les RPC de rappel ont une structure similaire à celles des demandes normales au serveur. Dans toutes les versions mineures du protocole NFSv4, il y a deux procédures de rappel RPC : CB_NULL et CB_COMPOUND. La procédure CB_COMPOUND est définie de façon analogue à celle de COMPOUND avec son propre ensemble d'opérations de rappel.

L'ajout de nouvelles opérations de serveur et de rappel au sein du cadre de demande COMPOUND et CB_COMPOUND fournit un moyen d'étendre le protocole dans les versions mineures suivantes.

Sauf pour un petit nombre d'opérations nécessaires pour la création de session, les demandes de serveur et les demandes de rappel sont effectuées dans le contexte d'une session. Les sessions fournissent au client un contexte pour chaque demande et prennent en charge une protection robuste contre la répétition pour les demandes non idempotentes.

2.4 Identifiants de client et propriétaires de client

Pour chaque opération qui obtient ou dépend de l'état de verrouillage, le client spécifique doit être identifiable par le serveur.

Chaque instance distincte de client est représentée par un identifiant de client. Un identifiant de client est une chaîne de 64 bits représentant un client spécifique à un moment donné. L'identifiant de client est changé chaque fois que le client ré-initialise, et peut changer quand le serveur ré-initialise. Les identifiants de client sont utilisés pour prendre en charge l'identification de verrou et la récupération de panne.

Durant le fonctionnement en régime permanent, l'identifiant de client associé à chaque opération est déduit de la session (voir le paragraphe 2.10) sur laquelle l'opération est envoyée. Une session est associée à un identifiant de client quand la session est créée.

À la différence de NFSv4.0, les seules opérations NFSv4.1 possibles avant l'établissement d'un identifiant de client sont celles nécessaires pour établir l'identifiant de client.

Une séquence d'une opération EXCHANGE_ID suivie par une opération CREATE_SESSION utilisant cet identifiant de

client (eir_clientid comme retourné de EXCHANGE_ID) est exigée pour établir et confirmer l'identifiant de client chez le serveur. L'établissement de l'identification par une nouvelle incarnation du client a aussi pour effet de libérer immédiatement tout état de verrouillage qu'une incarnation précédente de ce même client pourrait avoir eu sur le serveur. Un tel état libéré inclurait tout verrou de gamme d'octets, réservation partagée, état de disposition, et -- où le serveur ne prend en charge les types de réclamation ni CLAIM_DELEGATE_PREV ni CLAIM_DELEG_CUR_FH -- tout état de délégation associé au même client avec la même identité. Pour une discussion de la récupération d'état de délégation, voir le paragraphe 10.2.1. Pour une discussion de la récupération d'état de disposition, voir le paragraphe 12.7.1.

La libération d'un tel état exige que le serveur soit capable de déterminer qu'une instance de client est le successeur d'une autre. Lorsque cela ne peut pas être fait, pour un certain nombre de raisons, l'état de verrouillage va demeurer pour un temps qui dépend de l'expiration du prêt (voir le paragraphe 8.3) et le nouveau client devra attendre que cet état soit supprimé, si il fait des demandes de verrou qui sont en conflit avec lui.

L'identification de client est encapsulée dans le type de données de propriétaire de client suivant :

```
struct client_owner4 {
        verifier4       co_verifier;
        opaque          co_ownerid<NFS4_OPAQUE_LIMIT>;
};
```

Le premier champ, co_verifier, est un vérificateur d'incarnation de client, qui permet au serveur de distinguer les incarnations successives (par exemple, réamorçages) du même client. Le serveur va commencer le processus d'annulation de l'état de prêt du client si co_verifier est différent de ce que le serveur avait précédemment enregistré pour le client identifié (comme spécifié dans le champ co_ownerid).

Le second champ, co_ownerid, est une chaîne de longueur variable qui définit de façon univoque le client de telle sorte que les instances suivantes du même client portent le même co_ownerid avec un vérificateur différent.

Il y a plusieurs considérations sur comment le client génère la chaîne co_ownerid :

* La chaîne devrait être unique afin que plusieurs clients ne présentent pas la même chaîne. Les conséquences de deux clients présentant la même chaîne vont de ce que un client obtient une erreur à un client qui a son état de prêt annulé de façon abrupte et inattendue.

* La chaîne devrait être choisie de telle façon que les incarnations suivantes (par exemple, les redémarrages) du même client causent la présentation de la même chaîne par le client. On devrait se défier d'une approche qui exigerait que la chaîne soit enregistrée dans un fichier local parce que cela empêcherait l'utilisation de la mise en œuvre dans un environnement où il n'y a pas de disque local et où tout l'accès au fichier est par un serveur NFSv4.1.

* La chaîne devrait être la même pour chaque adresse réseau de serveur à laquelle accède le client. De cette façon, si un serveur a plusieurs interfaces, le client peut multiplexer le trafic sur plusieurs chemins de réseau comme décrit au paragraphe 2.10.5. (Note : l'exact opposé était conseillé dans la spécification NFSv4.0 [RFC3530].)

* L'algorithme pour générer la chaîne ne devrait pas supposer que l'adresse réseau du client ne va pas changer, sauf si la mise en œuvre de client sait qu'il utilise des adresses réseau allouées de façon statique. Cela inclut des changements entre incarnations de client et même des changements alors que le client fonctionne encore dans son incarnation actuelle. Donc, avec l'allocation dynamique d'adresse, si le client inclut juste l'adresse réseau du client dans la chaîne co_ownerid, il y a un risque réel qu'après que le client ait abandonné l'adresse réseau, un autre client, utilisant un algorithme similaire pour générer la chaîne co_ownerid, génère une chaîne co_ownerid en conflit.

Étant données les considérations ci-dessus, un exemple de chaîne co_ownerid bien générée est celui qui inclut :

* Si applicable, l'adresse réseau allouée statiquement du client.

* Toutes les informations supplémentaires qui tendent à être uniques, comme un ou plusieurs de :
  - le numéro de série de la machine du client (pour des raisons de confidentialité, il vaut mieux effectuer une fonction unidirectionnelle sur le numéro de série).
  - Une adresse de contrôle d'accès au support (MAC, *Media Access Control*) (là encore une fonction unidirectionnelle devrait être effectuée).
  - L'horodatage de quand le logiciel NFSv4.1 a été installé sur le client (bien que ceci soit sujet à l'avertissement sus mentionné sur l'utilisation d'informations mémorisées dans un fichier, parce que le fichier pourrait être seulement

accessible sur NFSv4.1).
- Un vrai nombre aléatoire. Cependant, comme ce nombre devrait être le même entre les incarnations de client, cela pose le même problème que d'utiliser l'horodatage de l'installation du logiciel.

* Pour un client NFSv4.1 de niveau utilisateur, il devrait contenir des informations supplémentaires pour distinguer le client d'autres clients de niveau utilisateur fonctionnant sur le même hôte, comme un identifiant de processeur ou autre séquence unique.

L'identifiant de client est alloué par le serveur (le résultat eir_clientid provenant de l'EXCHANGE_ID) et devrait être choisi de façon à n'être pas en conflit avec un identifiant de client alloué précédemment par le serveur. Cela s'applique à travers les redémarrages de serveur.

Dans le cas d'un redémarrage de serveur, un client peut découvrir que son identifiant de client actuel n'est plus valide quand il reçoit une erreur NFS4ERR_STALE_CLIENTID. Les circonstances précises dépendent des caractéristiques des sessions impliquées, en particulier si la session est persistante (voir le paragraphe 2.10.6.5) mais dans chaque cas, le client va recevoir cette erreur quand il tente d'établir une nouvelle session avec l'identifiant de client existant et qu'il reçoit l'erreur NFS4ERR_STALE_CLIENTID, indiquant qu'un nouvel identifiant de client doit être obtenu via un EXCHANGE_ID et la nouvelle session établie avec cet identifiant de client.

Quand une session n'est pas persistante, le client va découvrir qu'il a besoin de créer une nouvelle session suite à l'obtention d'un NFS4ERR_BADSESSION, car la session en question a été perdue au titre d'un redémarrage du serveur. Quand l'identifiant de client existant est présenté à un serveur au titre de la création d'une session et que l'identifiant de client n'est pas reconnu, comme cela va se produire après un redémarrage de serveur, le serveur va rejeter la demande avec l'erreur NFS4ERR_STALE_CLIENTID.

Dans le cas d'une session persistante, le client va rétablir la communication en utilisant la session existante après le redémarrage. Cette session va être associée à l'identifiant de client existant mais peut seulement être utilisée pour retransmettre des opérations que le client a précédemment transmises et dont il n'a pas vu les réponses. Les réponses aux opérations que le serveur avait précédemment effectuées vont venir de l'antémémoire de réponse ; autrement, NFS4ERR_DEADSESSION va être retourné. Donc, une telle session est dite "morte". Dans cette situation, afin d'effectuer de nouvelles opérations, le client doit établir une nouvelle session. Si il tente d'établir cette nouvelle session avec l'identifiant de client existant, le serveur va rejeter la demande avec NFS4ERR_STALE_CLIENTID.

Quand NFS4ERR_STALE_CLIENTID est reçu dans l'une de ces situations, le client a besoin d'obtenir un nouvel identifiant de client en utilisant l'opération EXCHANGE_ID, puis d'utiliser cet identifiant de client comme base d'une nouvelle session, et ensuite de procéder à toute autre récupération nécessaire pour le cas de redémarrage du serveur (voir le paragraphe 8.4.2). Voir les descriptions de EXCHANGE_ID (paragraphe 18.35) et CREATE_SESSION (paragraphe 18.36) pour la spécification complète de ces opérations.

2.4.1 Mise à niveau de NFSv4.0 à NFSv4.1

Pour faciliter la mise à niveau de NFSv4.0 à NFSv4.1, un serveur peut comparer une valeur de type de données client_owner4 dans un EXCHANGE_ID avec une valeur de type de données nfs_client_id4 qui a été établie en utilisant l'opération SETCLIENTID de NFSv4.0. Un serveur qui fait cela va permettre à un client mis à niveau d'éviter d'attendre l'expiration du prêt (c'est-à-dire, le prêt établi par le client d'instance NFSv4.0). Cela exige que la valeur du type de données client_owner4 soit construite de la même façon que le type de valeur de données nfs_client_id4. Si le contenu des dernières incluait l'adresse réseau du serveur (conformément aux recommandations de la spécification de NFSv4.0 [RFC3530]) et si le client NFSv4.1 ne souhaite pas utiliser un identifiant de client qui empêche le multiplexage, il devrait envoyer deux opérations EXCHANGE_ID. Le premier EXCHANGE_ID va avoir un client_owner4 égal au nfs_client_id4. Cela va supprimer l'état créé par le client NFSv4.0. Le second EXCHANGE_ID ne va pas avoir l'adresse réseau du serveur. L'état créé pour le second EXCHANGE_ID ne va pas devoir attendre l'expiration du prêt, parce qu'il n'y aura pas d'état à expirer.

2.4.2 Libération par le serveur d'un identifant de client

NFSv4.1 introduit une nouvelle opération appelée DESTROY_CLIENTID (paragraphe 18.50) que le client DEVRAIT utiliser pour détruire un identifiant de client dont il n'a plus besoin. Cela permet une libération en douceur, bilatérale, d'un identifiant de client. L'opération ne peut pas être utilisée si il y a des sessions associées à l'identifiant de client, ou un état avec un prêt non expiré.

Si le serveur détermine que le client détient un état non associé pour son identifiant de client (l'état associé inclut des sessions non révoquées, ouvertes, des verrous, des délégations, des dispositions, et vœux) le serveur PEUT choisir de libérer unilatéralement l'identifiant de client afin de conserver les ressources. Si le client contacte le serveur après cette libération, le serveur DOIT s'assurer que le client reçoit l'erreur appropriée afin qu'il utilise la séquence EXCHANGE_ID/CREATE_SESSION pour établir un nouvel identifiant de client. Le serveur devrait hésiter à libérer un identifiant de client car le travail résultant chez le client pour récupérer d'un tel événement va être la même charge que si le serveur avait eu une défaillance et redémarrait. Normalement, un serveur ne va pas libérer un identifiant de client sauf si il y a eu une non activité de nombreuses minutes sur ce client. Tant qu'il y a des sessions, ouvertures, verrous, délégations, dispositions, ou vœux, le serveur NE DOIT PAS libérer l'identifiant de client. Voir au paragraphe 2.10.13.1.4 la discussion sur la libération des sessions inactives.

2.4.3 Résolution des conflits de propriétaire de client

Quand le serveur obtient un EXCHANGE_ID pour un propriétaire de client qui n'a actuellement pas d'état, ou qui a un état mais dont le prêt a expiré, le serveur DOIT permettre le EXCHANGE_ID et confirmer le nouvel identifiant de client si il est suivi par le CREATE_SESSION approprié.

Quand le serveur obtient un EXCHANGE_ID pour une nouvelle incarnation de propriétaire de client qui a actuellement une vieille incarnation avec un état et un prêt non expiré, il est permis au serveur de disposer de l'état de l'incarnation précédente de propriétaire de client si une des conditions suivantes est vraie :

* Le principal qui a créé l'identifiant de client pour le propriétaire de client est le même que le principal qui envoie l'opération EXCHANGE_ID. Noter que si l'identifiant de client a été créé avec la protection d'état SP4_MACH_CRED (paragraphe 18.35) le principal DOIT être fondé sur l'authentification RPCSEC_GSS, le service RPCSEC_GSS utilisé DOIT être intégrité ou confidentialité, et le même mécanisme et principal GSS DOIT être utilisé que celui utilisé quand l'identifiant de client a été créé.

* L'identifiant de client a été établi avec la protection SP4_SSV (paragraphes 18.35, 2.10.8.3) et le client envoie le EXCHANGE_ID avec la nuance de sécurité réglée à RPCSEC_GSS en utilisant le mécanisme GSS SSV (paragraphe 2.10.9).

* L'identifiant de client a été établi avec la protection SP4_SSV, et dans les conditions décrites ici, le EXCHANGE_ID a été envoyé avec la protection d'état SP4_MACH_CRED. Parce que le SSV pourrait ne pas persister à travers les redémarrages de client et serveur, et parce que la première fois qu'un client envoie un EXCHANGE_ID à un serveur il n'a pas de SSV, le client PEUT envoyer le EXCHANGE_ID suivant dans une bride de SSV RPCSEC_GSS. À la place, comme avec la protection SP4_MACH_CRED, le principal DOIT se fonder sur l'authentification RPCSEC_GSS, le service RPCSEC_GSS utilisé DOIT être l'intégrité ou la confidentialité, et le même mécanisme GSS et principal DOIT être utilisé que utilisé quand l'identifiant de client a été créé.

Si aucune de ces situations ne s'applique, le serveur DOIT retourner NFS4ERR_CLID_INUSE.

Si le serveur accepte le principal et co_ownerid comme correspondant à celui qui a créé l'identifiant de client, et si le co_verifier dans l'EXCHANGE_ID diffère du co_verifier utilisé quand l'identifiant de client a été créé, alors après que le serveur reçoit un CREATE_SESSION qui confirme l'identifiant de client, le serveur supprime l'état. Si les valeurs de co_verifier sont les mêmes (par exemple, si le client met à jour les propriétés de l'identifiant de client (paragraphe 18.35) ou tente de multiplexer (paragraphe 2.10.5) le serveur NE DOIT PAS supprimer l'état.

2.5 Propriétaires de serveur

Le propriétaire de serveur est similaire à un propriétaire de client (paragraphe 2.4) mais à la différence du propriétaire de client, il n'y a pas d'identifiant de serveur abrégé. Le propriétaire de serveur est défini par le type de données suivant :

```
struct server_owner4 {
  uint64_t      so_minor_id;
  opaque        so_major_id<NFS4_OPAQUE_LIMIT>;
};
```

Le propriétaire de serveur est retourné de l'EXCHANGE_ID. Quand les champs so_major_id sont les mêmes dans deux résultats d'EXCHANGE_ID, les connexions sur lesquelles chaque EXCHANGE_ID a été envoyé peuvent être supposées

viser le même serveur (comme défini au paragraphe 1.7). Si les champs so_minor_id sont aussi les mêmes, alors non seulement les deux connexions se connectent au même serveur, mais la session peut être partagée à travers les deux connexions. Le lecteur est averti que plusieurs serveurs peuvent délibérément ou accidentellement prétendre avoir le même so_major_id ou so_major_id/so_minor_id ; le lecteur devrait examiner les paragraphes 2.10.5 et 18.35 afin d'éviter d'agir sur des valeurs de propriétaire de serveur faussement correspondantes.

Les considérations sur la génération d'un so_major_id sont similaires à celles sur la génération d'une chaîne co_ownerid (voir le paragraphe 2.4). Les conséquences de la génération par deux serveurs de valeurs de so_major_id en conflit sont moins sévères que pour des conflits de co_ownerid parce que le client peut utiliser RPCSEC_GSS pour comparer l'authenticité de chaque serveur (voir le paragraphe 2.10.5).

2.6 Négociation de service de sécurité

Avec l'offre potentielle par le serveur NFSv4.1 de plusieurs mécanismes de sécurité, le client a besoin d'une méthode pour déterminer ou négocier le mécanisme à utiliser pour sa communication avec le serveur. Le serveur NFS peut avoir plusieurs points au sein de son espace de noms de système de fichiers disponibles pour être utilisés par les clients NFS. Ces points peuvent être considérés comme des frontières de politique de sécurité, et, dans certaines mises en œuvre de NFS, sont liés aux points d'exportation de NFS. À son tour, le serveur NFS peut être configuré de façon à ce que chacune de ces frontières de politique de sécurité puisse avoir des mécanismes de sécurité différents ou multiples à utiliser.

La négociation de sécurité entre client et serveur DEVRAIT être faite avec un canal sûr pour éliminer la possibilité qu'un tiers intercepte la séquence de négociation et force le client et le serveur à choisir un niveau de sécurité inférieur à celui exigé ou désiré. Voir plus de discussion à la Section 21.

2.6.1 Couples de sécurité NFSv4.1

Un serveur NFS peut allouer un ou plusieurs "couples de sécurité" à chaque frontière de politique de sécurité dans son espace de noms. Chaque couple de sécurité consiste en une nuance de sécurité (voir le paragraphe 2.2.1.1) et, si la nuance est RPCSEC_GSS, un identifiant d'objet de mécanisme GSS-API, une qualité de protection GSS-API, et un service RPCSEC_GSS.

2.6.2 SECINFO et SECINFO_NO_NAME

Les opérations SECINFO et SECINFO_NO_NAME permettent au client de déterminer, sur la base de la bride de fichier, quel couple de sécurité est à utiliser pour l'accès au serveur. En général, le client ne va pas avoir à utiliser une de ces opérations sauf durant la communication initiale avec le serveur ou quand le client traverse les frontières de politique de sécurité chez le serveur. Cependant, la politique du serveur peut aussi changer à tout moment et forcer le client à négocier un nouveau couple de sécurité.

Lorsque l'utilisation de différents couples de sécurité affecterait le type d'accès qui serait permis si une demande était envoyée sur la même connexion utilisée pour l'accès à l'opération SECINFO ou SECINFO_NO_NAME (par exemple, lecture-seule contre lecture-écriture) les couples de sécurité qui permettent plus d'accès devraient être présentés en premier. Lorsque le niveau général d'accès est le même et que différentes nuances de sécurité limitent la gamme de principaux dont les privilèges sont reconnus (par exemple, permettre ou interdire l'accès à la racine) les nuances qui supportent la plus grande gamme de principaux devraient être mentionnées les premières.

2.6.3 Erreur de sécurité

Sur la base de l'hypothèse que chaque client et serveur NFSv4.1 DOIT prendre en charge un ensemble minimum de sécurité (c'est-à-dire, Kerberos V5 sous RPCSEC_GSS) le client NFS va initier l'accès au fichier au serveur avec un des couples de sécurité minimal. Durant la communication avec le serveur, le client peut recevoir une erreur NFS de NFS4ERR_WRONGSEC. Cette erreur permet au serveur de notifier au client que le couple de sécurité actuellement utilisé contrevient à la politique du serveur. Le client est alors responsable de déterminer (voir le paragraphe 2.6.3.1) quels couples de sécurité sont disponibles chez le serveur et d'en choisir un approprié pour le client.

2.6.3.1 Utilisation de NFS4ERR_WRONGSEC, SECINFO, et SECINFO_NO_NAME

Ce paragraphe explique le mécanisme de la négociation de sécurité NFSv4.1.

2.6.3.1.1 Opérations "Mettre une bride de fichier"

Le terme de "opération mettre une bride de fichier" *(put filehandle operation)* se réfère à PUTROOTFH, PUTPUBFH, PUTFH, et RESTOREFH. Chacun des sous paragraphes suivant décrit comment le serveur traite une sous série d'opérations qui commencent avec une opération "mettre une bride de fichier".

2.6.3.1.1.1 Opération Mettre une bride de fichier + SAVEFH

Le client sauvegarde une bride de fichier pour une future RESTOREFH, LINK, ou RENAME. SAVEFH NE DOIT PAS retourner NFS4ERR_WRONGSEC. Pour déterminer si l'opération Mettre une bride de fichier retourne ou non NFS4ERR_WRONGSEC, la mise en œuvre de serveur prétend que SAVEFH n'est pas dans la série des opérations et examine lesquelles des situations décrites dans les autres sous paragraphes du paragraphe 2.6.3.1.1 s'appliquent.

2.6.3.1.1.2 Deux ou plus opérations Mettre une bride de fichier

Pour une série de N opérations Mettre une bride de fichier, le serveur NE DOIT PAS retourner NFS4ERR_WRONGSEC au N-1 premières opérations Mettre une bride de fichier. La Nième opération Mettre une bride de fichier est traitée comme si elle était la première d'une sous série d'opérations. Par exemple, si le serveur a reçu une demande COMPOUND avec cette série d'opérations -- PUTFH, PUTROOTFH, LOOKUP -- alors l'opération PUTFH est ignorée pour les besoins de NFS4ERR_WRONGSEC, et la sous série PUTROOTFH, LOOKUP est traitée conformément au paragraphe 2.6.3.1.1.3.

2.6.3.1.1.3 Opération Mettre une bride de fichier + LOOKUP (ou OPEN d'un nom existant)

Cette situation s'applique aussi à une opération Mettre une bride de fichier suivie par une opération LOOKUP ou OPEN qui spécifie un nom de composant existant.

Dans cette situation, le client est potentiellement en train de traverser une frontière de politique de sécurité, et l'ensemble de couples de sécurité que prend en charge le répertoire parent peut différer de celui du fils. La mise en œuvre de serveur peut décider d'imposer des restrictions à l'administration de politique de sécurité. Il y a au moins trois approches (sec_policy_child est l'ensemble d'export fils, sec_policy_parent est celui du parent).

(a) sec_policy_child <= sec_policy_parent (<= pour un sous ensemble). Cela signifie que l'ensemble des couples de sécurité spécifiés dans la politique de sécurité d'un répertoire fils est toujours un sous ensemble de son répertoire parent.

(b) sec_policy_child ^ sec_policy_parent != {} (^ pour intersection, {} pour l'ensemble vide). Cela signifie que l'ensemble des couples de sécurité spécifiés dans la politique de sécurité d'un répertoire fils a toujours une intersection non vide avec celui du parent.

(c) sec_policy_child ^ sec_policy_parent == {}. Cela signifie que l'ensemble des couples de sécurité spécifiés dans la politique de sécurité d'un répertoire fils ne peut pas avoir d'intersection avec celui du parent. En d'autres termes,il n'y a pas de restriction à la façon dont l'administrateur du système peut régler ces couples.

Afin qu'un serveur prenne en charge les approches (b) (pour le cas où un client choisit une nuance qui n'est pas membre de sec_policy_parent) et (c), l'opération Mettre une bride de fichier ne peut pas retourner NFS4ERR_WRONGSEC quand il y a une discordance de couple de sécurité. À la place, il devrait être retourné à partir du LOOKUP (ou OPEN par un nom de composant existant) qui suit.

Comme cette directive n'est pas en contradiction avec l'approche (a), elle devrait être suivie en général. Même si l'approche (a) est mise en œuvre, il est possible que le couple de sécurité utilisé soit acceptable pour la cible du LOOKUP mais pas pour les brides de fichier utilisées dans l'opération Mettre une bride de fichier. L'opération Mettre une bride de fichier pourrait être un PUTROOTFH ou PUTPUBFH, où le client ne peut pas connaître les couples de sécurité pour la racine ou la bride de fichier publique. Ou la politique de sécurité pour la bride de fichier utilisée par l'opération Mettre une bride de fichier pourrait avoir changé depuis le moment où la bride de fichier a été obtenue.

Donc, un serveur NFSv4.1 NE DOIT PAS retourner NFS4ERR_WRONGSEC en réponse à l'opération Mettre une bride de fichier si l'opération est immédiatement suivie par un LOOKUP ou un OPEN par nom de composant.

2.6.3.1.1.4 Opération Mettre une bride de fichier + LOOKUPP

Comme SECINFO fonctionne seulement en descendant, il n'y a pas de moyen pour que LOOKUPP puisse retourner NFS4ERR_WRONGSEC sans SECINFO_NO_NAME. SECINFO_NO_NAME résout ce problème via le style SECINFO_STYLE4_PARENT, qui fonctionne dans la direction opposée de SECINFO. Comme au paragraphe 2.6.3.1.1.3,

une opération Mettre une bride de fichier qui est suivie par un LOOKUPP NE DOIT PAS retourner NFS4ERR_WRONGSEC. Si le serveur ne prend pas en charge SECINFO_NO_NAME, le seul recours du client est d'envoyer la séquence d'opérations Mettre une bride de fichier, LOOKUPP, GETFH avec chaque couple de sécurité qu'il prend en charge.

Sans considération de si SECINFO_NO_NAME est pris ou non en charge, un serveur NFSv4.1 NE DOIT PAS retourner NFS4ERR_WRONGSEC en réponse à une opération Mettre une bride de fichier si l'opération est immédiatement suivie par un LOOKUPP.

2.6.3.1.1.5 Opération Mettre une bride de fichier + SECINFO/SECINFO_NO_NAME

Il est permis à un client sensible à la sécurité de choisir un couple de sécurité fort quand il interroge un serveur pour déterminer les couples de sécurité permis d'un objet de fichier. Le couple de sécurité choisi par le client n'a pas à être inclus dans la liste de couples de la politique de sécurité ni du répertoire parent indiqué dans l'opération Mettre une bride de fichier ni de l'objet de fichier fils indiqué dans SECINFO (ou tout répertoire parent indiqué dans SECINFO_NO_NAME). Bien sûr, le serveur doit être configuré pour tout couple de sécurité que le client choisit ; autrement, la demande va échouer à la couche RPC avec une erreur d'authentification appropriée.

En théorie, il n'y a pas de connexion entre la nuance de sécurité utilisée par SECINFO ou SECINFO_NO_NAME et celle prise en charge par la politique de sécurité. Mais en pratique, le client peut commencer à chercher des nuances fortes à partir de celles prises en charge par la politique de sécurité, suivies par celles de l'ensemble EXIGÉ.

Le serveur NFSv4.1 NE DOIT PAS retourner NFS4ERR_WRONGSEC à une opération Mettre une bride de fichier qui est immédiatement suivie par SECINFO ou SECINFO_NO_NAME. Le serveur NFSv4.1 NE DOIT PAS retourner NFS4ERR_WRONGSEC à partir de SECINFO ou SECINFO_NO_NAME.

2.6.3.1.1.6 Opération Mettre une bride de fichier + rien

Le serveur NFSv4.1 NE DOIT PAS retourner NFS4ERR_WRONGSEC.

2.6.3.1.1.7 Opération Mettre une bride de fichier + quelque chose d'autre

"Quelque chose d'autre" inclut OPEN par bride de fichier.

La politique de sécurité s'applique à la bride de fichier spécifiée dans l'opération Mettre une bride de fichier. Donc, l'opération Mettre une bride de fichier DOIT retourner NFS4ERR_WRONGSEC quand il y a une discordance de couple de sécurité. Cela évite la complexité d'ajouter NFS4ERR_WRONGSEC comme erreur permise à chaque autre opération.

Un COMPOUND contenant la série d'opérations Mettre une bride de fichier + SECINFO_NO_NAME (style SECINFO_STYLE4_CURRENT_FH) est pour le client une façon efficace de récupérer de NFS4ERR_WRONGSEC.

Le serveur NFSv4.1 NE DOIT PAS retourner NFS4ERR_WRONGSEC à une opération autre qu'une opération Mettre une bride de fichier, LOOKUP, LOOKUPP, et OPEN (par nom de composant).

2.6.3.1.1.8 Opérations après SECINFO et SECINFO_NO_NAME

Supposons qu'un client envoie une procédure COMPOUND contenant la série SEQUENCE, PUTFH, SECINFO_NONAME, READ, et supposons que le couple de sécurité utilisé ne corresponde pas à celui exigé pour le fichier cible. Selon la règle (voir le paragraphe 2.6.3.1.1.5) ni PUTFH ni SECINFO_NO_NAME ne peuvent retourner NFS4ERR_WRONGSEC. Selon la règle (voir le paragraphe 2.6.3.1.1.7) READ ne peut pas retourner NFS4ERR_WRONGSEC. Le problème est résolu par le fait que SECINFO et SECINFO_NO_NAME consomment la bride de fichier actuelle (noter que c'est un changement par rapport à NFSv4.0). Cela laisse une bride de fichier non actuelle à utiliser pour READ, et READ retourne NFS4ERR_NOFILEHANDLE.

2.6.3.1.2 LINK et RENAME

Les opérations LINK et RENAME utilisent les deux brides de fichier courante et sauvegardée. Techniquement, le serveur PEUT retourner NFS4ERR_WRONGSEC à partir de LINK ou RENAME si la politique de sécurité de la bride de fichier sauvegardée rejette la nuance de sécurité utilisée dans les accréditifs de la demande COMPOUND. Si le serveur fait ainsi, alors si il n'y a pas d'intersection entre les politiques de sécurité des brides de fichier sauvegardée et courante, cela signifie qu'il va être impossible au client d'effectuer l'opération LINK ou RENAME prévue.

Par exemple, supposons que le client envoie cette demande COMPOUND : SEQUENCE, PUTFH bFH, SAVEFH, PUTFH aFH, RENAME "c" "d", où les brides de fichier bFH et aFH se réfèrent à différents répertoires. Supposons qu'il existe un couple de sécurité non commun entre les politiques de sécurité de aFH et bFH. Si le client envoie la demande en utilisant des accréditifs acceptables à la politique de sécurité de bFH mais pas pour la politique de aFH, alors l'opération PUTFH de aFH va échouer avec NFS4ERR_WRONGSEC. Après une demande SECINFO_NO_NAME, le client envoie SEQUENCE, PUTFH bFH, SAVEFH, PUTFH aFH, RENAME "c" "d", en utilisant des accréditifs acceptables pour la politique de sécurité de aFH mais pas pour la politique de bFH. Le serveur retourne NFS4ERR_WRONGSEC sur l'opération RENAME.

Pour empêcher un client de faire une séquence sans fin de demandes contenant LINK ou RENAME, suivie par une demande contenant SECINFO_NO_NAME ou SECINFO, le serveur DOIT détecter quand les politiques de sécurité des brides de fichier courante et sauvegardée ont des couples de sécurité non mutuellement acceptables, et NE DOIT PAS retourner NFS4ERR_WRONGSEC à partir de LINK ou RENAME dans cette situation. À la place, le serveur DOIT faire une des deux choses suivantes :

* Le serveur peut retourner NFS4ERR_XDEV.
* Le serveur peut permettre que la politique de sécurité de la bride de fichier courante outrepasse celle de la bride de fichier sauvegardée, et donc retourne NFS4_OK.

2.7 Version mineure

Pour traiter l'exigence d'un protocole NFS qui peut évoluer en tant que de besoin, le protocole NFSv4.1 contient les règles et le cadre pour permettre de futurs changements ou version mineurs.

L'hypothèse de base par rapport aux versions mineures est que toute future version mineure acceptée va être documentée dans une ou plusieurs RFC sur la voie de la normalisation. La version mineure 0 du protocole NFSv4 est représentée par la [RFC3530], et la version mineure 1 est représentée par la présente RFC. Les procédures COMPOUND et CB_COMPOUND prennent en charge le codage de la version mineure demandée par le client.

Les éléments suivants représentent les règles de base pour le développement de versions mineures. Noter qu'une future version mineure peut modifier ou ajouter aux règles suivantes au titre de la définition de la version mineure.

1. Les procédures ne sont pas ajoutées ou supprimées. Pour maintenir le modèle RPC général, les versions mineures de NFSv4 ne vont pas ajouter ou supprimer des procédures du programme NFS.

2. Des versions mineures peuvent ajouter des opérations aux procédures COMPOUND et CB_COMPOUND. L'ajout d'opérations aux procédures COMPOUND et CB_COMPOUND n'affecte pas le modèle RPC.
   * Les versions mineures peuvent ajouter des attributs au bitmap4 qui représente les ensembles d'attributs et au fattr4 qui représente des ensembles de valeurs d'attribut. Cela permet l'expansion du modèle d'attribut en vue d'une future croissance ou adaptation.
   * La version mineure X doit ajouter tous les nouveaux attributs après le dernier attribut documenté. Comme les résultats d'attribut sont spécifiés comme un dispositif opaque de résultats codés en XDR par attribut, la complexité de l'ajout de nouveaux attributs au milieu des définitions actuelles serait trop lourde.

3. Les versions mineures ne doivent pas modifier la structure des arguments ou résultats d'une opération existante. Là encore, la complexité du traitement de multiples définitions de structure pour une seule opération est trop lourde. Les nouvelles opérations devraient être ajoutées plutôt que de modifier les structures existantes pour une version mineure. Cette règle n'empêche pas les adaptations suivantes dans une version mineure :
   * ajouter des bits aux champs de fanion, comme de nouveaux attributs au type de données bitmap4 de GETATTR, et fournir les variantes correspondantes des dispositifs opaques, comme un notify4 utilisé avec de tels bitmaps,
   * ajouter des bits aux attributs existants comme les ACL qui ont des mots fanions,
   * étendre des types énumérés (y compris NFS4ERR_*) avec de nouvelles valeurs,
   * ajouter des cas à une union commutée.

4. Les versions mineures ne doivent pas modifier la structure des attributs existants.

5. Les versions mineures ne doivent pas supprimer des opérations. Cela empêche la réutilisation potentielle d'un "créneau" d'opération particulière dans une future version mineure.

6. Les versions mineures ne doivent pas supprimer des attributs.

7. Les versions mineures ne doivent pas supprimer des bits fanions ou des valeurs d'énumération.

8. Les versions mineures peuvent déclarer qu'une opération NE DOIT PAS être mise en œuvre. Spécifier qu'une opération NE DOIT PAS être mise en œuvre est équivalent à rendre obsolète une opération. Pour le client, cela signifie que l'opération NE DOIT PAS être envoyée au serveur. Pour le serveur, une erreur NFS peut être retournée par opposition à "éliminer" la demande comme une erreur de décodage XDR. Cette approche permet l'obsolescence d'une opération tout en maintenant sa structure afin qu'une future version mineure puisse réintroduire l'opération.
   1. Les versions mineures peuvent déclarer qu'un attribut NE DOIT PAS être mis en œuvre.
   2. Les versions mineures peuvent déclarer qu'un bit fanion ou une valeur d'énumération NE DOIT PAS être mis en œuvre.

9. Les versions mineures peuvent dégrader des caractéristiques de EXIGÉ en RECOMMANDÉ, ou de RECOMMANDÉ en FACULTATIF.

10. Les versions mineures peuvent promouvoir des caractéristiques de FACULTATIF en RECOMMANDÉ, ou de RECOMMANDÉ à EXIGÉ.

11. Un client et un serveur qui prennent en charge la version mineure X DEVRAIENT aussi prendre en charge les versions mineures zéro à X-1.

12. Sauf pour des changements d'infrastructure, une version mineure ne doit pas introduire de nouvelles caractéristiques EXIGÉES. Cette règle permet l'introduction de nouvelles fonctionnalités et force l'utilisation d'une expérience de mise en œuvre avant de désigner une caractéristique comme EXIGÉE. Par ailleurs, certaines classes de caractéristiques sont infra-structurales et ont de larges effets. Permettre que des caractéristiques infra-structurales soient RECOMMANDÉES ou FACULTATIVES complique la mise en œuvre de la version mineure.

13. Un client NE DOIT PAS tenter d'utiliser un identifiant d'état, une bride de fichier, ou objet retourné similaire de la procédure COMPOUND avec une version mineure X pour une autre procédure COMPOUND avec la version mineure Y, où X != Y.

2.8 Services de sécurité non fondés sur RPC

Comme décrit au paragraphe 2.2.1.1.1.1, NFSv4.1 s'appuie sur RPC pour l'identification, l'authentification, l'intégrité, et la confidentialité. NFSv4.1 lui-même fournit ou permet des services de sécurité supplémentaires, comme décrit dans les sous paragraphes suivants.

2.8.1 Autorisation

L'autorisation d'accéder à un objet de fichier via une opération NFSv4.1 est en fin de compte déterminée par le serveur NFSv4.1. Un client peut prédéterminer son accès à un objet fichier via les opérations OPEN (paragraphe 18.16) et ACCESS (paragraphe 18.1).

Les principaux avec des droits d'accès appropriés peuvent modifier l'autorisation sur un objet fichier via l'opération SETATTR (paragraphe 18.30). Les attributs qui affectent les droits d'accès incluent mode, owner, owner_group, acl, dacl, et sacl. Voir à la Section 5.

2.8.2 Audit

NFSv4.1 fournit l'audit par objet de fichier, via les attributs acl et sacl comme décrit à la Section 6. Il sort du domaine d'application de la présente spécification de spécifier les formats d'enregistrement d'audit ou des politiques de gestion.

2.8.3 Détection d'intrusion

NFSv4.1 fournit le contrôle d'alarmes par objet de fichier, via les attributs acl et sacl comme décrit à la Section 6. Les alarmes peuvent servir de base de la détection d'intrusion. Il sort du domaine d'application de la présente spécification de spécifier les heuristiques pour détecter l'intrusion via des alarmes.

2.9 Couches de transport

2.9.1 Propriétés de transports exigées et recommandées

NFSv4.1 fonctionne sur des transports fondés sur l'accès direct à la mémoire distante (RDMA, *Remote Direct Memory Access*) et non RDMA avec les attributs suivants :

* Le transport prend en charge la livraison fiable de données, que NFSv4.1 exige mais ni NFSv4.1 ni RPC n'ont de facilités pour l'assurer [NFS-Serv].
* Le transport livre les données dans leur ordre d'envoi. La livraison ordonnée simplifie la détection des erreurs de transmission, et simplifie l'envoi de demandes et réponses de taille arbitraire via le protocole de marquage d'enregistrement [RFC5531].

Lorsque une mise en œuvre de NFSv4.1 prend en charge le fonctionnement sur le protocole de réseau IP, tout transport utilisé entre NFS et IP DOIT être un des protocoles de transport à contrôle d'encombrement approuvé par l'IETF. Au moment de la rédaction du présent document, les deux seuls transports qui ont les attributs ci-dessus sont TCP et le protocole de transmission de commandes de flux (SCTP, *Stream Control Transmission Protocol*). Pour améliorer la possibilité d'interopérabilité, une mise en œuvre de NFSv4.1 DOIT prendre en charge le fonctionnement sur le protocole de transport TCP.

Même si NFSv4.1 est utilisé sur un protocole réseau non IP, il est RECOMMANDÉ que le transport prenne en charge le contrôle d'encombrement.

Il est permis d'utiliser un transport sans connexion avec NFSv4.1 ; cependant, une livraison fiable et ordonnée des données combinée avec le contrôle d'encombrement par le transport sans connexion est EXIGÉE. Par conséquent, UDP par lui-même NE DOIT PAS être utilisé comme transport NFSv4.1. NFSv4.1 suppose qu'une adresse de transport de client et une adresse de transport de serveur utilisées pour envoyer des données sur un transport constituent ensemble une connexion, même si le transport sous-jacent évite le concept de connexion.

2.9.2 Comportement de transport du client et du serveur

Si un transport en mode connexion (par exemple, TCP) est utilisé, le client et le serveur DEVRAIENT utiliser des connexions de longue durée pour au moins trois raisons :

1. Cela va empêcher l'affaiblissement des mécanismes de contrôle d'encombrement du transport via des connexions de courte durée.
2. Cela va améliorer les performances pour l'environnement de WAN en éliminant le besoin des prises de contact de l'établissement de connexion.
3. Le modèle de rappel NFSv4.1 diffère de celui de NFSv4.0, et exige que le client et le serveur maintiennent un canal de retour créé par le client (voir le paragraphe 2.10.3.1) pour l'usage du serveur.

Afin de réduire l'encombrement, si un transport en mode connexion est utilisé, et si la demande n'est pas la procédure NULLE :

* Un demandeur NE DOIT PAS réesayer une demande sauf si la connexion sur laquelle la demande a été envoyée a été perdue avant la réception de la réponse.
* Un répondeur NE DOIT PAS éliminer en silence une demande, même si la demande est un ré-essai. (Le comportement d'élimination en silence de RPCSEC_GSS [RFC2203] ne s'applique pas parce que ce comportement se produit à la couche RPCSEC_GSS, une couche inférieure dans le traitement de demande.) À la place, le répondeur DEVRAIT retourner une erreur appropriée (voir le paragraphe 2.10.6.1) ou il PEUT déconnecter la connexion.

Quand il envoie une réponse, le répondeur DOIT envoyer la réponse à la même adresse réseau complète (par exemple, si on utilise un transport fondé sur IP, l'accès de source du demandeur fait partie de l'adresse réseau complète) d'où le demandeur a envoyé la demande. Si on utilise un transport en mode connexion, les réponses DOIVENT être envoyées sur la même connexion d'où la demande a été reçue.

Si une connexion est éliminée après que le répondeur a reçu la demande mais avant que le répondeur envoi la réponse, le répondeur pourrait avoir une réponse en instance. Si une connexion est établie avec la même adresse réseau complète de source et destination que la connexion éliminée, alors le répondeur NE DOIT PAS envoyer la réponse jusqu'à ce que le demandeur réessaye la demande. La raison de cette interdiction est que le demandeur PEUT réessayer une demande sur une connexion différente (pourvu que cette connexion soit associée à la session de la demande d'origine).

Quand on utilise des transports RDMA, il y a d'autres raisons pour ne pas tolérer des réessais sur la même connexion :

* Les transports RDMA utilisent des "crédits" pour appliquer le contrôle de flux, où un crédit est un droit pour un homologue de transmettre un message. Si un homologue devait retransmettre une demande (ou réponse) il consommerait un crédit supplémentaire. Si le répondeur retransmet une réponse, il va certainement en résulter une perte de connexion RDMA, car le demandeur va normalement seulement établir une seule mémoire tampon de réception pour chaque demande. Si le demandeur retransmet une demande, le crédit supplémentaire consommé sur le serveur pourrait conduire à la perte de la connexion RDMA sauf si le client comptait sur elle et a diminué son crédit disponible, conduisant à un gâchis de ressources.

* Les crédits RDMA posent un nouveau problème à la mémoire tampon de réponse dans NFSv4.1. La mémoire tampon de réponse peut être utilisée quand une connexion au sein d'une session est perdue, comme après que le client se reconnecte. Les informations de crédit sont une propriété dynamique de la connexion RDMA, et des valeurs périmées ne doivent pas être répétées à partir de la mémoire tampon. Cela implique que le contenu de la mémoire tampon ne doit pas être utilisé aveuglément quand des réponses sont envoyées à partir d'elle, et les informations de crédit appropriées pour le canal doivent être rafraîchies par la couche RPC.

De plus, comme décrit au paragraphe 2.10.6.2, lorsque une session est active, le demandeur NFSv4.1 NE DOIT PAS cesser d'attendre une réponse.

2.9.3 Accès

Historiquement, les serveurs NFSv3 ont écouté sur l'accès TCP 2049. L'accès enregistré 2049 [RFC3232] pour le protocole NFS devrait être la configuration par défaut. Les clients NFSv4.1 NE DEVRAIENT PAS utiliser les protocole de lien RPC comme décrit dans la [RFC1833].

2.10 Session

Les clients et serveurs NFSv4.1 DOIVENT prendre en charge et DOIVENT utiliser les caractéristiques de session comme décrit dans ce paragraphe.

2.10.1 Motif et généralités

Les versions et versions mineures précédentes de NFS ont souffert des défauts suivants :

* Absence de prise en charge de la sémantique de "exactement une fois" (EOS, *Exactly Once Semantics*). Cela inclut l'absence de prise en charge de EOS dans la défaillance et récupération du serveur.

* Prise en charge limitée du rappel, incluant la non prise en charge de l'envoi de rappels à travers les pare-feu, et les concurrences entre réponses aux demandes normales et rappels.

* Multiplexage limité sur plusieurs chemins de réseau.

* Exigence d'accréditifs de machine pour un fonctionnement pleinement sûr.

Par l'introduction d'une session, NFSv4.1 traite les défauts suivants avec des solutions pratiques :

* EOS est activé par une mémoire tampon de réponse d'une taille limitée, rendant faisable de garder la mémoire tampon dans une mémorisation persistante et permettant l'EOS à travers les défaillances et récupérations du serveur. Une raison pour laquelle les précédentes révisions de NFS ne prenaient pas en charge EOS était que certaines approches d'EOS limitaient souvent le parallélisme. Comme on l'explique au paragraphe 2.10.6, NFSv4.1 prend en charge à la fois EOS et un parallélisme sans limite.

* Le client NFSv4.1 (défini au paragraphe 1.7) crée des connexions de transport et les fournit au serveur qui les utilise pour envoyer les demandes de rappel, résolvant donc le problème du pare-feu (paragraphe 18.34). Les compétitions entre réponses provenant des demandes du client et les rappels causés par les demandes sont détectées via les propriétés de séquençage de la session qui sont la conséquence de EOS (paragraphe 2.10.6.3).

* Le client NFSv4.1 peut associer un nombre arbitraire de connexions à la session, et donc fournir le multiplexage (paragraphe 2.10.5).

* Le client et le serveur NFSv4.1 produisent une clé de session indépendante des accréditifs de machine du client et du serveur, qui peut être utilisée à calculer un résumé pour protéger les opérations critiques de gestion de session (paragraphe 2.10.8.3).

* Le client NFSv4.1 peut aussi créer des contextes RPCSEC_GSS sûrs à utiliser par le canal de retour de la session qui n'exigent pas que le serveur authentifie le principal de la machine cliente (paragraphe 2.10.8.2).

Une session est un objet de serveur de longue durée créé dynamiquement par un client et utilisé au fil du temps à partir d'une ou plusieurs connexions de transport. Sa fonction est de maintenir l'état du serveur par rapport à la ou les connexions appartenant à une instance de client. Cet état est entièrement indépendant de la connexion elle-même, et bien sûr, l'état existe que la connexion existe ou non. Un client peut avoir une ou plusieurs sessions associées à lui de sorte que l'état associé au client peut être accédé en utilisant toute session associée à l'identifiant de client de ce client, quand des connexions sont associées à ces sessions. Quand aucune connexion n'est associée à aucune session d'un identifiant de client pendant une longue période, les objets comme les verrous, ouvertures, délégations, dispositions, etc. sont sujets à expiration. La session sert d'objet représentant un moyen d'accès par un client à l'état de client associé sur le serveur, indépendamment des moyens physiques d'accès à cet état.

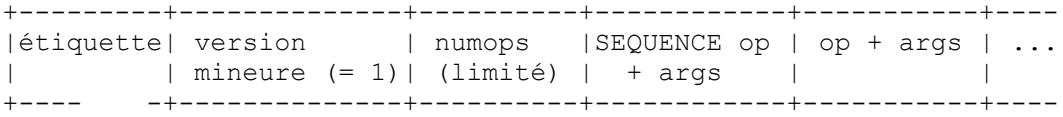
Un seul client peut créer plusieurs sessions. Une seule session NE DOIT PAS servir plusieurs clients.

2.10.2 Intégration de NFSv4

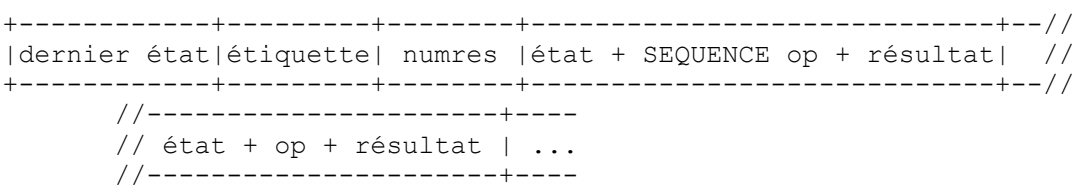
Les sessions font partie de NFSv4.1 et pas de NFSv4.0. Normalement, un changement majeur d'infrastructure comme les sessions exigerait un nouveau numéro de version majeure pour un programme RPC de Open Network Computing (ONC) comme NFS. Cependant, parce que NFSv4 encapsule sa fonction dans une seule procédure, COMPOUND, et parce que COMPOUND peut prendre en charge un nombre arbitraire d'opérations, les sessions ont été ajoutées à NFSv4.1 avec peu de difficultés. COMPOUND inclut un champ Numéro de version mineure, et pour NFSv4.1 cette version mineure est réglée à 1. Quand le serveur NFSv4 traite un COMPOUND avec la version mineure réglée à 1, il attend un ensemble différent d'opérations que pour NFSv4.0. NFSv4.1 définit l'opération SEQUENCE, qui est exigée pour chaque COMPOUND qui opère sur une session établie, à l'exception de certaines opérations d'administration de session, comme DESTROY_SESSION (paragraphe 18.37).

2.10.2.1 SEQUENCE et CB_SEQUENCE

Dans NFSv4.1, quand l'opération SEQUENCE est présente, elle DOIT être la première opération de la procédure COMPOUND. Le principal objet de SEQUENCE est de porter l'identifiant de session. L'identifiant de session associe toutes les autres opérations dans la procédure COMPOUND à une session particulière. SEQUENCE contient aussi les informations requises pour maintenir l'EOS (voir le paragraphe 2.10.6). Les demandes COMPOUND NFSv4.1 activées pour une session ont donc la forme :



et les réponses ont la forme :



Une procédure de demande et réponse CB_COMPOUND a une forme similaire à celle de COMPOUND, mais au lieu d'une opération SEQUENCE, il y a une opération CB_SEQUENCE. CB_COMPOUND a aussi un champ supplémentaire appelé "rappel_ident", qui est superflu dans NFSv4.1 et DOIT être ignoré par le client. CB_SEQUENCE a les mêmes informations que SEQUENCE, et inclut aussi d'autres informations nécessaires pour résoudre les conflits de rappel (paragraphe 2.10.6.3).

2.10.2.2 Identifiant de client et association de session

Chaque identifiant de client (paragraphe 2.4) peut avoir zéro, une ou plusieurs sessions actives. Un identifiant de client et la session associés sont obligés d'effectuer l'accès au fichier dans NFSv4.1. Chaque fois qu'une session est utilisée (par un client qui envoie une demande au serveur ou par le client qui répond à une demande de rappel du serveur) l'état prêté à son identifiant de client associé est automatiquement renouvelé.

L'état (qui peut consister en réservations partagées, verrous, délégations, et dispositions (paragraphe 1.8.4)) est lié à l'identifiant de client. L'état de client n'est pas lié à une session individuelle. Les opérations successives qui changent l'état d'un propriétaire d'état donné PEUVENT s'étendre sur différentes sessions, pourvu que la session soit associée au même identifiant de client. Un rappel PEUT arriver sur une session différente de celle de la demande qui a originellement acquis l'état appartenant au rappel. Par exemple, si la session A est utilisée pour acquérir une délégation, une demande de rappeler la délégation PEUT arriver sur la session B si les deux sessions sont associées au même identifiant de client. Les paragraphes 2.10.8.1 et 2.10.8.2 discutent des considérations de sécurité autour des rappels.

2.10.3 Canaux

Un canal n'est pas une connexion. Un canal représente la direction dans laquelle sont envoyées les demandes de RPC ONC.

Chaque session a un ou deux canaux : le canal avant et le canal de retour. Parce qu'il y a au plus deux canaux par session, et parce que chaque canal a un objet distinct, il n'est pas alloué d'identifiant aux canaux.

Le canal avant est utilisé pour les demandes ordinaires du client au serveur, et porte les demandes et réponses COMPOUND. Une session a toujours un canal avant.

Le canal de retour est utilisé pour les demandes de rappel du serveur au client, et porte les demandes et réponses CB_COMPOUND. Qu'il y ait ou non un canal de retour est décidé par le client ; cependant, de nombreuses caractéristiques de NFSv4.1 exigent un canal de retour. Les serveurs NFSv4.1 DOIVENT prendre en charge les canaux de retour.

Chaque session a des ressources pour chaque canal, incluant des mémoires tampons de réponse séparées (voir le paragraphe 2.10.6.1). Noter que même le canal de retour exige une mémoire tampon de réponse (ou, au moins, un tableau de créneaux afin de détecter les réessais) parce que certaines opérations de rappel ne sont pas idempotentes.

2.10.3.1 Association de connexions, canaux, et sessions

Chaque canal est associé à zéro, une ou plusieurs connexions de transport (du même ou de différents protocoles de transport). Une connexion peut être associée à un canal ou aux deux canaux d'une session ; le client et le serveur négocient si une connexion va porter le trafic pour un canal ou les deux via les opérations CREATE_SESSION (paragraphe 18.36) et BIND_CONN_TO_SESSION (paragraphe 18.34). Quand une session est créée via CREATE_SESSION, la connexion qui transportait la demande CREATE_SESSION est automatiquement associée au canal avant, et facultativement au canal de retour. Si le client spécifie la non protection d'état (paragraphe 18.35) quand la session est créée, alors quand SEQUENCE est transmis sur une connexion différente, la connexion est automatiquement associée au canal avant de la session spécifiée dans l'opération SEQUENCE.

L'association d'une connexion avec une session n'est pas exclusive. Une connexion associée au ou aux canaux d'une session peut être simultanément associée au ou aux canaux d'autres sessions incluant des sessions associées à d'autres identifiants de client.

Il est permis que des connexions de multiples types de transport soient associées au même canal. Par exemple, des connexions TCP et RDMA peuvent être associées au canal avant. Dans le cas où une connexion RDMA et une connexion non RDMA sont associées dans le même canal, le nombre maximum de créneaux DEVRAIT être d'au moins un de plus que le nombre total de crédits RDMA (paragraphe 2.10.6.1). De cette façon, si tous les crédits RDMA sont utilisés, la connexion non RDMA peut avoir au moins une demande en instance. Si un serveur prend en charge plusieurs types de transport, il DOIT permettre à un client d'associer des connexions provenant de chaque transport à un canal.

Il est permis à une connexion d'un type de transport d'être associée au canal avant, et à une connexion d'un type différent d'être associée au canal de retour.

2.10.4 Portée de serveur

Les serveurs spécifient chacun une valeur de portée de serveur sous la forme d'une chaîne opaque eir_server_scope retournée au titre du résultat d'une opération EXCHANGE_ID. L'objet de la portée de serveur est de permettre à un groupe de serveurs d'indiquer aux clients qu'un ensemble de serveurs partageant la même valeur de portée de serveur s'est arrangé pour utiliser des valeurs distinctes d'identifiants opaques afin que deux serveurs n'allouent jamais la même valeur à deux objets distincts. Donc, les identifiants générés par deux serveurs au sein de cet ensemble peuvent être supposés compatibles de sorte que, dans certains cas importants, les identifiants générés par un serveur de cet ensemble peuvent être présentés à un autre serveur de la même portée.

L'utilisation de telles valeurs compatibles n'implique pas qu'une valeur générée par un serveur va toujours être acceptée par un autre. Dans la plupart des cas, elle ne va pas l'être. Cependant, un serveur ne va pas par inadvertance accepter une valeur générée par un autre serveur. Quand il l'accepte, cela va être parce que elle est reconnue comme valide et portant la même signification que sur un autre serveur de la même portée.

Quand les serveurs sont de la même portée de serveur, cette compatibilité de valeurs s'applique aux identifiants suivants :

* Valeurs de bride de fichier. Une valeur de bride de fichier acceptée par deux serveurs de la même portée de serveur note le même objet. Une opération WRITE envoyée à un serveur est reflétée immédiatement dans un READ envoyé à l'autre.

* Valeurs de propriétaire de serveur. Quand les valeurs de portée de serveur sont les mêmes, la valeur de propriétaire de serveur peut être validement comparée. Dans les cas où les valeurs de portée de serveur sont différentes, les valeurs de propriétaire de serveur sont traitées comme différentes même si elles contiennent des chaînes d'octet identiques.

La coordination entre serveurs dont il est exigé qu'ils fournissent une telle compatibilité peut être assez minimale, et limitée à une simple partition de l'espace d'identifiants. La reconnaissance de valeurs communes exige de la mise en œuvre supplémentaire, mais cela peut être fait sur mesure dans les situations spécifiques où cette reconnaissance est désirée.

Les clients auront l'occasion de comparer les valeurs de portée de serveur de plusieurs serveurs dans un certain nombre de circonstances, dont chacune va être discutée dans le paragraphe fonctionnel approprié :

* Quand des valeurs de propriétaire de serveur reçues en réponse à es opérations EXCHANGE_ID envoyées à plusieurs adresses réseau sont comparées pour déterminer la validité de diverses formes de multiplexage, comme décrit au paragraphe 11.5.2.

* Quand la reconfiguration de réseau ou serveur cause la direction de la même adresse réseau sur des serveurs différents, avec la nécessité pour le client de déterminer quand la réclamation de verrou devrait être tentée, comme décrit au paragraphe 8.4.2.1.

Quand deux réponses à EXCHANGE_ID, chacune provenant de deux différentes adresses réseau de serveur, ont la même portée de serveur, un certain nombre de façons dont un client peut valider que la portée de serveur commune sont dues à ce que les deux serveurs coopèrent dans un groupe.

* Si les deux demandes EXCHANGE_ID ont été envoyées avec RPCSEC_GSS ([RFC2203], [RFC5403], [RFC7861]) l'authentification et le serveur principal sont les mêmes pour les deux cibles, l'égalité de portée de serveur est validée. Il est RECOMMANDÉ que deux serveurs qui ont l'intention de partager la même portée de serveur et le même identifiant majeur de propriétaire de serveur partagent aussi le même nom de principal. Dans certains cas, cela simplifie la tâche du client pour valider la portée de serveur.

* Le client peut accepter l'apparition du second serveur dans l'attribut fs_locations ou fs_locations_info pour un système de fichiers pertinent. Par exemple, si il y a un événement de migration pour un système de fichiers particulier ou si il y a des verrous à réclamer sur un système de fichiers particulier, les attributs pour ce système de fichiers particulier peuvent être utilisés. Le client envoie la demande GETATTR au premier serveur pour l'attribut fs_locations ou fs_locations_info avec l'authentification RPCSEC_GSS. Il peut devoir faire cela à l'avance du besoin de vérifier la portée de serveur commune. Si le client réussit à authentifier la réponse à GETATTR, et si la demande et la réponse GETATTR contenant l'attribut fs_locations ou fs_locations_info se rèfèrent au second serveur, alors l'égalité de portée de serveur est prise en charge. Un client peut choisir de limiter l'utilisation de cette forme de prise en charge aux informations pertinentes pour le système de fichiers spécifique impliqué (par exemple un système de fichiers qui migre).

2.10.5 Multiplexage

Le multiplexage est l'utilisation de plusieurs connexions entre un client et un serveur afin d'augmenter la vitesse du transfert de données. NFSv4.1 prend en charge deux types de multiplexage : le multiplexage de session et le multiplexage d'identifiant de client.

Dans le contexte d'une seule adresse réseau de serveur, on peut supposer que toutes les connexions accèdent au même serveur, et les serveurs NFSv4.1 DOIVENT prendre en charge les deux formes de multiplexage. Quand plusieurs connexions utilisent un ensemble d'adresses réseau pour accéder au même serveur, le serveur DOIT prendre en charge les deux formes de multiplexage. Les serveurs NFSv4.1 dans une configuration en grappe PEUVENT permettre que des adresses réseau pour différents serveurs utilisent un multiplexage d'identifiant de client.

Les clients peuvent utiliser l'une et l'autre forme de multiplexage tant qu'ils ne violent pas, quand il y a multiplexage entre différentes adresses réseau de serveur, le mandat des serveurs quant à la sorte de multiplexage permis (voir ci-dessous). À l'égard des canaux de rappel, le client DOIT permettre au serveur de choisir parmi tous les canaux de rappel valides pour un identifiant de client donné et DOIT prendre en charge le multiplexage quand les connexions qui prennent en charge le canal de retour permettent d'utiliser le multiplexage de session ou d'identifiant de client pour les rappels.

Le multiplexage de session est essentiellement l'association de plusieurs connexions, chacune avec une cible et/ou adresse réseau de source potentiellement différentes, pour la même session. Quand les adresses réseau de la cible (adresses de serveur) des deux connexions sont les mêmes, le serveur DOIT prendre en charge un tel multiplexage de session. Quand les adresses réseau de la cible sont différentes, le serveur PEUT indiquer une telle prise en charge en utilisant les données retournées par l'opération EXCHANGE_ID (voir ci-dessous).

Le multiplexage d'identifiant de client est l'association de plusieurs sessions au même identifiant de client. Les serveurs DOIVENT prendre en charge le multiplexage d'identifiant de client pour deux adresses réseau cible chaque fois qu'ils permettent le multiplexage de session pour ces deux mêmes adresses réseau. De plus, un serveur PEUT, en présentant le même identifiant majeur de propriétaire de serveur (paragraphe 2.5) et la même portée de serveur (paragraphe 2.10.4) permettre un cas supplémentaire de multiplexage d'identifiant de client. Quand deux serveurs retournent le même propriétaire de serveur majeur et portée de serveur, cela signifie que les deux serveurs coopèrent sur la gestion d'état de verrou, ce qui est un prérequis pour le multiplexage d'identifiant de client.

Distinguer quand il est permis au client d'utiliser le multiplexage de session et d'identifiant de client exige la compréhension de comment les résultats de l'opération EXCHANGE_ID (paragraphe 18.35) identifient un serveur. Supposons qu'un client envoie des EXCHANGE_ID sur deux connexions différentes, chacune avec une adresse réseau cible éventuellement différente, mais que chaque opération EXCHANGE_ID ait la même valeur dans le champ eia_clientowner. Si le même serveur NFSv4.1 écoute sur chaque connexion, alors chaque résultat de EXCHANGE_ID DOIT retourner les mêmes valeurs de eir_clientid, eir_server_owner.so_major_id, et eir_server_scope. Le client peut alors traiter chaque connexion comme se référant au même serveur (sous réserve de vérification; voir le paragraphe 2.10.5.1) et il peut utiliser chaque connexion pour multiplexer les demandes et réponses. Le choix du client est si le multiplexage de session ou le multiplexage d'identifiant de client s'applique.

Multiplexage de session. Si l'argument eia_clientowner est le même dans deux demandes EXCHANGE_ID différentes, et si les résultats de eir_clientid, eir_server_owner.so_major_id, eir_server_owner.so_minor_id, et eir_server_scope correspondent dans les deux résultats d'EXCHANGE_ID, alors il est permis au client d'effectuer le multiplexage de session. Si le client n'a pas une transposition de session au couple de eir_clientid, eir_server_owner.so_major_id, eir_server_scope, et eir_server_owner.so_minor_id, alors il crée la session via une opération CREATE_SESSION sur une des connexions, qui associe la connexion à la session. Si il y a une session pour le couple, le client peut envoyer BIND_CONN_TO_SESSION pour associer la connexion à la session.

Bien sûr, si le client ne désire pas utiliser le multiplexage de session, il n'est pas obligé de le faire. Il peut invoquer CREATE_SESSION sur la connexion. Il va en résulter un multiplexage d'identifiant de client comme décrit ci-dessous. Il peut aussi décider d'abandonner la connexion si il choisit de ne pas utiliser le multiplexage.

Multiplexage d'identifiant de client. Si l'argument eia_clientowner est le même dans deux demandes EXCHANGE_ID différentes, et si les résultats de eir_clientid, eir_server_owner.so_major_id, et eir_server_scope correspondent dans les deux résultats d'EXCHANGE_ID, il est alors permis au client d'effectuer le multiplexage d'identifiant de client (sans considération de si les résultats de eir_server_owner.so_minor_id correspondent). Le client peut associer chaque connexion à des sessions différentes, où chaque session est associée au même serveur.

Le client réalise l'acte de multiplexage d'identifiant de client en invoquant CREATE_SESSION sur chaque connexion, en

utilisant le même identifiant de client que retourné dans eir_clientid. Ces invocations créent deux sessions et aussi associent chaque connexion à sa session respective. Le client est libre de refuser d'utiliser le multiplexage d'identifiant de client en éliminant simplement la connexion à ce moment.

Quand on fait le multiplexage d'identifiant de client, l'état de verrouillage est partagé à travers les sessions associées au même identifiant de client. Cela exige que le serveur coordonne l'état à travers les sessions et que le client soit capable d'associer le même état de verrouillage à plusieurs sessions.

Il est toujours possible que, par suite de diverses sortes d'événement de reconfiguration, les valeurs de eir_server_scope et eir_server_owner puissent être différentes sur les demande EXCHANGE_ID suivantes faites à la même adresse réseau.

Dans la plupart des cas, de tels événements de reconfiguration vont être perturbants et indiquent qu'une adresse IP précédemment connectée à un serveur est maintenant connectée à un serveur entièrement différent.

On donne des lignes directrices sur le traitement de telles situations par le client :

* Quand eir_server_scope change, le client n'a pas d'assurance que les identifiants qu'il a obtenus précédemment (par exemple, des brides de fichier) peuvent être valablement utilisés sur le nouveau serveur, et, même si le nouveau serveur les accepte, il n'y a pas d'assurance que ce ne soit pas par accident. Donc, il vaut mieux traiter tous ces états comme perdus ou périmés, bien qu'un client puisse supposer que la probabilité d'une acceptation par inadvertance est faible et traite cette situation comme dans le cas suivant.

* Quand eir_server_scope reste le même et eir_server_owner.so_major_id change, le client peut utiliser les brides de fichier qu'il a, considérer son état de verrouillage comme perdu, et tenter de réclamer ou autrement obtenir à nouveau ses verrous. Il pourrait trouver que sa bride de fichier est maintenant périmée. Cependant, si NFS4ERR_STALE n'est pas retourné, il peut procéder à réclamer ou autrement ré-obtenir son état d'ouverture de verrouillage.

* Quand eir_server_scope et eir_server_owner.so_major_id restent les mêmes, le client doit alors utiliser les valeurs courantes de eir_server_owner.so_minor_id pour décider des formes appropriées de multiplexage. Il peut en résulter l'élimination de connexions ou la création de nouvelles sessions.

2.10.5.1 Vérification des revendications de correspondance d'identité de serveur

Quand le serveur répond en utilisant deux connexions différentes qui prétendent à des valeurs correspondantes ou ou partiellement correspondantes de eir_server_owner, eir_server_scope, et eir_clientid, le client n'a pas à faire confiance aux revendications des serveurs. Le client peut vérifier de la façon suivante ces revendications avant de multiplexer le trafic :

* Pour le multiplexage de session, les clients DEVRAIENT vérifier de façon fiable si les connexions entre différents chemins de réseau sont en fait associées au même serveur NFSv4.1 et utilisables sur la même session, et les serveurs DOIVENT permettre aux clients d'effectuer une vérification fiable. Quand un identifiant de client est créé, le client DEVRAIT spécifier que BIND_CONN_TO_SESSION est à vérifier en accord avec les options de protection d'état SP4_SSV ou SP4_MACH_CRED (paragraphe 18.35). Pour SP4_SSV, la vérification fiable dépend d'un secret partagé (le SSV) qui est établi via l'opération SET_SSV (voir le paragraphe 18.47).

  Quand une nouvelle connexion est associée à la session (via l'opération BIND_CONN_TO_SESSION, voir le paragraphe 18.34) si le client a spécifié la protection d'état SP4_SSV pour l'opération BIND_CONN_TO_SESSION, le client DOIT envoyer le BIND_CONN_TO_SESSION avec la protection RPCSEC_GSS, en utilisant l'intégrité ou la confidentialité, et une bride RPCSEC_GSS créée avec le mécanisme GSS SSV (voir le paragraphe 2.10.9).

  Si le client essaye par erreur d'associer une connexion à une session d'un mauvais serveur, le serveur va soit rejeter la tentative parce que il ne connaît pas l'identifiant de session des arguments BIND_CONN_TO_SESSION, soit rejeter la tentative parce que l'authentification RPCSEC_GSS échoue. Même si le serveur accepte par erreur ou malveillance la tentative d'association de la connexion, le vérificateur RPCSEC_GSS qu'il calcule dans la réponse ne va pas être vérifié par le client, donc le client va savoir qu'il ne peut pas utiliser la connexion pour multiplexer la session spécifiée.

  Si le client a spécifié la protection d'état SP4_MACH_CRED, l'opération BIND_CONN_TO_SESSION va utiliser l'intégrité ou la confidentialité RPCSEC_GSS, en utilisant les mêmes accréditifs qu'utilisés quand l'identifiant de client a été créé. L'authentification mutuelle via RPCSEC_GSS assure au client que la connexion est associée à la session correcte du serveur correct.

* Pour le multiplexage d'identifiant de client, le client a au moins deux options pour vérifier que le même identifiant de

client obtenu de deux opérations EXCHANGE_ID différentes vient du même serveur. La première option est d'utiliser l'authentification RPCSEC_GSS lors de l'envoi de chaque opération EXCHANGE_ID. Chaque fois qu'un EXCHANGE_ID est envoyé avec l'authentification RPCSEC_GSS, le client note le nom principal de la cible GSS. Si le résultat du EXCHANGE_ID indique que le multiplexage d'identifiant de client est possible, et si les noms principaux de la cible GSS sont les mêmes, les serveurs sont le même et le multiplexage d'identifiant de client est permis.

La seconde option de vérification est d'utiliser la protection SP4_SSV. Quand le client envoie l'EXCHANGE_ID, il spécifie la protection SP4_SSV. Le premier EXCHANGE_ID qu'envoie le client doit toujours être confirmé par une invocation de CREATE_SESSION. Le client envoie alors SET_SSV. Plus tard, le client envoie un EXCHANGE_ID à une seconde adresse réseau de destination différente de celle où le premier EXCHANGE_ID a été envoyé. Le client vérifie que chaque réponse d'EXCHANGE_ID a le même eir_clientid, eir_server_owner.so_major_id, et eir_server_scope. Si il en est ainsi, le client vérifie la revendication en envoyant une opération CREATE_SESSION à la seconde adresse de destination, protégée avec l'intégrité RPCSEC_GSS en utilisant une bride RPCSEC_GSS retournée par le second EXCHANGE_ID. Si le serveur accepte la demande CREATE_SESSION, et si le client vérifie le vérificateur RPCSEC_GSS et les codes d'intégrité, alors le client a la preuve que le second serveur connaît le SSV, et donc que les deux serveurs coopèrent pour les besoins de la spécification de la portée de serveur et le multiplexage d'identifiant de client.

2.10.6 Sémantique de "exactement une fois"

Via la session, NFSv4.1 offre la sémantique de exactement une fois (EOS, *exactly once semantics*) pour les demandes envoyées sur un canal. EOS est pris en charge sur les deux canaux, avant et de retour.

Chaque demande COMPOUND ou CB_COMPOUND envoyée avec une opération SEQUENCE ou CB_SEQUENCE en tête DOIT être exécutée exactement une fois par le receveur. Cette exigence tient sans considération de si la demande est envoyée en spécifiant la mise en mémoire tampon de la réponse (voir le paragraphe 2.10.6.1.3). L'exigence tient même si le demandeur envoie la demande sur une session créée entre un client de données NFS et un serveur de données pNFS. Pour comprendre la raison de cette exigence, on divise les demandes en trois classes :

* Demandes non idempotentes,
* Demandes modifiantes idempotentes.
* Demandes non modifiantes idempotentes.

Un exemple de demande non idempotente est RENAME. Évidemment, si un répondeur exécute deux fois la même demande RENAME, et si la première exécution réussit, la ré-exécution va échouer. Si le répondeur retourne le résultat de la ré-exécution, ce résultat est incorrect. Donc, l'EOS est exigé pour les demandes non idempotentes.

Un exemple de demande modifiante idempotente est une demande COMPOUND contenant une opération WRITE. L'exécution répétée du même WRITE a le même effet que l'exécution de ce WRITE une seule fois. Néanmoins, appliquer l'EOS pour les WRITE et autres demandes modifiantes idempotentes est nécessaire pour éviter la corruption des données.

Supposons qu'un client envoie un WRITE A à un serveur non conforme qui n'applique pas l'EOS, et ne reçoit pas de réponse, peut-être due à une partition du réseau. Le client se reconnecte au serveur et renvoie WRITE A. Maintenant, le serveur a deux instances de A en cours. Le serveur peut être dans une situation dans laquelle il exécute et répond au ré-essai de A, tandis que le premier A est encore en attente dans le système d'entrée/sortie interne du serveur pour des ressources. À réception de la réponse à la seconde tentative de WRITE A, le client croit que son WRITE est fait, de sorte qu'il est libre d'envoyer WRITE B, qui se chevauche avec la gamme d'octets de A. Quand le A original est sorti du système d'entrée/sortie du serveur et exécuté (donc la seconde fois que A va avoir été écrit) alors ce qui a été écrit par B peut être écrasé et donc corrompu.

Un exemple d'une demande idempotente non modifiante est un COMPOUND contenant SEQUENCE, PUTFH, READLINK, et rien d'autre. La ré-exécution d'une telle demande ne va pas causer de corruption des données ou produire un résultat incorrect. Néanmoins, pour que la mise en œuvre reste simple, le répondeur DOIT appliquer l'EOS pour toutes les demandes, qu'elles soient ou non idempotentes et non modifiantes.

Noter qu'une vraie et complète EOS n'est possible que si le serveur fait persister la mémoire tampon de réponse dans une mémorisation stable, et si le serveur est mis en œuvre de façon à ne jamais exiger un redémarrage (bien sûr, si un tel serveur existe, la distinction entre une mémoire tampon de réponse gardée dans une mémorisation stable et une qui ne l'est pas n'a pas de sens). Voir au paragraphe 2.10.6.5 pour une discussion de la persistance dans la mémoire tampon de réponse. Néanmoins, même si le serveur ne conserve pas la mémoire tampon de réponse, EOS améliore la robustesse et la correction par rapport aux précédentes versions de NFS parce que les mémoires tampon traditionnelles de demande/réponse

dupliquées étaient fondées sur l'identifiant de transaction (XID) ONC RPC. Le paragraphe 2.10.6.1 explique les insuffisances du XID comme base d'une mémoire tampon de réponse et décrit comment les sessions NFSv4.1 améliorent le XID.

2.10.6.1 Identifiants de créneaux et antémémoire de réponse

La couche RPC fournit un identifiant de transaction (XID) qui, bien qu'exigé qu'il soit unique, n'est pas pratique pour suivre les demandes, pour deux raisons. D'abord, le XID n'a de signification que pour le demandeur ; il ne peut pas être interprété par le répondeur sauf pour un essai d'égalité avec les demandes envoyées précédemment. Quand on consulte une mémoire tampon de demande dupliquée fondée sur RPC, l'opacité du XID requiert une recherche coûteuse en calcul (souvent via un hachage qui inclut le XID et l'adresse de source). Les demandes NFSv4.1 utilisent un identifiant de créneau non opaque, qui est un indice dans un tableau de créneaux, ce qui est bien plus efficace. Ensuite, parce que les demandes RPC peuvent être exécutées dans n'importe quel ordre par le répondeur, il n'y a pas de limite au nombre de demandes qui peuvent être en instance à tout moment. Pour réaliser une EOS parfaite, utiliser ONC RPC exigerait de mémoriser toutes les réponses dans la mémoire tampon de réponse. Les XID font 32 bits ; mémoriser plus de quatre milliards (2^(32)) de réponses dans la mémoire tampon de réponse n'est pas pratique. En pratique, les précédentes versions de NFS ont choisi de mémoriser un nombre fixe de réponses dans la mémoire tampon, et d'utiliser l'approche du moins récemment utilisé (LRU, *least recently used*) pour remplacer les entrées de mémoire tampon par de nouvelles entrées quand la mémoire tampon est pleine. Dans NFSv4.1, le nombre de demandes en instance est limité par la taille du tableau de créneaux, et un identifiant de séquence par créneau est utilisé pour dire au répondeur quand il est sûr de supprimer une réponse dans la mémoire tampon.

Dans la mémoire tampon de réponse NFSv4.1, quand le demandeur envoie une nouvelle demande, il choisit un identifiant de créneau dans la gamme 0..N, où N est l'identifiant de créneau maximum courant du répondeur accordé au demandeur sur la session sur laquelle la demande est à envoyer. La valeur de N commence comme égale à ca_maxrequests - 1 (paragraphe 18.36) mais peut être ajustée par la réponse à SEQUENCE ou CB_SEQUENCE comme décrit plus loin. L'identifiant de créneau doit être inutilisé par les demandes que le demandeur a déjà actives sur la session. "Inutilisé" signifie ici que le demandeur n'a pas de demande en instance pour cet identifiant de créneau.

Un créneau contient un identifiant de séquence et la réponse mise en mémoire tampon qui correspond à la demande envoyée avec cet identifiant de séquence. L'identifiant de séquence est une valeur de 32 bit non signée, et est donc dans la gamme 0..0xFFFFFFFF (2^(32) - 1). La première fois qu'un créneau est utilisé, le demandeur DOIT spécifier un identifiant de séquence de un (paragraphe 18.36). Chaque fois qu'un créneau est réutilisé, la demande DOIT spécifier un identifiant de séquence qui est supérieur de un à celui de la demande précédente sur le créneau. Si le précédent identifiant de séquence était 0xFFFFFFFF, alors la prochaine demande pour le créneau DOIT avoir l'identifiant de séquence réglé à zéro (c'est-à-dire, (2^(32) - 1) + 1 mod 2^(32)).

L'identifiant de séquence accompagne l'identifiant de créneau dans chaque demande. C'est pour la vérification critique chez le répondeur : il est utilisé pour déterminer efficacement si une demande utilisant un certain identifiant de créneau est une retransmission ou une nouvelle demande, jamais vue auparavant. Il n'est pas faisable que le demandeur certifie qu'il retransmet pour mettre en œuvre cela, parce que pour toute demande, le demandeur ne peut pas savoir si le répondeur l'a vue sauf si le répondeur répond réellement. Bien sûr, si le demandeur a vu la réponse, il ne va pas retransmettre.

Le répondeur compare l'identifiant de séquence de chaque demande reçue avec le dernier reçu précédemment pour cet identifiant de créneau, pour voir si la nouvelle demande est :

* Une nouvelle demande, dans laquelle l'identifiant de séquence est supérieur de un à celui vu précédemment dans le créneau (en tenant compte du retour à zéro de la séquence). Le répondeur procède à l'exécution de la nouvelle demande, et le répondeur DOIT augmenter de un l'identifiant de séquence du créneau.

* Une demande retransmise, dans laquelle l'identifiant de séquence est égal à celui actuellement enregistré dans le créneau. Si la demande d'origine est entièrement exécutée, le répondeur retourne la réponse en mémoire tampon. Voir au paragraphe 2.10.6.2 les directives sur comment le répondeur traite les réessais de demandes qui sont encore en cours.

* Un ré-essai décalé, dans lequel l'identifiant de séquence est moins que (en tenant compte du retour à zéro de la séquence) celui précédemment vu dans le créneau. Le répondeur DOIT retourner NFS4ERR_SEQ_MISORDERED (comme résultat de SEQUENCE ou CB_SEQUENCE).

* Une nouvelle demande mal ordonnée, dans laquelle l'identifiant de séquence est supérieur de deux ou plus que (en tenant compte du retour à zéro de la séquence) celui précédemment vu dans le créneau. Noter que parce que l'identifiant de séquence DOIT revenir à zéro quand il atteint 0xFFFFFFFF, une nouvelle demande mal ordonnée et un ré-essai mal ordonné ne peuvent pas être distingués. Donc, le répondeur DOIT retourner NFS4ERR_SEQ_MISORDERED (comme

résultat de SEQUENCE ou CB_SEQUENCE).

À la différence de XID, l'identifiant de créneau est toujours dans une gamme spécifique ; cela a deux implications. La première implication est que pour une session donnée, le répondeur a seulement besoin de mettre en mémoire tampon le résultat d'un nombre limité de demandes COMPOUND. La seconde implication découle de la première, qui est que à la différence des mémoires tampons de réponse indexées par XID (aussi appelées des mémoires tampons de demandes dupliquées - DRC, *duplicate request cache*) la mémoire tampon de réponse fondée sur l'identifiant de créneau ne peut pas déborder. Grâce à l'utilisation de l'identifiant de séquence pour identifier les demandes retransmises, le répondeur n'a pas besoin en fait de mettre la demande elle-même en mémoire tampon, réduisant les exigences de mémorisation de la mémoire tampon de réponse. Ces facilités rendent pratique le maintien de toutes les entrées requises pour une mémoire tampon de réponse efficace.

L'identifiant de créneau, l'identifiant de séquence, et l'identifiant de session prennent donc le rôle traditionnel du XID et de l'adresse réseau de source dans la mise en œuvre de mémoire tampon de réponse du répondeur. Cette approche est considérablement plus portable et complètement robuste -- elle n'est pas sujette à la réallocation d'accès lorsque les clients se reconnectent sur les réseaux IP. De plus, le XID RPC n'est pas utilisé dans la mémoire tampon de réponse, améliorant la robustesse de la mémoire tampon en face de toute réutilisation rapide des XID par le demandeur. Alors que le répondeur ne se soucie pas du XID pour les besoins de la gestion de mémoire tampon de réponse (mais le répondeur DOIT retourner le même XID que dans la demande) il y a néanmoins des considérations sur le XID dans NFSv4.1 qui sont les mêmes que dans toutes les autres versions précédentes de NFS. Le XID RPC reste dans chaque message et doit être formulé dans les demandes NFSv4.1 comme dans toute autre demande RPC ONC. Les raisons incluent :

* La couche RPC conserve sa sémantique et mise en œuvre existantes.

* Le demandeur et le répondeur doivent être capables d'inter opérer à la couche RPC, avant le décodage NFSv4.1 de l'opération SEQUENCE ou CB_SEQUENCE.

* Si une opération utilisée ne commence pas par SEQUENCE ou CB_SEQUENCE (par exemple, BIND_CONN_TO_SESSION) alors le XID RPC est nécessaire pour un fonctionnement correct pour rapprocher la réponse de la demande.

* L'opération SEQUENCE ou CB_SEQUENCE peut générer une erreur. Si c'est le cas, l'identifiant de créneau, l'identifiant de séquence, et l'identifiant de session (si présent) incorporés dans la demande ne vont pas être dans la réponse, et le demandeur aura seulement le XID pour confronter la réponse à la demande.

Étant donné que des XID bien formulés continuent d'être exigés, cela soulève la question de pourquoi les réponses à SEQUENCE et CB_SEQUENCE ont un identifiant de session, un identifiant de créneau, et un identifiant de séquence ? Avoir l'identifiant de session dans la réponse signifie que le demandeur n'a pas à utiliser le XID pour chercher l'identifiant de session, qui serait nécessaire si la connexion était associée à plusieurs sessions. Avoir l'identifiant de créneau et l'identifiant de séquence dans la réponse signifie que le demandeur n'a pas à utiliser le XID pour chercher l'identifiant de créneau et l'identifiant de séquence. De plus, comme le XID fait seulement 32 bits, il est trop petit pour garantir la ré-association d'une réponse avec sa demande [RPC-XID] ; avoir l'identifiant de session, l'identifiant de créneau, et l'identifiant de séquence dans la réponse permet au client de valider que la réponse appartient en fait à la demande correspondante.

L'opération SEQUENCE (et CB_SEQUENCE) porte aussi une valeur "highest_slotid" *(plus haut identifiant de créneau)* qui porte des informations suppélmentaires sur l'usage du créneau de demandeur. Le demandeur DOIT toujours indiquer l'identifiant de créneau représentant la demande en cours avec la plus haute valeur de créneau. Le demandeur devrait dans tous les cas fournir la valeur la plus faible possible, bien qu'elle puisse être un peu augmentée au dessus de l'usage instantané actuel pour maintenir un niveau minimum ou optimal. Cela fournit au demandeur un moyen de redonner des créneaux inutilisés au répondeur, qui à son tour peut utiliser les informations pour réallouer les ressources.

Le répondeur répond avec les deux nouveaux highest_slotid cibles et un highest_slotid appliqué, décrits comme suit :

* Le highest_slotid cible est une indication au demandeur du highest_slotid que le répondeur souhaite qu'utilise le demandeur. Cela permet au répondeur de supprimer (ou ajouter) des ressources provenant d'un demandeur qui s'est trouvé ne pas les utiliser, afin de partager plus équitablement les ressources parmi un niveau variable de demandes d'autres demandeurs. Le demandeur doit toujours respecter les mises à jour de valeur du répondeur, car elles indiquent les nouvelles limites établies sur l'accès aux ressources de session du demandeur. Cependant, à cause de l'exécution de demandes en paralllèle, le demandeur peut avoir des demandes actives en cours reflétant des valeurs antérieures ; donc, le répondeur ne doit pas exiger que le demandeur s'y conforme immédiatement.

* Le highest_slotid appliqué indique le plus haut identifiant de créneau qu'il est permis au demandeur d'utiliser sur une opération SEQUENCE ou CB_SEQUENCE suivante. Le highest_slotid appliqué du répondeur DEVRAIT être non inférieur au highest_slotid que le demandeur a indiqué dans les arguments SEQUENCE ou CB_SEQUENCE.

Un demandeur peut être intransigeant à l'égard de la diminution de son argument highest_slotid d'une opération SEQUENCE, c'est-à-dire que le demandeur continue d'ignorer le highest_slotid cible dans la réponse à l'opération SEQUENCE, et continue de régler son argument highest_slotid à être supérieur au highest_slotid cible. Ce comportement peut être considéré comme particulièrement maladroit quand le répondeur sait qu'il n'y a pas de demande en instance avec un identifiant de créneau supérieur à son highest_slotid cible. En face d'une telle intransigeance, le répondeur est libre d'effectuer une action plus énergique, et PEUT répondre avec un nouveau highest_slotid appliqué inférieur à son highest_slotid appliqué précédent. Ensuite, si le demandeur continue d'envoyer des demandes avec un highest_slotid supérieur au nouveau highest_slotid appliqué du répondeur, le serveur PEUT retourner NFS4ERR_BAD_HIGH_SLOT, sauf si l'identifiant de créneau dans la demande est supérieur au nouveau highest_slotid appliqué et si la demande est un ré-essai.

Le répondeur DEVRAIT conserver les créneaux qu'il veut retirer jusqu'à ce que le demandeur envoie une demande avec un highest_slotid inférieur ou égal au nouveau highest_slotid appliqué du répondeur.

Le demandeur peut aussi être intransigeant à l'égard de l'envoi de demandes non de ré-essai avec un identifiant de créneau qui excède le highest_slotid du répondeur. Une fois que le répondeur a été forcé de diminuer le highest_slotid appliqué, il est seulement permis au demandeur d'envoyer des réessais sur les créneaux qui excédent le highest_slotid du répondeur. Si une demande est reçue avec un identifiant de créneau supérieur au nouveau highest_slotid appliqué, et si l'identifiant de séquence est supérieur de un à ce qui est dans la mémoire tampon de réponse du créneau, alors le serveur peut à la fois retirer le créneau et retourner NFS4ERR_BADSLOT (cependant, le serveur NE DOIT PAS faire l'un et pas l'autre). La raison pour laquelle il est sûr de retirer le créneau est parce que en utilisant le prochain identifiant de séquence, le demandeur indique qu'il a reçu la réponse précédente pour le créneau.

* Le demandeur DEVRAIT utiliser le plus bas créneau disponible quand il envoie une nouvelle demande. De cette façon, le répondeur peut être capable de retirer plus vite les entrées de créneau. Cependant, lorsque le répondeur ajuste activement son highest_slotid accordé, il ne va pas être capable d'utiliser seulement la réception de l'identifiant de créneau et le highest_slotid dans la demande. Ni l'identifiant de créneau ni le highest_slotid utilisé dans une demande ne peuvent refléter l'idée actuelle du répondeur de la limite de session du demandeur, parce que la demande peut avoir été envoyée du demandeur avant que la mise à jour ait été reçue. Donc, dans le cas de l'ajustement vers le bas, le répondeur peut devoir conserver un nombre d'entrées de mémoire tampon de réponse au moins aussi grand que l'ancienne valeur de maximum de demandes en instances, jusqu'à ce qu'il puisse déduire que le demandeur a vu une réponse contenant le nouveau highest_slotid accordé. Le répondeur peut déduire que le demandeur a vu une telle réponse quand il reçoit une nouvelle demande avec le même identifiant de créneau que celui auquel la demande répond et le prochain plus haut identifiant de séquence.

2.10.6.1.1 Mise en antémémoire des réponses SEQUENCE et CB_SEQUENCE

Quand une opération SEQUENCE ou CB_SEQUENCE est exécutée avec succès, sa réponse DOIT toujours être mise en antémémoire. Précisément, l'identifiant de session, l'identifiant de séquence, et l'identifiant de créneau DOIVENT être mis en antémémoire dans la mémoire tampon de réponse. La réponse provenant de SEQUENCE inclut aussi le plus haut identifiant de créneau, le plus haut identifiant de créneau cible, et les fanions d'état. Au lieu de mettre en antémémoire ces valeurs, le serveur PEUT recalculer les valeurs à partir de l'état actuel du canal avant, de la session, et/ou de l'identifiant de client comme approprié. De même, la réponse provenant de CB_SEQUENCE inclut un plus haut identifiant de créneau et un plus haut identifiant de créneau cible. Le client PEUT recalculer les valeurs à partir de l'état actuel de la session comme approprié.

Sans considération de si un répondeur recalcule ou non le plus haut identifiant de créneau, l'identifiant de créneau cible, et l'état des réponses aux ré-essais, le demandeur NE DOIT PAS supposer que les valeurs sont recalculées chaque fois qu'il reçoit une réponse après l'envoi d'un ré-essai, car il n'a pas de moyen de savoir si la réponse qu'il a reçu a été envoyée par le répondeur en réponse au ré-essai ou est une réponse en retard à la demande d'origine. Donc, il se peut que le plus haut identifiant de créneau, l'identifiant de créneau cible, ou les bits d'état puissent refléter l'état des affaires quand la demande a été exécutée pour la première fois. Bien qu'il soit valide d'agir sur la base de telles informations retardées, cela peut causer un travail inutile au receveur de la réponse. Les demandeurs PEUVENT choisir d'envoyer des demandes supplémentaires pour obtenir l'état actuel des affaires ou utiliser l'état des affaires rapporté par les demandes suivantes, de préférence à agir immédiatement sur des données qui pourraient être périmées.

2.10.6.1.2 Erreurs de SEQUENCE et CB_SEQUENCE

Chaque fois que SEQUENCE ou CB_SEQUENCE retourne une erreur, l'identifiant de séquence du créneau NE DOIT PAS changer. Le répondeur NE DOIT PAS modifier l'entrée de mémoire tampon de réponse pour le créneau chaque fois qu'une erreur est retournée de SEQUENCE ou CB_SEQUENCE.

2.10.6.1.3 Mise en antémémoire facultative de réponse

Demande par demande, le demandeur peut choisir de diriger le répondeur sur la mise en antémémoire de la réponse pour toutes les opérations après la première opération (SEQUENCE ou CB_SEQUENCE) via les champs sa_cachethis ou csa_cachethis des arguments à SEQUENCE ou CB_SEQUENCE. La raison pour laquelle il n'inviterait pas le répondeur à mettre en antémémoire la réponse entière est que la demande est composée de toutes les opérations idempotentes [NFS-Serv]. Mettre en antémémoire la réponse peut offrir peu d'avantages. Si la réponse est trop grande (voir le paragraphe 2.10.6.4) elle peut ne pas être mettable en antémémoire de toutes façons. Même si la réponse à une demande idempotente est assez petite pour être mise en antémémoire, une mise en antémémoire non nécessaire de la réponse ralentit le serveur et augmente la latence de RPC.

Que le demandeur demande ou non que la réponse soit mise en antémémoire n'a pas d'effet sur le traitement du créneau. Si le résultat de SEQUENCE ou CB_SEQUENCE est NFS4_OK, alors l'identifiant de séquence du créneau DOIT être incrémenté de un. Si un demandeur n'invite pas le répondeur à mettre la réponse en antémémoire, le répondeur DOIT faire une des choses suivantes :

* Le répondeur peut mettre en antémémoire la réponse originale entière. Même si sa_cachethis ou csa_cachethis est FAUX, le répondeur est toujours libre de mettre en antémémoire. Il peut choisir cette approche afin de simplifier la mise en œuvre.

* Le répondeur entre dans sa mémoire tampon de réponse une réponse consistant en le résultat original de l'opération SEQUENCE ou CB_SEQUENCE, et avec la prochaine opération dans COMPOUND ou CB_COMPOUND qui a l'erreur NFS4ERR_RETRY_UNCACHED_REP. Donc, si le demandeur réessaye plus tard la demande, il va obtenir NFS4ERR_RETRY_UNCACHED_REP. Si un répondeur reçoit une opération SEQUENCE réessayée où la réponse au COMPOUND ou CB_COMPOUND n'a pas été mise en antémémoire, alors le répondeur,
  - PEUT retourner NFS4ERR_RETRY_UNCACHED_REP en réponse à une opération SEQUENCE si l'opération SEQUENCE n'est pas la première opération (étant donné qu'un demandeur qui fait cela est en violation du protocole NFSv4.1) ;
  - NE DOIT PAS retourner NFS4ERR_RETRY_UNCACHED_REP dans la réponse à une opération SEQUENCE si l'opération SEQUENCE est la première opération.

* Si la seconde opération est illégale, ou une opération qui était légale dans une version mineure précédente de NFSv4 et NE DOIT PAS être supportée dans la version mineure actuelle (par exemple, SETCLIENTID) le répondeur NE DOIT PAS retourner NFS4ERR_RETRY_UNCACHED_REP. À la place, le répondeur DOIT retourner NFS4ERR_OP_ILLEGAL ou NFS4ERR_BADXDR ou NFS4ERR_NOTSUPP comme approprié.

* Si la seconde opération peut résulter en un autre état d'erreur, le répondeur PEUT retourner un état autre que NFS4ERR_RETRY_UNCACHED_REP, pourvu que l'opération ne soit pas exécutée d'une façon telle que l'état du répondeur change. Des exemples d'un tel état d'erreur incluent : NFS4ERR_NOTSUPP retourné pour une opération légale mais non EXIGÉE dans les versions mineures courantes, et donc non supportée par le répondeur, NFS4ERR_SEQUENCE_POS, et NFS4ERR_REQ_TOO_BIG.

La discussion ci-dessus suppose que la demande réessayée corresponde à l'originale. Le paragraphe 2.10.6.1.3.1 discute ce que le répondeur pourrait faire, et DOIT faire quand les demandes originale et réessayée ne correspondent pas. Comme le répondeur peut seulement mettre en antémémoire une petite quantité des informations qui seraient nécessaires pour déterminer si ceci est un cas de faux ré-essai, le répondeur peut envoyer au client une des réponses suivantes :

* La réponse en antémémoire à la demande d'origine (si le répondeur l'a entièrement mise en antémémoire et si les utilisateurs de la demande d'origine et de ré-essai correspondent).
* Une réponse qui consiste seulement en l'opération SEQUENCE avec l'erreur NFS4ERR_SEQ_FAUX_RETRY.
* Une réponse consistant en la réponse à SEQUENCE avec l'état NFS4_OK, avec la seconde opération comme elle apparaissait dans la demande répétée avec une erreur de NFS4ERR_RETRY_UNCACHED_REP ou une autre erreur comme décrit ci-dessus.
* Une réponse qui consiste en la réponse à SEQUENCE avec l'état NFS4_OK, avec la seconde opération telle qu'elle apparaissait dans la demande d'origine avec une erreur de NFS4ERR_RETRY_UNCACHED_REP ou une autre erreur comme décrit ci-dessus.

2.10.6.1.3.1 Ré-essai falsifié

Si un demandeur a envoyé une opération SEQUENCE avec un identifiant de créneau et identifiant de séquence qui sont dans la mémoire tampon de réponse mais si le répondeur a détecté que la demande réessayée n'est pas la même que la demande d'origine, incluant un ré-essai qui a des opérations différentes ou des arguments différents dans les opérations de l'originale et un ré-essai qui utilise un principal différent dans le champ Accréditifs de la demande RPC qui se traduit en un utilisateur différent, alors c'est un faux ré-essai. Quand le répondeur détecte un faux ré-essai, il lui est permis (mais pas toujours obligatoire) de retourner NFS4ERR_SEQ_FALSE_RETRY en réponse à l'opération SEQUENCE quand il détecte un faux ré-essai.

Les traductions de valeurs d'utilisateur particulièrement privilégiés en autres utilisateurs à cause de l'absence d'accréditifs d'une sûreté appropriée, comme configurée chez le répondeur, devraient être appliquées avant de déterminer si les utilisateurs sont les mêmes ou différents. Si le répondeur détermine que les utilisateurs sont différents entre la demande d'origine et un ré-essai, alors le répondeur DOIT retourner NFS4ERR_SEQ_FALSE_RETRY.

Si une opération de ré-essai est illégale, ou une opération qui était légale dans une précédente version mineure de NFSv4 et NE DOIT PAS être supportée dans la version mineure actuelle (par exemple, SETCLIENTID) le répondeur PEUT retourner NFS4ERR_SEQ_FALSE_RETRY (et DOIT le faire si les utilisateurs de la demande d'origine et du ré-essai diffèrent). Autrement, le répondeur PEUT retourner NFS4ERR_OP_ILLEGAL ou NFS4ERR_BADXDR ou NFS4ERR_NOTSUPP comme approprié. Noter que le traitement est différent de celui où le répondeur traite des réessais de demandes avec une réponse non mise en antémémoire. La différence est due à ce que NFS4ERR_SEQ_FALSE_RETRY est une erreur valide seulement pour les opérations SEQUENCE, tandis que NFS4ERR_RETRY_UNCACHED_REP est une erreur valide pour toutes les opérations sauf les opérations illégales et les opérations qui NE DOIVENT PAS être prises en charge dans la version mineure courante de NFSv4.

2.10.6.2 Réessai et répétition de réponse

Un demandeur NE DOIT PAS réessayer une demande, sauf si la connexion qu'il a utilisée pour envoyer la demande se déconnecte. Le demandeur peut alors reconnecter et envoyer la demande à nouveau, ou il peut renvoyer la demande sur une connexion différente qui est associée à la même session.

Si le demandeur est un serveur qui veut renvoyer une opération de rappel sur le canal de retour d'une session, le demandeur ne peut pas se reconnecter, bien sûr, parce que seul le client peut associer les connexions au canal de retour. Le serveur peut renvoyer la demande sur une autre connexion qui est liée au canal de retour de la même session. Si une telle connexion n'existe pas, le serveur DOIT indiquer que la session n'a pas de canal de retour en établissant le fanion SEQ4_STATUS_CB_PATH_DOWN_SESSION dans la réponse à la prochaine opération SEQUENCE provenant du client. Le client DOIT alors associer une connexion à la session (ou détruire la session).

Noter qu'il n'est pas fatal pour un demandeur de réessayer sans une déconnexion entre la demande et le ré-essai. Cependant, le ré-essai consomme des ressources, en particulier avec RDMA, où chaque demande, ré-essai ou non, consomme un crédit. Les réessais sans raison, en particulier les réessais envoyés peu après la tentative précédente, sont un mauvais usage de la bande passante du réseau et contreviennent à l'objet du système de contrôle d'encombrement inhérent à un transport.

Un demandeur DOIT attendre une réponse à une demande avant d'utiliser le créneau pour une autre demande. Si il n'attend pas la réponse, alors le demandeur ne sait pas quel identifiant de séquence utiliser pour le créneau sur sa prochaine demande. Par exemple, supposons qu'un demandeur envoie une demande avec l'identifiant de séquence 1, et n'attende pas la réponse. La prochaine fois qu'il utilise le créneau, il envoie la nouvelle demande avec l'identifiant de séquence 2. Si le répondeur n'a pas vu la demande avec l'identifiant de séquence 1, alors le répondeur n'attend pas l'identifiant de séquence 2, et rejette la nouvelle demande du demandeur avec NFS4ERR_SEQ_MISORDERED (par suite de SEQUENCE ou CB_SEQUENCE).

Le tissu RDMA ne garantit pas que les brides de mémoire (étiquettes "Steering") au sein de chaque "tronçon" RPC/RDMA [RFC8166] sont valides sur une portée hors d'une seule connexion. Donc, les brides utilisées par les opérations directes deviennent invalides après la perte de la connexion. Le serveur doit s'assurer que toutes les opérations RDMA qui doivent être répétées à partir de la mémoire tampon de réponse utilisent la ou les nouvelles brides fournies à partir de la demande la plus récente.

Un ré-essai pourrait être envoyé alors que la demande d'origine est encore en cours chez le répondeur. Le répondeur DEVRAIT traiter la question en retournant NFS4ERR_DELAY comme réponse à une opération SEQUENCE ou CB_SEQUENCE, mais les mises en œuvre PEUVENT retourner NFS4ERR_MISORDERED. Comme les erreurs provenant de SEQUENCE et CB_SEQUENCE ne sont jamais enregistrées dans la mémoire tampon de réponse, cette approche permet que le résultat de l'exécution de la demande d'origine soit proprement enregistré dans la mémoire tampon

de réponse (en supposant que le demandeur a spécifié que la réponse doit être mise en antémémoire).

2.10.6.3 Résolution de compétition de rappel de serveur

Il est possible que des rappels de serveur arrivent chez le client avant la réponse provenant des opérations du canal avant qui s'y rapportent. Par exemple, une délégation peut avoir été accordée à un client pour un fichier qu'il a ouvert, mais la réponse au OPEN (informant le client de l'accord de la délégation) peut être retardée dans le réseau. Si une opération en conflit arrive au serveur, il va rappeler la délégation en utilisant le canal de retour, qui peut être sur une connexion de transport différente, peut-être même sur un réseau différent, ou même une session différente associée au même identifiant de client.

La présence d'une session entre le client et le serveur atténue ce problème. Quand une session est en place, chaque demande de client est identifiée de façon univoque par son triplet { identifiant de session, identifiant de créneau, identifiant de séquence }. Selon les règles par lesquelles les entrées de créneau (entrées de mémoire tampon de réponse) sont retirées, le serveur a connaissance de si le client a "vu" chaque réponse du serveur. Le serveur peut donc fournir des informations suffisantes au client pour lui permettre de distinguer entre une condition d'erreur et une compétition de rappels conflictuels.

Pour chaque opération de client qui pourrait résulter en une sorte de rappel du serveur, le serveur DEVRAIT "se souvenir" du triplet { identifiant de session, identifiant de créneau, identifiant de séquence } de la demande du client jusqu'à ce que les règles de retrait d'identifiant de créneau permettent au serveur de déterminer que le client a, en fait, vu la réponse du serveur. Jusqu'au moment ou le triplet { identifiant de session, identifiant de créneau, identifiant de séquence } de demande peut être retiré, tous les rappels de l'objet associé DOIVENT porter un dispositif de ces identifiants de référence (dans les arguments de l'opération CB_SEQUENCE) pour le bénéfice du client. Après ce moment, il n'est pas nécessaire que le serveur fournisse ces informations dans les rappels qui s'y rapportent, car il est certain qu'une condition de compétition ne peut plus se produire.

L'opération CB_SEQUENCE qui commence chaque rappel de serveur porte une liste de triplets "référents" { identifiant de session, identifiant de créneau, identifiant de séquence }. Si le client trouve que la demande correspondant aux identifiant de session, identifiant de créneau, et identifiant de séquence référents est actuellement en instance (c'est-à-dire, la réponse du serveur n'a pas encore été vue par le client) il peut déterminer que le rappel a pris de court la réponse, et agir en conséquence. Si le client ne trouve pas que la demande correspondant au triplet référent est en instance (incluant le cas d'un identifiant de session se référant à une session détruite) alors il n'y a pas de concurrence par rapport à ce triplet. Le serveur DEVRAIT limiter les triplets référents aux demandes qui se réfèrent juste aux objets visés dans la procédure CB_COMPOUND.

Le client ne doit pas simplement attendre indéfiniment que la réponse de serveur attendue arrive avant de répondre au CB_COMPOUND qui a gagné la course, parce que il est possible qu'il soit retardé indéfiniment. Le client devrait supposer le cas probable que la réponse va arriver dans le délai moyen d'aller-retour pour les demandes COMPOUND au serveur, et attendre pendant cette période. Si cette période arrive à expiration, il peut répondre au CB_COMPOUND avec NFS4ERR_DELAY. Il y a d'autres scénarios dans lesquels les rappels peuvent concurrencer les réponses. Parmi eux sont les rappels de disposition pNFS comme décrit au paragraphe 12.5.5.2.

2.10.6.4 Questions de construction de COMPOUND et CB_COMPOUND

Les très grandes demandes et réponses peuvent poser les deux problèmes de gestion d'antémémoire (en particulier avec RDMA) et de mémoire tampon de réponse. Quand la session est créée (paragraphe 18.36) pour chaque canal (avant et de retour) le client et le serveur négocient la taille maximum de demande qu'ils vont envoyer ou traiter (ca_maxrequestsize) la taille maximum de réponse qu'ils vont retourner ou traiter (ca_maxresponsesize) et la taille maximum de réponse qu'ils vont mémoriser dans la mémoire tampon de réponse (ca_maxresponsesize_cached).

Si une demande excède ca_maxrequestsize, la réponse va avoir l'état NFS4ERR_REQ_TOO_BIG. Un répondeur PEUT retourner NFS4ERR_REQ_TOO_BIG comme état pour la première opération (SEQUENCE ou CB_SEQUENCE) dans la demande (ce qui signifie que les non opérations dans la demande exécutée et que l'état du créneau dans la mémoire tampon de réponse sont inchangés) ou il PEUT opter pour le retourner sur une opération suivante dans la même demande COMPOUND ou CB_COMPOUND (ce qui signifie que au moins une opération s'est exécutée et que l'état du créneau dans la mémoire tampon de réponse change bien). Le répondeur DEVRAIT établir NFS4ERR_REQ_TOO_BIG sur l'opération qui excède ca_maxrequestsize.

Si une réponse excède ca_maxresponsesize, la réponse va avoir l'état NFS4ERR_REP_TOO_BIG. Un répondeur PEUT retourner NFS4ERR_REP_TOO_BIG comme état pour la première opération (SEQUENCE ou CB_SEQUENCE) dans la demande, ou il PEUT opter pour le retourner sur une opération suivante (dans la même réponse COMPOUND ou

CB_COMPOUND). Un répondeur PEUT retourner NFS4ERR_REP_TOO_BIG dans la réponse à SEQUENCE ou CB_SEQUENCE, même si la réponse excéderait encore ca_maxresponsesize.

Si sa_cachethis ou csa_cachethis est VRAI, alors le répondeur DOIT mettre en antémémoire une réponse sauf si une erreur est retournée par l'opération SEQUENCE ou CB_SEQUENCE (voir le paragraphe 2.10.6.1.2). Si la réponse excède ca_maxresponsesize_cached (et si sa_cachethis ou csa_cachethis est VRAI) alors le serveur DOIT retourner NFS4ERR_REP_TOO_BIG_TO_CACHE. Même si NFS4ERR_REP_TOO_BIG_TO_CACHE (ou toute autre erreur pour ce sujet) est retournée sur une opération autre que la première opération (SEQUENCE ou CB_SEQUENCE) alors la réponse DOIT être mise en antémémoire si sa_cachethis ou csa_cachethis est VRAI. Par exemple, si un COMPOUND a onze opérations, incluant SEQUENCE, la cinquième opération est un RENAME, et la dixième opération est un READ pour un million d'octets, le serveur peut retourner NFS4ERR_REP_TOO_BIG_TO_CACHE sur la dixième opération. Comme le serveur a exécuté plusieurs opérations, en particulier le non idempotent RENAME, la demande du client de mettre en antémémoire la réponse doit être honorée pour le fonctionnement correct de la sémantique de "exactement une fois". Si le client réessaye la demande, le serveur aura mis en antémémoire une réponse qui contient des résultats pour dix des onze opérations demandées, la dixième opération ayant un état de NFS4ERR_REP_TOO_BIG_TO_CACHE.

Un client doit veiller à ce que, quand il envoie des opérations qui changent la bride de fichier actuelle (sauf pour PUTFH, PUTPUBFH, PUTROOTFH, et RESTOREFH) il n'excède pas la mémoire tampon maximum de réponse avant l'opération GETFH. Autrement, le client va devoir réessayer l'opération qui a changé la bride de fichier actuelle, afin d'obtenir la bride de fichier désirée. Pour l'opération OPEN (voir le paragraphe 18.16) le ré-essai n'est pas toujours une option disponible. Les lignes directrices suivantes sont conseillées pour le traitement des opérations qui changent la bride de fichier :

* Au sein de la même procédure COMPOUND, un client DEVRAIT envoyer GETFH immédiatement après une opération qui change la bride de fichier actuelle. Un client DOIT envoyer GETFH après une opération qui change la bride de fichier actuelle et est aussi non idempotente (par exemple, l'opération OPEN) sauf si l'opération est RESTOREFH. RESTOREFH est une exception, parce que même si elle n'est pas idempotente, la bride de fichier RESTOREFH produite est générée à partir d'une opération qui est soit idempotente (par exemple, PUTFH, LOOKUP) soit non idempotente (par exemple, OPEN, CREATE). Si l'origine n'est pas idempotente, alors parce que le client DOIT envoyer GETFH après l'opération d'origine, le client peut récupérer si RESTOREFH retourne une erreur.

* Un serveur PEUT retourner NFS4ERR_REP_TOO_BIG ou NFS4ERR_REP_TOO_BIG_TO_CACHE (si sa_cachethis est VRAI) sur une opération qui change la bride de fichier si la réponse serait trop grande sur la prochaine opération.

* Un serveur DEVRAIT retourner NFS4ERR_REP_TOO_BIG ou NFS4ERR_REP_TOO_BIG_TO_CACHE (si sa_cachethis est VRAI) sur une opération non idempotente qui change la bride de fichier si la réponse serait trop grande sur la prochaine opération, en particulier si l'opération est OPEN.

* Un serveur PEUT retourner NFS4ERR_UNSAFE_COMPOUND à une opération non idempotente qui change la bride de fichier actuelle, si il regarde la prochaine opération (dans la même procédure COMPOUND) et trouve qu'elle n'est pas GETFH. Le serveur DEVRAIT faire cela si il n'est pas capable de déterminer à l'avance si la taille totale de la réponse va excéder ca_maxresponsesize_cached ou ca_maxresponsesize.

2.10.6.5 Persistance

Comme la mémoire tampon de réponse est limitée, il est pratique que la mémoire tampon de réponse persiste à travers les redémarrages du serveur. Le répondeur DOIT conserver les informations suivantes si il a accepté de conserver la session (quand la session a été créée ; voir le paragraphe 18.36) :

* L'identifiant de session.
* Le tableau des créneaux incluant l'identifiant de séquence et la réponse mise en antémémoire pour chaque créneau.

Cela est suffisant pour qu'un répondeur fournisse la sémantique d'EOS pour toute demande qui a été envoyée et exécutée avant le redémarrage du serveur. Si le répondeur est un client, alors il n'est pas besoin qu'il conserve plus d'informations, sauf si le client va conserver tous les autres états à travers le redémarrage du client, auquel cas, le serveur ne va jamais voir de manifestation de protocole de niveau NFSv4.1 d'un redémarrage de client. Si le répondeur est un serveur, avec juste le tableau de créneaux et l'identifiant de session qui persistent, toutes les demandes que le client réessaye après le redémarrage du serveur vont retourner les résultats qui sont mémorisés dans la mémoire tampon de réponse, et toute nouvelle demande (c'est-à-dire, l'identifiant de séquence est supérieur de un à l'identifiant de séquence du créneau) DOIT être rejetée avec NFS4ERR_DEADSESSION (retourné par SEQUENCE). Une telle session est considérée comme morte. Un serveur PEUT ranimer une session après un redémarrage de serveur afin que la session accepte de nouvelles demandes ainsi que les réessais. Pour ranimer une session, le serveur doit conserver les informations supplémentaires suivantes à travers le redémarrage du serveur :

* L'identifiant de client. C'est un prérequis pour laisser le client créer plus de sessions associées au même identifiant de client que la session réanimée.
* L'identifiant de séquence de l'identifiant de client qui est utilisé pour créer des sessions (voir les paragraphes 18.35 et 18.36). C'est un prérequis pour laisser le client créer plus de sessions.
* Le principal qui a créé l'identifiant de client. Cela permet au serveur d'authentifier le client quand il envoie le EXCHANGE_ID.
* Le SSV, si la protection d'état SP4_SSV était spécifiée quand l'identifiant de client a été créé (voir le paragraphe 18.35). Ceci permet au client de créer de nouvelles sessions, et associe les connexions aux sessions nouvelles et existantes.
* Les propriétés de l'identifiant de client comme défini au paragraphe 18.35.

Une mémoire tampon de réponse persistante place certaines demandes chez le serveur. L'exécution de la séquence des opérations (commençant avec SEQUENCE) et le placement de son résultat dans la mémoire tampon persistante DOIT être atomique. Si un client réessaye une séquence d'opérations qui ont éte précédemment exécutées sur le serveur, les seuls résultats acceptables sont soit la réponse originale mise en antémémoire, soit une indication que l'identifiant de client ou de session a été perdu (indiquant une perte catastrophique de la mémoire tampon de réponse ou d'une session qui a été supprimée parce que le client a échoué à utiliser la session pendant une longue durée).

Un serveur pourrait avoir une défaillance et redémarrer au milieu d'une procédure COMPOUND qui contient une ou plusieurs opérations non idempotentes ou idempotentes mais modificatrices. Cela crée un défi encore plus fort pour l'exécution atomique et le placement de résultats dans la mémoire tampon de réponse. Une façon de voir le problème est comme une seule transaction consistant en chaque opération dans le COMPOUND suivie par la mémorisation du résultat dans la mémorisation persistante, puis finalement un engagement de transaction. Si il y a un échec avant l'engagement de la transaction, alors le serveur fait une reprise de la transaction. Si le serveur lui-même échoue, alors quand il redémarre, sa logique de récupération pourrait reprendre la transaction avant de démarrer le serveur NFSv4.1.

Bien que la description de la mise en œuvre de l'exécution atomique de la demande et la mise en antémémoire de la réponse sorte du domaine d'application du présent document, un exemple de mise en œuvre pour NFSv2 [RFC1094] est décrit dans [HANS].

2.10.7 Considérations de RDMA

Une discussion complète du fonctionnement des protocoles fondés sur RPC sur les transports RDMA figure dans la [RFC8166]. Une discussion du fonctionnement de NFSv4, incluant NFSv4.1, sur RDMA est dans la [RFC8267]. Lorsque RDMA est considéré, la présente spécification suppose l'utilisation d'une telle mise en couches ; elle traite seulement les questions de couche supérieure pertinentes pour faire le meilleur usage de RPC/RDMA.

2.10.7.1 Ressources de connexion RDMA

RDMA exige de ses utilisateurs qu'ils enregistrent la mémoire et envoient des mémoires tampons de taille et nombre spécifiques pour les opérations de réception.

L'enregistrement de mémoire peut être une opération relativement coûteuse en frais généraux, car elle exige d'épingler les mémoires tampon, l'allocation d'attributs (par exemple, lecture/écriture) et l'initialisation de traduction de matériel. Le pré enregistrement est désirable pour réduire les frais généraux. Ces enregistrements sont spécifiques des interfaces de matériel et même des points d'extrémité de connexion RDMA ; donc, la négociation de leurs limites est désirable pour gérer efficacement les ressources.

Suivant l'enregistrement de base, ces mémoires tampons doivent être envoyées par la couche RPC pout traiter les réceptions. Ces mémoires tampons restent utilisées par la mise en œuvre de RPC/NFSv4.1 ; leur taille et leur nombre doit être connu de l'homologue distant afin d'éviter des erreurs RDMA qui causeraient une erreur fatale sur la connexion RDMA.

NFSv4.1 gère des créneaux comme ressources sur la base de la session (voir le paragraphe 2.10) tandis que les connexions RDMA gèrent des crédits sur la base de la connexion. Cela signifie que afin qu'un homologue envoie des données sur RDMA à une mémoire tampon distante, il doit avoir à la fois un créneau NFSv4.1 et un crédit RDMA. Si plusieurs connexions RDMA sont associées à une session, alors si le nombre total de crédits sur toutes les connexions RDMA associées à la session est X, et le nombre de créneaux dans la session est Y, alors le nombre maximum de demandes en instance est le moindre de X et Y.

2.10.7.2 Contrôle de flux

Les précédentes versions de NFS ne fournissaient pas de contrôle de flux ; à la place, elles s'appuyaient sur la fenêtre fournie par les transports comme TCP pour réguler les demandes. Cela ne fonctionne pas avec RDMA, qui ne fournit pas de contrôle de flux d'opération et va terminer une connexion en erreur quand les limites sont dépassées. Des limites comme le nombre maximum de demandes en instance sont donc négociées quand une session est créée (voir le champ ca_maxrequests au paragraphe 18.36). Ces limites fournissent alors les maxima au sein desquels chaque connexion associée au canal de la session doit rester. Les connexions RDMA sont gérées dans ces limites comme décrit au paragraphe 3.3 de la [RFC8166] ; si il y a plusieurs connexions RDMA, alors le nombre maximum de demandes pour un canal va être divisé entre les connexions RDMA. Dit autrement, il incombe au répondeur de s'assurer que le nombre total de crédits RDMA sur toutes les connexions associées au canal du répondeur n'excède pas le nombre maximum de demandes en instance du canal.

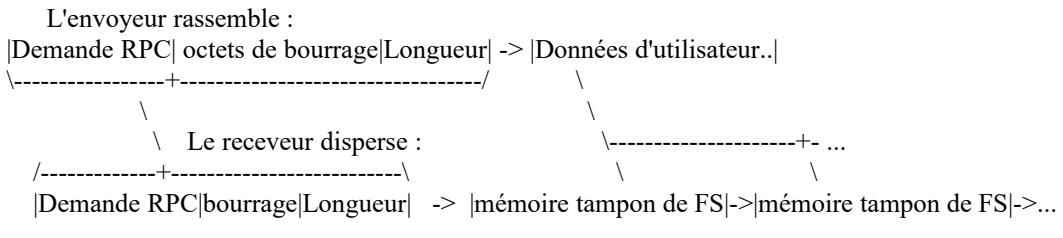
Les limites peuvent aussi être modifiées dynamiquement au choix du répondeur en manipulant certains paramètres présents dans chaque réponse NFSv4.1. De plus, l'opération de rappel CB_RECALL_SLOT (voir le paragraphe 20.8) peut être envoyée par un serveur à un client pour retourner des crédits RDMA au serveur, diminuant ainsi le nombre maximum de demandes qu'un client peut avoir en instance chez le serveur.

2.10.7.3 Bourrage

Le bourrage d'en-tête est demandé par chaque homologue à l'initialisation de session (voir l'argument ca_headerpadsize à CREATE_SESSION au paragraphe 18.36) et ensuite utilisé par la couche RPC RDMA, comme décrit dans la [RFC8166]. Le bourrage zéro est permis.

Le bourrage développe l'utile propriété que RDMA préserve l'alignement des données, même quand elles sont placées dans des mémoires tampon anonymes (non étiquetées). Si c'est demandé, les écritures en ligne du client vont insérer les octets de bourrage appropriés dans l'en-tête de la demande pour aligner la charge utile de données sur la frontière spécifiée. Le client est invité à ajouter un bourrage suffisant (jusqu'à la taille négociée) afin que le champ "data" de l'opération WRITE soit aligné. La plupart des serveurs peuvent faire un bon usage de ce bourrage, qui permet à certains d'eux d'enchaîner les mémoires tampon de réception d'une façon telle que toutes les données portées dans les demandes de client soient placées dans les mémoires tampon appropriées chez le serveur, prêtes pour le traitement du système de fichiers. La couche RPC du receveur ne rencontre pas de frais généraux en sautant les octets de bourrage, et les hautes performances de la couche RDMA font de l'insertion et de la transmission de bourrage chez l'envoyeur une optimisation significative. De cette façon, le besoin que les serveurs effectuent un Read RDMA pour satisfaire toutes les écritures du client sauf la plus grande est masqué. Un avantage supplémentaire est la réduction des allers-retours de message sur le réseau -- un compromis potentiellement bon lorsque de la latence est présente.

La valeur à choisir pour le bourrage est soumise à un certain nombre de critères. Les principales sources de données de longueur variable dans l'en-tête RPC sont les informations d'authentification, dont la forme est déterminée par le client, éventuellement en réponse à la spécification du serveur. Le contenu des COMPOUND, les tailles de chaînes comme celles passées à RENAME, etc. entrent toutes dans la détermination d'une taille maximale de demande NFSv4.1 et donc de la taille minimale de la mémoire tampon. Le client doit choisir avec soin sa valeur offerte, afin d'éviter de surcharger le serveur, et vice versa. L'avantage d'une valeur de bourrage approprié est de meilleures performances.



Dans le cas ci-dessus, le serveur peut recycler les mémoires tampon non utilisés pour les prochaines réceptions envoyées si elles ne sont pas utilisées par les demandes reçues actuelles, ou peut passer les mémoires tampon maintenant complètes par référence pour un traitement d'écriture normal. Pour un serveur qui peut l'utiliser, cela supprime le besoin de copier les données entrantes, sans recourir à des annonces et une gestion compliquées de bout en bout de mémoire tampon. Cela inclut la plupart des conceptions de serveur fondées sur le noyau et intégrées, parmi de nombreuses autres. Le client peut effectuer des optimisations similaires, si c'est désiré.

2.10.7.4 Transports RDMA duels et non RDMA

Certains transports RDMA (par exemple, [RFC5040]) permettent une phase (non RDMA) de "flux directs", où du trafic

ordinaire pourrait s'écouler avant de "revenir" au mode RDMA, en commençant par le trafic RDMA. Certains transports RDMA commencent les connexions toujours en mode RDMA. NFSv4.1 permet, mais ne suppose pas, une phase de flux directs avant le mode RDMA. Quand une connexion est associée à une session, le client et le serveur négocient si la connexion est utilisé en mode RDMA ou non (voir les paragraphes 18.36 et 18.34).

2.10.8 Sécurité de session

2.10.8.1 Sécurité de rappel de session

Via l'association de session/connexion, NFSv4.1 améliore la sécurité fournie par NFSv4.0 pour le canal de retour. La connexion est initiée par le client (voir le paragraphe 18.34) et soumise aux même vérifications de pare-feu et d'acheminement que le canal avant. Au choix du client (voir le paragraphe 18.35) l'association de connexion est pleinement authentifiée avant d'être activée (voir le paragraphe 18.34). Le trafic du serveur sur le canal de retour est authentifié exactement comme le client le spécifie (voir le paragraphe 2.10.8.2).

2.10.8.2 Sécurité de canal de retour RPC

Quand le client NFSv4.1 établit le canal de retour, il informe le serveur des nuances de sécurité et des principaux à utiliser quand il envoie les demandes. Si la nuance de sécurité est RPCSEC_GSS, le client exprime le principal sous la forme d'un contexte RPCSEC_GSS établi. Le serveur est libre d'utiliser toute combinaison de nuance/principal qu'offre le client, mais il NE DOIT PAS utiliser des combinaisons non offertes. De cette façon, le client n'a pas besoin de fournir un principal GSS cible pour le canal de retour comme il le faisait avec NFSv4.0, ni le serveur de mettre en œuvre un initiateur RPCSEC_GSS comme il le faisait avec NFSv4.0 [RFC3530].

Les opérations CREATE_SESSION (paragraphe 18.36) et BACKCHANNEL_CTL (paragraphe 18.33) permettent au client de spécifier des combinaisons de nuance/principal.

Noter aussi que le mode de protection d'état SP4_SSV (voir les paragraphes 18.35 et 2.10.8.3) a l'avantage supplémentaire de fournir des contextes RPCSEC_GSS déduits du SSV (paragraphe 2.10.9).

2.10.8.3 Protection contre les changements d'état non autorisés

Comme décrit à ce point de la spécification, le modèle d'état de NFSv4.1 est vulnérable à un attaquant qui envoie une opération SEQUENCE avec un identifiant de session falsifié et un identifiant de créneau qu'il s'attend à ce que le client légitime utilise ensuite. Quand le client légitime utilise l'identifiant de créneau avec le même numéro de séquence, le serveur retourne le résultat de l'attaquant provenant de la mémoire tampon de réponse, qui perturbe le client légitime et donc lui dénie le service. De même, un attaquant pourrait envoyer un CREATE_SESSION avec un identifiant de client falsifié pour créer une nouvelle session associée à l'identifiant de client. L'attaquant pourrait envoyer des demandes en utilisant la nouvelle session qui change l'état de verrouillage, comme des opérations LOCKU pour libérer les verrous que le client légitime a acquis. Établir une politique de sécurité sur le fichier qui exige des accréditifs RPCSEC_GSS quand il manipule l'état du fichier est un contournement potentiel, mais présente l'inconvénient d'empêcher un client légitime de libérer l'état quand RPCSEC_GSS est obligé de le faire, mais un contexte GSS ne peut pas être obtenu (éventuellement parce que l'utilisateur a déconnecté le client).

NFSv4.1 fournit trois options à un client pour la protection de l'état, qui sont spécifiées quand un client crée un identifiant de client via EXCHANGE_ID (paragraphe 18.35).

La première (SP4_NONE) est de simplement de renoncer à la protection d'état.

Les deux autres options (SP4_MACH_CRED et SP4_SSV) partagent plusieurs traits :

* Un accréditif fondé sur RPCSEC_GSS est utilisé pour authentifier l'identifiant de client et les opérations de maintenance de session incluant la création et la destruction d'une session, l'association d'une connexion à la session, et la destruction de l'identifiant de client.

* Parce que RPCSEC_GSS est utilisé pour authentifier l'identifiant de client et la maintenance de session, l'attaquant ne peut pas associer une connexion pirate avec une session légitime, ou associer une session pirate avec un identifiant de client légitime afin d'altérer de façon malveillante l'état de verrouillage de l'identifiant de client via CLOSE, LOCKU, DELEGRETURN, LAYOUTRETURN, etc.

* Dans les cas où les politiques de sécurité du serveur sur une portion de son espace de noms exigent l'authentification

RPCSEC_GSS, un client peut devoir utiliser un accréditif RPCSEC_GSS pour supprimer l'état par fichier (par exemple, LOCKU, CLOSE, etc.). Le serveur peut exiger que le principal qui supprime l'état réponde à certains critères (par exemple, le principal pourrait devoir être le même que celui qui a acquis l'état). Cependant, le client pourrait ne pas avoir de contexte RPCSEC_GSS pour ce principal, et pourrait n'être pas capable de créer un tel contexte (peut-être parce que l'utilisateur s'est déconnecté). Quand le client établit la protection SP4_MACH_CRED ou SP4_SSV, il peut spécifier une liste d'opérations que le serveur DOIT permettre en utilisant l'accréditif de machine (si SP4_MACH_CRED est utilisé) ou l'accréditif SSV (si SP4_SSV est utilisé).

L'option de protection d'état SP4_MACH_CRED utilise un accréditif de machine où le principal qui crée l'identifiant de client DOIT aussi être le principal qui effectue les opérations d'identifiant de client et de maintenance de session. La sécurité de l'approche de protection d'état par accréditif de machine dépend entièrement de la sauvegarde de l"accréditif de machine. En supposant une sauvegarde appropriée en utilisant l'accréditif par machine pour des opérations comme CREATE_SESSION, BIND_CONN_TO_SESSION, DESTROY_SESSION, et DESTROY_CLIENTID vont empêcher un attaquant d'associer une connexion pirate à une session, ou d'associer une session pirate à un identifiant de client.

Il y a au moins trois scénarios pour l'option SP4_MACH_CRED :

1. L'administrateur de système configure un accréditif unique permanent par machine pour un des mécanismes GSS obligatoires (par exemple, si Kerberos V5 est utilisé, un "keytab" contenant un principal déduit d'un nom d'hôte client pourrait être utilisé).

2. Le client est utilisé par un seul utilisateur, et donc l'identifiant de client et ses sessions sont utilisés juste par cet utilisateur. Si l'accréditif de l'utilisateur arrive à expiration, alors la maintenance de session et d'identifiant de client ne peut pas se faire, mais comme le client a un seul utilisateur, seul cet utilisateur est gêné.

3. Le client physique a plusieurs utilisateurs, mais la mise en œuvre de client a un unique identifiant de client pour chacun. C'est effectivement le même cas que le second scénario, mais un inconvénient est que à chaque utilisateur doit être allouée au moins une session, donc l'approche souffre d'un manque d'économie.

L'option de protection SP4_SSV utilise le SSV (paragraphe 1.7) via le RPCSEC_GSS et le mécanisme SSV GSS (paragraphe 2.10.9) pour protéger l'état d'une attaque. L'option de protection SP4_SSV est destinée à une situation où un client qui a plusieurs utilisateurs actifs et un administrateur de système qui veut éviter la charge d'installer un accréditif de machine permanent sur chaque client. Le SSV est établi et mis à jour sur le serveur via SET_SSV (voir le paragraphe 18.47). Pour empêcher l'espionnage, un client DEVRAIT envoyer SET_SSV via RPCSEC_GSS avec le service de confidentialité. Plusieurs aspects de SSV font qu'il est impossible à un attaquant de deviner le SSV, et donc d'associer des connexions pirates à une session, et des sessions pirates à un identifiant de client :

* Les arguments à SET_SSV et ses résultats incluent des résumés respectivement de l'ancien et du nouveau SSV.

* Parce que la valeur initiale du SSV est zéro, donc connue, le client qui opte pour la protection SP4_SSV et choisit d'appliquer la protection SP4_SSV à BIND_CONN_TO_SESSION et CREATE_SESSION DOIT envoyer au moins une opération SET_SSV avant la première opération BIND_CONN_TO_SESSION ou avant la seconde opération CREATE_SESSION sur un identifiant de client. Si il ne le fait pas, le mécanisme SSV ne va pas générer de jetons (paragraphe 2.10.9). Un client DEVRAIT envoyer SET_SSV aussitôt qu'une session est créée.

* Une demande SET_SSV ne remplace pas le SSV par l'argument à SET_SSV. À la place, le SSV actuel sur le serveur est l'objet d'une comparaison logique exclusive (OUX) avec l'argument de SET_SSV. Chaque fois qu'un nouveau principal utilise un identifiant de client pour la première fois, le client DEVRAIT envoyer un SET_SSV avec les accréditifs de RPCSEC_GSS de ce principal, avec le service RPCSEC_GSS réglé à RPC_GSS_SVC_PRIVACY.

Voici les types d'attaques qui peuvent être tentées par un attaquant nommé Eve sur une victime nommée Bob, et comment la protection SP4_SSV déjoue chaque attaque :

* Supposons que Eve est le premier utilisateur à s'enregistrer sur un client légitime. L'utilisation par Eve d'un système de fichiers NFSv4.1 va causer la création par le client légitime d'un identifiant de client avec protection SP4_SSV, spécifiant que l'opération BIND_CONN_TO_SESSION DOIT utiliser l'accréditif SSV. L'utilisation par Eve du système de fichiers cause aussi la création d'un SSV. L'opération SET_SSV qui crée le SSV va être protégée par le contexte RPCSEC_GSS créé par le client légitime, qui utilise le principal GSS et les accréditifs de Eve. Eve peut espionner sur le réseau lorsque son contexte RPCSEC_GSS est créé et que le SET_SSV qui utilise son contexte est envoyé. Même si le client légitime envoie le SET_SSV avec RPC_GSS_SVC_PRIVACY, parce que Eve connaît ses propres accréditifs, elle peut déchiffrer le SSV. Eve peut calculer un accréditif RPCSEC_GSS que BIND_CONN_TO_SESSION va

accepter, et donc associer une nouvelle connexion à la session légitime. Eve peut changer l'identifiant de créneau et l'état de séquence d'une session légitime, et/ou l'état de SSV, de telle façon que quand Bob accède au serveur via le même client légitime, le client légitime va être incapable d'utiliser la session.

Le seul recours du client est de créer un nouvel identifiant de client à utiliser par Bob, et établir un nouveau SSV pour l'identifiant de client. Le client va être incapable de supprimer le vieil identifiant de client, et va laisser le prêt expirer sur le vieil identifiant de client.

Une fois que le client légitime a établit un SSV sur la nouvelle session en utilisant le contexte RPCSEC_GSS de Bob, Eve peut utiliser la nouvelle session via le client légitime, mais elle ne peut pas perturber Bob. De plus, parce que le client DEVRAIT avoir modifié le SSV du fait que Eve utilise la nouvelle session, Bob ne peut pas prendre sa revanche sur Eve en associant une connexion pirate à la session.

La question est comment le client légitime va t-il détecter que Eve avait capturé la vieille session ? Quand le client détecte qu'un nouveau principal, Bob, veut utiliser la session, il DEVRAIT avoir envoyé un SET_SSV, ce qui conduit aux sous scénarios suivants :

- Supposons qu'à partir de la connexion pirate, Eve envoie un SET_SSV avec le même identifiant de créneau et identifiant de séquence que le client légitime utilisera ensuite. Le serveur va supposer que le SET_SSV envoyé avec les accréditifs de Bob est un ré-essai, et va retourner au client légitime la réponse qu'il a envoyé à Eve. Cependant, sauf si Eve peut deviner correctement le SSV que le client légitime va utiliser, les vérifications de résumé dans la réponse SET_SSV vont échouer. C'est une indication pour le client que la session a apparemment été capturée.

- Autrement, Eve envoie un SET_SSV avec un identifiant de créneau différent de celui que le client légitime utilise pour son SET_SSV. Alors la vérification de résumé du SET_SSV envoyé avec les accréditifs de Bob échoue sur le serveur, et l'erreur retournée au client rend évident que la session a été capturée.

- Autrement, Eve envoie une opération autre que SET_SSV, mais avec le même identifiant de créneau et séquence que le client légitime utilise pour son SET_SSV. Le serveur retourne au client légitime la réponse qu'il a envoyé à Eve. Le client voit que la réponse n'est pas du tout celle qu'il attend. Le client suppose soit la capture de la session soit une erreur du serveur, et de toute façon détruit la vieille session.

* Eve associe une connexion pirate à la session comme ci-dessus, et détruit alors la session. Là encore, Bob va utiliser le serveur à partir du client légitime, qui envoie un SET_SSV en utilisant les accréditifs de Bob. Le client reçoit une erreur qui indique que la session n'existe pas. Quand le client essaye de créer une nouvelle session, cela échoue parce que le SSV qu'il a ne correspond pas à celui qu'a le serveur, et maintenant le client sait que la session a été capturée. Le client légitime établit un nouvel identifiant de client.

* Si Eve crée une connexion avant que le client légitime établisse un SSV, parce que la valeur initiale du SSV est zéro et donc connue, Eve peut envoyer un SET_SSV qui va réussir la vérification de résumé. Cependant, parce que la nouvelle connexion n'a pas été associée à la session, le SET_SSV est rejeté pour cette raison.

En résumé, la perturbation d'état d'un attaquant quand la protection SP4_SSV est utilisée est limitée à la période de formation d'un identifiant de client, à sa première session, et à l'établissement du SSV. Une fois qu'un utilisateur non malveillant utilise l'identifiant de client, le client détecte rapidement toute capture et rectifie la situation. Si un utilisateur non malveillant réussit à modifier le SSV, l'attaquant ne peut pas utiliser des opérations NFSv4.1 pour perturber l'utilisateur non malveillant.

Noter que ni l'approche de protection SP4_MACH_CRED ni celle de SP4_SSV n'empêchent la capture d'une connexion de transport qui a été précédemment associée à une session. Si le but d'une stratégie de contre menace est d'empêcher la capture de connexion, l'utilisation de IPsec est RECOMMANDÉ.

Si une capture de connexion se produit, l'attaquant pourrait en théorie changer l'état de verrouillage et impacter négativement le service aux clients légitimes. Cependant, si le serveur est configuré à exiger l'utilisation de RPCSEC_GSS avec l'intégrité ou la confidentialité sur les objets de fichier affectés, et si la capacité EXCHGID4_FLAG_BIND_PRINC_STATEID (paragraphe 18.35) est activée, cela va déjouer les tentatives non autorisées de changer l'état de verrouillage.

2.10.9 Mécanisme GSS de vérificateur d'état secret (SSV)

Le vérificateur d'état secret (SSV, *Secret State Verifier*) fournit la clé secrète pour un mécanisme GSS interne à NFSv4.1 que NFSv4.1 utilise pour la protection de l'état. Les contextes pour ce mécanisme ne sont pas établis via le protocole RPCSEC_GSS. À la place, les contextes sont créés automatiquement quand l'EXCHANGE_ID spécifie la protection SP4_SSV. Les seuls jetoms définis sont PerMsgToken (émis par GSS_GetMIC) et SealedMessage (émis par GSS_Wrap).

L'OID de mécanisme pour le mécanisme de SSV est "iso.org.dod.internet.private.enterprise.Michael Eisler.nfs.ssv_mech" (1.3.6.1.4.1.28882.1.1). Bien que le mécanisme de SSV ne définisse pas de jeton de contexte initial, l'OID peut être utilisé pour que les serveurs indiquent que le mécanisme de SSV est acceptable chaque fois que le client envoie une opération SECINFO ou SECINFO_NO_NAME (voir le paragraphe 2.6).

Le mécanisme de SSV définit quatre sous clés déduites de la valeur de SSV. Chaque fois que SET_SSV est invoqué, les sous clés sont recalculées par le client et le serveur. Le calcul de chacune des quatre sous clés dépend de chacune des quatre valeurs respectives de ssv_subkey4 énumérées. Le calcul utilise l'algorithme HMAC [RFC2104], avec le SSV courant comme clé, l'algorithme de hachage unidirectionnel négocié par l'EXCHANGE_ID, et le texte d'entrée comme représenté par la valeur d'énumération codée en XDR pour cette sous clé de type de données ssv_subkey4. Si la longueur du résultat de l'algorithme HMAC excéde la longueur de clé de l'algorithme de chiffrement (qui est aussi négocié par l'EXCHANGE_ID) alors la sous clé DOIT être tronquée à partir du résultat du HMAC, c'est-à-dire, si la sous clé est de N octets, alors les N premiers octets du résultat du HMAC DOIVENT être utilisés pour la sous clé. La spécification de EXCHANGE_ID déclare que la longueur du résultat de l'algorithme HMAC NE DOIT PAS être moins que la longueur de la sous clé nécessaire pour l'algorithme de chiffrement (voir le paragraphe 18.35).

```
/* Entrée pour calculer les sous clés */
enum ssv_subkey4 {
        SSV4_SUBKEY_MIC_I2T     = 1,
        SSV4_SUBKEY_MIC_T2I     = 2,
        SSV4_SUBKEY_SEAL_I2T    = 3,
        SSV4_SUBKEY_SEAL_T2I    = 4.
};
```

La sous clé déduite de SSV4_SUBKEY_MIC_I2T est utilisée pour calculer les codes d'intégrité de message (MIC, *message integrity code*) générés par le client NFSv4.1, qu'ils fassent partie d'une demande sur le canal avant ou d'une réponse sur le canal de retour. La sous clé déduite de SSV4_SUBKEY_MIC_T2I est utilisée pour les MIC originaires du serveur NFSv4.1. La sous clé déduite de SSV4_SUBKEY_SEAL_I2T est utilisée pour chiffrer le texte originaire du client NFSv4.1, et la sous clé déduite de SSV4_SUBKEY_SEAL_T2I est utilisée pour le chiffrement du texte originaire du serveur NFSv4.1.

La description de PerMsgToken est fondée sur une définition XDR :

```
/* Entrée pour calculer smt_hmac */
struct ssv_mic_plain_tkn4 {
  uint32_t      smpt_ssv_seq;
  opaque        smpt_orig_plain<>;
};

/* Jeton de SSV GSS PerMsgToken */
struct ssv_mic_tkn4 {
  uint32_t      smt_ssv_seq;
  opaque        smt_hmac<>;
};
```

Le champ smt_hmac est un HMAC calculé en utilisant la sous clé déduite de SSV4_SUBKEY_MIC_I2T ou de SSV4_SUBKEY_MIC_T2I comme clé, l'algorithme de hachage unidirectionnel comme négocié par EXCHANGE_ID, et le texte d'entrée représenté pas des données de type ssv_mic_plain_tkn4. Le champ smpt_ssv_seq est le même que smt_ssv_seq. Le champ smpt_orig_plain est l'entrée de "message" passée à GSS_GetMIC() (voir le paragraphe 2.3.1 de la [RFC2743]). L'appelant de GSS_GetMIC() fournit un pointeur sur une mémoire tampon contenant le texte en clair. Le mécanisme de point d'entrée de SSV pour GSS_GetMIC() code cela en un dispositif opaque, et le codage va inclure une longueur initiale de quatre octets, plus le bourrage nécessaire. La valeur de smpt_ssv_seq codée en XDR va être ajoutée devant cela, constituant un codage XDR d'une valeur de type de données ssv_mic_plain_tkn4, qui à son tour est l'entrée du HMAC.

Le jeton émis par GSS_GetMIC() est codé en XDR et de type de données XDR ssv_mic_tkn4. Le champ smt_ssv_seq vient du numéro de séquence SSV, qui est égal à un après que SET_SSV (paragraphe 18.47) est invoqué pour la première fois sur un identifiant de client. Ensuite, le numéro de séquence SSV est incrémenté sur chaque SET_SSV. Donc, smt_ssv_seq représente la version du SSV au moment où GSS_GetMIC() a été invoqué. Comme noté au paragraphe 18.35, le client et le serveur peuvent maintenir plusieurs versions concurrentes du SSV. Cela permet que le SSV soit changé sans mettre à la suite toutes les invocations de RPC qui utilisent le mécanisme de SSV avec des opérations SET_SSV. Une fois le HMAC calculé, il est codé en XDR dans smt_hmac, qui va inclure une longueur initiale de quatre octets, et tout le bourrage nécessaire. La valeur de smt_ssv_seq codée en XDR va être ajoutée devant cela.

La description de SealedMessage est fondée sur une définition XDR :

```
/* Entrée pour calculer ssct_encr_data et ssct_hmac */
struct ssv_seal_plain_tkn4 {
  opaque        sspt_confounder<>;
  uint32_t      sspt_ssv_seq;
  opaque        sspt_orig_plain<>;
  opaque        sspt_pad<>;
};

/* Jeton SSV GSS SealedMessage */
struct ssv_seal_cipher_tkn4 {
  uint32_t     ssct_ssv_seq;
  opaque       ssct_iv<>;
  opaque       ssct_encr_data<>;
  opaque       ssct_hmac<>;
};
```

Le jeton émis par GSS_Wrap() est codé en XDR et de type de données XDR ssv_seal_cipher_tkn4.

Le champ ssct_ssv_seq a la même signification que smt_ssv_seq.

Le champ ssct_encr_data est le résultat du chiffrement d'une valeur du type de données ssv_seal_plain_tkn4 codé en XDR. La clé de chiffrement est la sous clé déduite de SSV4_SUBKEY_SEAL_I2T ou de SSV4_SUBKEY_SEAL_T2I, et l'algorithme de chiffrement est celui négocié par EXCHANGE_ID.

Le champ ssct_iv est la valeur d'initialisation (IV) pour l'algorithme de chiffrement (si applicable) et est envoyé en clair. Le contenu et la taille de l'IV DOIT se conformer à la spécification de l'algorithme de chiffrement. Par exemple, l'algorithme id-aes256-CBC DOIT utiliser une valeur d'initialisation de 16 octets, qui DOIT être imprévisible pour chaque instance d'une valeur de type de données ssv_seal_plain_tkn4 chiffrée avec une clé SSV particulière.

Le champ ssct_hmac est le résultat du calcul d'un HMAC utilisant la valeur de type de données ssv_seal_plain_tkn4 codé en XDR comme texte d'entrée. La clé est la sous clé déduite de SSV4_SUBKEY_MIC_I2T ou de SSV4_SUBKEY_MIC_T2I, et l'algorithme de hachage unidirectionnel est celui négocié par l'EXCHANGE_ID.

Le champ sspt_confounder est une valeur aléatoire.

Le champ sspt_ssv_seq est le même que ssvt_ssv_seq.

Le champ sspt_orig_plain est le texte en clair original et est l'entrée "input_message" passée au GSS_Wrap() (voir le paragraphe 2.3.3 de la [RFC2743]). Comme avec le traitement du texte en clair par le mécanisme de point d'entrée de GSS_GetMIC() du SSV, le point d'entrée pour GSS_Wrap() attend un pointeur sur le texte en clair, et va coder en XDR un dispositif opaque dans sspt_orig_plain représentant le texte en clair, avec les autres champs d'une instance de type de données ssv_seal_plain_tkn4.

Le champ sspt_pad est présent pour prendre en charge les algorithmes de chiffrement qui exigent que les entrées soient dans des blocs de taille fixe. Le contenu de sspt_pad est rempli de zéros sauf pour la longueur. On note que le codage XDR de ssv_seal_plain_tkn4 contient trois dispositifs de longueur variable, et donc chaque dispositif consomme quatre octets pour une longueur de dispositif, et chaque dispositif qui suit la longueur est toujours bourré à un multiple de quatre octets selon la norme XDR.

Par exemple, supposons que l'algorithme de chiffrement utilise des blocs de 16 octets, et que sspt_confounder soit long de trois octets, et le champ sspt_orig_plain soit de 15 octets. Le codage XDR de sspt_confounder utilise huit octets (4 + 3 + 1 - octet de bourrage) le codage XDR de sspt_ssv_seq utilise quatre octets, le codage XDR de sspt_orig_plain utilise 20 octets (4 + 15 + 1 - octet de bourrage) et le plus petit codage XDR du champ sspt_pad fait quatre octets. Cela fait un total de 36 octets. Le prochain multiple de 16 est 48 ; donc, le champ Longueur de sspt_pad needs va être réglé à 12 octets, ou un codage total de 16 octets. Le nombre total d'octets codés en XDR est donc 8 + 4 + 20 + 16 = 48.

GSS_Wrap() émet un jeton qui est un codage XDR d'une valeur de type de données ssv_seal_cipher_tkn4. Noter que sans considération de si l'invocateur de GSS_Wrap() demande la confidentialité, le jeton a toujours la confidentialité. C'est parce que le mécanisme de SSV est pour RPCSEC_GSS, et RPCSEC_GSS ne produit jamais de jetons GSS_wrap() sans la confidentialité.

Il y a un SSV par identifiant de client. Il y a un seul contexte GSS pour une paire identifiant de client / SSV. Toutes les brides RPCSEC_GSS de mécanisme de SSV d'une paire d'identifiant de client / SSV partagent le même contexte GSS. Les contextes GSS de SSV n'expirent pas sauf quand le SSV est détruit (les causes vont inclure la destruction de l'identifiant de client ou un redémarrage de serveur). Comme un des objets de l'expiration de contexte est de remplacer les clés qui ont été utilisées pendant "trop longtemps", donc vulnérables à la compromission par une attaque en force brute ou par accident, le client peut remplacer la clé de SSV en envoyant des opérations périodiques SET_SSV, qui est faite en tournant à travers les accréditifs RPCSEC_GSS de différents utilisateurs. De cette façon, le SSV est remplacé sans détruire les contextes GSS du SSV.

Les brides RPCSEC_GSS de SSV peuvent être terminées ou supprimées par le serveur à tout moment, et l'opération EXCHANGE_ID peut être utilisée pour créer plus de brides RPCSEC_GSS de SSV. L'expiration des brides RPCSEC_GSS de SSV n'implique pas que le SSV ou son contexte GSS a expiré.

Le client DOIT établir un SSV via SET_SSV avant que le contexte GSS de SSV puisse être utilisé pour émettre des jetons à partir de GSS_Wrap() et GSS_GetMIC(). Si SET_SSV n'a pas encore été invoqué avec succès, les tentatives d'émettre des jetons DOIVENT échouer.

Le mécanisme de SSV ne prend pas en charge la détection de répétitions et le séquençage dans ses jetons parce que RPCSEC_GSS n'utilise pas ces caractéristiques (voir "Demandes de création de contexte" au paragraphe 5.2.2 de la [RFC2203]). Cependant, le paragraphe 2.10.10 discute des considérations particulières pour le mécanisme de SSV quand il est utilisé avec RPCSEC_GSS.

2.10.10 Considérations sur la sécurité pour RPCSEC_GSS avec le mécanisme de SSV

Quand un identifiant de client est créé avec la protection d'état SP4_SSV (voir le paragraphe 18.35) il est permis au client d'associer plusieurs brides RPCSEC_GSS au seul contexte GSS de SSV (voir le paragraphe 2.10.9). À cause de la façon dont RPCSEC_GSS (les deux versions 1 et 2, voir les [RFC2203] et [RFC5403]) calculent le vérificateur de la réponse, un soin particulier doit être porté par la mise en œuvre de client NFSv4.1 à prévenir les attaques par un interposé. Le vérificateur d'une réponse RPCSEC_GSS est le résultat de GSS_GetMIC() appliqué à la valeur d'entrée du champ seq_num de l'accréditif RPCSEC_GSS (type de données rpc_gss_cred_ver_1_t) (voir le paragraphe 5.3.3.2 de la [RFC2203]). Si plusieurs brides RPCSEC_GSS partagent le même contexte GSS, alors si une bride est utilisée pour envoyer une demande avec la même valeur de seq_num qu'une autre bride, un attaquant pourrait bloquer la réponse, et la remplacer par le vérificateur utilisé pour l'autre bride.

Il y a plusieurs façons d'empêcher l'attaque sur le vérificateur SSV RPCSEC_GSS dans la réponse. La plus simple est estimée être la suivante :

* Chaque fois qu'une ou plusieurs nouvelles brides RPCSEC_GSS de SSV sont créées via EXCHANGE_ID, le client DEVRAIT envoyer une opération SET_SSV pour modifier le SSV. En changeant le SSV, les nouvelles brides ne vont pas résulter en la réutilisation d'un vérificateur RPCSEC_GSS de SSV dans une réponse.

* Quand un demandeur décide d'utiliser N brides RPCSEC_GSS de SSV, il DEVRAIT allouer une gamme unique et non chevauchante de seq_nums à chaque bride RPCSEC_GSS de SSV. La taille de chaque gamme DEVRAIT être égale à MAXSEQ / N (voir à la Section 5 de la [RFC2203] la définition de MAXSEQ). Quand une bride RPCSEC_GSS de SSV atteint son maximum, elle DEVRAIT forcer le répondeur à détruire la bride par l'envoi d'une demande RPC NULLE avec le numéro de séquence réglé à MAXSEQ + 1 (voir le paragraphe 5.3.3.3 de la [RFC2203]).

* Quand le demandeur veut augmenter ou diminuer N, il DEVRAIT forcer le répondeur à détruire toutes les N brides en

envoyant une demande RPC NULLE sur chaque bride avec le numéro de séquence réglé à MAXSEQ + 1. Si le demandeur est le client, il DEVRAIT envoyer une opération SET_SSV avant d'utiliser de nouvelles brides. Si le demandeur est le serveur, alors le client DEVRAIT envoyer une opération SET_SSV quand il détecte que le serveur l'a forcé à détruire la bride RPCSEC_GSS de SSV d'un canal de retour. Par l'envoi d'une opération SET_SSV, le SSV va changer, et donc il va être indisponible à l'attaquant pour réussir à répéter un vérificateur précédent dans une réponse au demandeur.

Noter que si le répondeur crée avec soin les brides RPCSEC_GSS de SSV, le risque relatif à un interposé qui glisserait un accréditif RPCSEC_GSS falsifié de SSV avec un vérificateur pour une autre bride n'existe pas. C'est parce que le vérificateur dans une demande RPCSEC_GSS est calculé à partir de l'entrée qui inclut à la fois la bride RPCSEC_GSS et le numéro de séquence (voir le paragraphe 5.3.1 de la [RFC2203]). Pourvu que le répondeur prenne soin d'éviter de réutiliser la valeur d'une bride RPCSEC_GSS qu'il a créée, comme en incluant un numéro de génération dans la bride, l'interposé ne sera pas capable de réussir à ré-exécuter un vérificateur précédent dans la demande à un répondeur.

2.10.11 Mécanique de session - état permanent

2.10.11.1 Obligations du serveur

Le serveur a l'obligation principale de surveiller les ressources de l'état de canal de retour que le client a créé pour le serveur (contextes RPCSEC_GSS et connexions de canal de retour). Si les ressources disparaissent, le serveur prend les actions spécifiées au paragraphe 2.10.13.2.

2.10.11.2 Obligations du client

Le client DEVRAIT respecter les obligations suivantes pour utiliser la session :

* Empêcher une session nécessaire de devenir inactive sur le serveur. Un client qui exige une session mais n'envoie cependant pas d'opérations risque de voir la session détruite par le serveur. C'est parce que les sessions consomment des ressources, et des limitations de ressources peuvent forcer le serveur à détruire une session inactive. Un serveur PEUT considérer qu'une session est inactive si le client ne l'a pas utilisée avant l'expiration du temporisateur d'inactivité de session (paragraphe 2.10.12).

* Détruire la session quand elle n'est pas nécessaire. Si un client a plusieurs sessions, dont une n'a pas de demandes attendant des réponses, et a été inactive pendant un certain temps, il DEVRAIT détruire la session.

* Maintenir les contextes GSS et les brides RPCSEC_GSS pour le canal de retour. Si le client demande au serveur d'utiliser la nuance de sécurité RPCSEC_GSS pour les rappels, alors il a besoin d'être sûr que les brides RPCSEC_GSS et/ou leurs contextes GSS qui sont passés au serveur via BACKCHANNEL_CTL ou CREATE_SESSION ne sont pas expirés

* Préserver une connexion pour un canal de retour. Le serveur exige un canal de retour afin de rappeler en douceur l'état rappelable ou notifier certains événements au client. Noter que si la connexion n'est pas utilisée pour le canal avant, le client n'a pas de moyen de dire si la connexion est encore active (par exemple, le serveur a redémarré sans envoyer de déconnexion). La charge de déterminer si la connexion du canal de retour est active, et d'indiquer dans la réponse à une opération SEQUENCE quand la dernière connexion associée au canal de retour d'une session s'est déconnectée, est sur le serveur, pas sur le client.

2.10.11.3 Étapes d'établissement de session par le client

Si le client n'a pas d'identifiant de client, le client envoie un EXCHANGE_ID pour établir un identifiant de client. Si il opte pour la protection SP4_MACH_CRED ou SP4_SSV, dans la liste des opérations spo_must_enforce, il DEVRAIT au minimum spécifier CREATE_SESSION, DESTROY_SESSION, BIND_CONN_TO_SESSION, BACKCHANNEL_CTL, et DESTROY_CLIENTID. Si il opte pour la protection SP4_SSV, le client doit demander des brides RPCSEC_GSS fondées sur le SSV.

Le client utilise l'identifiant de client pour envoyer un CREATE_SESSION sur une connexion au serveur. Le résultat de CREATE_SESSION indique si le serveur va ou non faire durer la mémoire tampon de réponse de session sur un redémarrage de serveur, et le client le note pour une future référence.

Si le client a spécifié la protection d'état SP4_SSV quand l'identifiant de client a été créé, alors il DEVRAIT envoyer SET_SSV dans le premier COMPOUND après la création de la session. Chaque fois qu'un nouveau principal utilise

l'identifiant de client, il DEVRAIT envoyer encore un SET_SSV.

Si le client veut utiliser des délégations, des dispositions, des notifications de répertoire, ou tout autre état qui exige un canal de retour, alors il doit ajouter une connexion au canal de retour si CREATE_SESSION ne l'a pas déjà fait. Le client crée une connexion, et invoque BIND_CONN_TO_SESSION pour associer la connexion à la session et au canal de retour de la session. Si CREATE_SESSION ne l'a pas déjà fait, le client DOIT dire au serveur quelle sécurité est exigée afin que le client accepte les rappels. Le client fait cela via BACKCHANNEL_CTL. Si le client a choisi la protection SP4_MACH_CRED ou SP4_SSV quand il a invoqué EXCHANGE_ID, alors le client DEVRAIT spécifier que le canal de retour utilise les contextes RPCSEC_GSS pour la sécurité.

Si le client veut utiliser des connexions supplémentaires pour le canal de retour, alors il doit invoquer BIND_CONN_TO_SESSION sur chaque connexion qu'il veut utiliser avec la session. Si le client veut utiliser des connexions supplémentaires pour le canal avant, alors il a besoin d'invoquer BIND_CONN_TO_SESSION si il a spécifié la protection d'état SP4_SSV ou SP4_MACH_CRED quand l'identifiant de client a été créé.

À ce point, la session a atteint l'état permanent.

2.10.12 Temporisateur d'inactivité de session

Le serveur PEUT tenir un temporisateur d'inactivité de session pour chaque session. Si le temporisateur d'inactivité de session expire, alors le serveur PEUT détruire la session. Pour éviter de perdre une session à cause de son inactivité, le client DOIT renouveler le temporisateur d'inactivité de session. La longueur du temporisateur d'inactivité de session NE DOIT PAS être moins que l'attribut lease_time (paragraphe 5.8.1.11). Comme avec le renouvellement de prêt (paragraphe 8.3) quand le serveur reçoit une opération SEQUENCE, il réinitialise le temporisateur d'inactivité de session, et NE DOIT PAS permettre que le temporisateur expire pendant que le reste des opérations de la demande de la procédure COMPOUND est en cours d'exécution. Une fois la dernière opération terminée, le serveur DOIT régler le temporisateur d'inactivité de session à expirer pas plus tôt que la somme de l'heure courante et la valeur de l'attribut lease_time.

2.10.13 Mécanique de session - récupération

2.10.13.1 Événements qui exigent une action du client

Les événements suivants exigent une action du client pour la récupération.

2.10.13.1.1 Perte du contexte RPCSEC_GSS par le chemin de rappel

Si toutes les brides RPCSEC_GSS accordées par le client au serveur pour l'utilisation de rappels ont expiré, le client DOIT établir une nouvelle bride via BACKCHANNEL_CTL. Le champ sr_status_fanions du résultat du SEQUENCE indique quand les brides de rappel sont presque expirées, ou complètement expirées (voir le paragraphe 18.46.3).

2.10.13.1.2 Perte de connexion

Si le client perd la dernière connexion de la session et veut conserver la session, alors il doit créer une nouvelle connexion, et si, quand l'identifiant de client a été créé, BIND_CONN_TO_SESSION était spécifié dans la liste spo_must_enforce, le client DOIT utiliser BIND_CONN_TO_SESSION pour associer la connexion à la session.

Si il y avait une demande en instance au moment de la perte de connexion, alors si le client veut continuer d'utiliser la session, il DOIT réessayer la demande, comme décrit au paragraphe 2.10.6.2. Noter qu'il n'est pas nécessaire de réessayer les demandes sur une connexion avec la même adresse réseau de source ou la même adresse réseau de destination que la connexion perdue. Tant que l'identifiant de session, l'identifiant de créneau, et l'identifiant de séquence dans le ré-essai correspondent à ceux de la demande d'origine, le serveur va reconnaître la demande comme un ré-essai si il a exécuté la demande avant de déconnecter.

Si la connexion qui a été perdue était la dernière associée au canal de retour, et si le client veut conserver le canal de retour et/ou empêcher la révocation d'état rappelable, le client doit se reconnecter, et si il le fait, il DOIT associer la connexion à la session et au canal de retour via BIND_CONN_TO_SESSION. Le serveur DEVRAIT indiquer quand il n'a pas de connexion de rappel via le résultat sr_status_flags provenant de SEQUENCE.

2.10.13.1.3 Perte de contexte sur le canal de retour GSS

Via le résultat sr_status_flags de l'opération SEQUENCE ou d'autres moyens, le client va apprendre si certains ou tous les contextes RPCSEC_GSS qu'il a alloué au canal de retour ont été perdus. Si le client veut conserver le canal de retour et/ou ne pas mettre d'état rappelable sujet à révocation, le client doit utiliser BACKCHANNEL_CTL pour allouer de nouveaux contextes.

2.10.13.1.4 Perte de session

Le répondeur pourrait perdre un enregistrement de la session. Les causes incluent :

* Défaillance et redémarrage du répondeur.

* Une catastrophe qui cause la corruption ou la perte de la mémoire tampon de réponse sur le support de mémorisation. Cela s'applique même si le répondeur a indiqué dans le résultat de CREATE_SESSION qu'elle persisterait dans la mémoire tampon.

* Le serveur purge la session d'un client qui a été inactif pendant une très longue période.

* Par suite d'un changement de configuration parmi un ensemble de serveurs en grappe, d'une adresse réseau précédemment connectée à un serveur qui devient connectée à un serveur différent sans connaissance de la session en question. Un tel changement de configuration va généralement seulement se produire quand le serveur original cesse de fonctionner pour une certaine durée.

La perte de mémoire tampon de réponse est équivalente à la perte de session. Le répondeur indique la perte de session au demandeur en retournant NFS4ERR_BADSESSION sur la prochaine opération qui utilise l'identifiant de session qui se réfère à la session perdue.

Après un événement comme un redémarrage de serveur, le client peut avoir perdu ses connexions. Le client suppose pour le moment que la session n'a pas été perdue. Il se reconnecte, et si il a spécifié l'application d'association de connexion quand la session a été créée, il invoque BIND_CONN_TO_SESSION en utilisant l'identifiant de session. Autrement, il invoque SEQUENCE. Si BIND_CONN_TO_SESSION ou SEQUENCE retourne NFS4ERR_BADSESSION, le client sait que la session n'est pas disponible pour lui quand il communique avec cette adresse réseau. Si la connexion survit à la perte de session, alors la prochaine opération SEQUENCE que le client envoie sur la connexion va renvoyer NFS4ERR_BADSESSION. Le client sait là encore que la session est perdue.

Voici un algorithme suggéré pour le client quand il reçoit NFS4ERR_BADSESSION. Il n'est pas obligatoire en ce que, si un client ne veut pas tirer parti de caractéristiques comme le multiplexage, il peut en omettre des parties. Cependant, c'est un exemple utile qui attire l'attention sur divers problèmes de récupération possibles :

1. Si le client a d'autres connexions à d'autres adresses réseau de serveur associées à la même session, tenter un COMPOUND avec une seule opération, SEQUENCE, sur chacune des autres connexions.

2. Si la tentative réussit, la session est toujours vivante, et c'est un fort indicateur que l'adresse réseau du serveur a bougé. Le client pourrait envoyer un EXCHANGE_ID sur la connexion qui a retourné NFS4ERR_BADSESSION pour voir si il y a des opportunités pour un multiplexage d'identifiant de client (c'est-à-dire, le même identifiant de client et la valeur so_major_id sont retournés). Le client pourrait utiliser le DNS pour voir si l'adresse réseau déplacée a été remplacée par une autre, afin que les avantages de performances et de disponibilité du multiplexage de session puissent continuer.

3. Si les demandes SEQUENCE échouent avec NFS4ERR_BADSESSION, alors la session n'existe plus sur aucune des adresses réseau du serveur pour lesquelles le client avait des connexions associées à l'identifiant de session. Il est possible que la session soit encore vivante et disponible sur d'autres adresses réseau. Le client envoie un EXCHANGE_ID sur toutes les connexions pour voir si le propriétaire de serveur écoute encore sur ces adresses réseau. Si le même propriétaire de serveur est retourné mais qu'un nouvel identifiant de client est retourné, c'est une forte indication d'un redémarrage du serveur. Si à la fois le même propriétaire de serveur et le même identifiant de client sont retournés, alors c'est une forte indication que le serveur a supprimé la session, et le client devra envoyer un CREATE_SESSION si il n'a pas d'autres sessions pour cet identifiant de client. Si un propriétaire de serveur différent est retourné, le client peut utiliser le DNS pour trouver d'autres adresses réseau. Si il ne le fait pas, ou si le DNS ne trouve pas d'autre adresse pour le serveur, alors le client va être incapable de fournir le service NFSv4.1, et une erreur fatale devrait être retournée aux processus qui utilisaient le serveur. Si le client utilise un paradigme de "monte", le démontage du serveur est conseillé.

4. Si le client sait qu'aucune autre connexion associée à l'identifiant de session et aux adresses réseau du serveur ne sont, ou ont été, associées à l'identifiant de session, alors le client peut utiliser le DNS pour trouver d'autres adresses réseau. Si il ne le fait pas, ou si le DNS ne trouve pas d'autre adresse pour le serveur, alors le client va être incapable de fournir le service NFSv4.1, et une erreur fatale devrait être retournée aux processus qui utilisaient le serveur. Si le client utilise un paradigme de "monte", le démontage du serveur est conseillé.

Si il y a un événement de reconfiguration qui résulte en l'allocation de la même adresse réseau aux serveurs où la valeur de eir_server_scope est différente, il ne peut pas être garanti qu'un identifiant de session généré par le premier va être reconnu comme invalide par le second. Donc, en gérant les reconfigurations de serveurs parmi des serveurs avec des valeurs différentes de portée de serveur, il est nécessaire de s'assurer que tous les clients sont déconnectés du premier serveur avant d'effectuer la reconfiguration. Néanmoins, les clients ne devraient pas supposer que les serveurs vont toujours respecter cette exigence ; les clients DOIVENT être prêts à traiter des effets inattendus de reconfigurations de serveurs. Même lorsque un identifiant de session est à tort reconnu comme valide, il est probable que la connexion ne va pas être reconnue comme valide ou qu'une valeur de séquence pour un créneau ne va pas être correcte. Donc, quand un client reçoit des résultats qui indiquent de telles erreurs inattendues, l'utilisation de EXCHANGE_ID pour déterminer la configuration de serveur actuelle est RECOMMANDÉE.

Une variante du cas ci-dessus est que après que l'adresse réseau d'un serveur bouge, il n'y a pas de serveur NFSv4.1 qui écoute, par exemple, pas d'écoutant sur l'accès 2049. Dans cet exemple, il se produit une des choses suivantes : le serveur NFSv4 retourne NFS4ERR_MINOR_VERS_MISMATCH, le serveur NFS retourne une erreur PROG_MISMATCH, l'écoutant RPC sur l'accès 2049 retourne PROG_UNVAIL, ou tente de se reconnecter à la fin de la temporisation de l'adresse réseau. Cela DEVRAIT être traité comme équivalent à un SEQUENCE qui retourne NFS4ERR_BADSESSION pour cet objet.

Quand le client détecte une perte de session, il doit invoquer CREATE_SESSION pour récupérer. Toutes les opérations non idempotentes qui étaient en cours pourraient avoir été effectuées sur le serveur au moment de la perte de session. Le client n'a pas de moyen général de récupérer de cela.

Noter que la perte de session n'implique pas la perte de verrou de gamme d'octets, d'ouverture, de délégation, ou d'état de disposition parce que les verrous, ouvertures, délégations, et dispositions sont liées à l'identifiant de client et dépendent de l'identifiant de client, pas de la session. Pas plus que la perte de verrou de gamme d'octets, d'ouverture, délégation, ou état de disposition n'implique la perte de l'état de session, parce que la session dépend de l'identifiant de client ; la perte de l'identifiant de client n'implique cependant pas la perte de session, de verrou de gamme d'octets, d'ouverture, de délégation, et d'état de disposition. Voir au paragraphe 8.4.2. Une session peut survivre à un redémarrage de serveur, mais la récupération de verrous peut quand même être nécessaire.

Il est possible que CREATE_SESSION échoue avec NFS4ERR_STALE_CLIENTID (par exemple, le serveur redémarre et ne préserve pas l'état d'identifiant de client). Si il en est ainsi, le client doit invoquer EXCHANGE_ID, suivi par CREATE_SESSION.

2.10.13.2 Événements exigeant une action du serveur

Les événements suivants exigent une action du serveur pour récupérer.

2.10.13.2.1 Panne et redémarrage du client

Comme décrit au paragraphe 18.35, un client qui a redémarré envoie EXCHANGE_ID de telle façon qu'il cause la suppression par le serveur de toutes les sessions qu'il avait.

2.10.13.2.2 Panne du client sans redémarrage

Si un client tombe en panne et ne récupère jamais, il ne va jamais envoyer un EXCHANGE_ID avec son vieux propriétaire de client. Donc, le serveur a un état de session qui ne va jamais être encore utilisé. Après une longue période, et si le serveur a des contraintes de ressources, il PEUT détruire la vieille session ainsi que l'état de verrouillage.

2.10.13.2.3 Partition de réseau étendue

Pour le serveur, la partition étendue de réseau peut n'être pas différente d'une panne de client sans redémarrage (voir le paragraphe 2.10.13.2.2). Sauf si le serveur peut discerner que il y a une partition de réseau, il est libre de traiter la situation comme si le client avait une panne permanente.

2.10.13.2.4 Perte de connexion du canal de retour

Si il y avait des demandes de rappel en instance au moment d'une perte de connexion, alors le serveur DOIT réessayer les demandes, comme décrit au paragraphe 2.10.6.2. Noter qu'il n'est pas nécessaire de réessayer les demandes sur une connexion avec la même adresse réseau de source ou la même adresse réseau de destination que la connexion perdue. Tant que l'identifiant de session, l'identifiant de créneau, et l'identifiant de séquence dans le ré-essai correspondent à ceux de la demande d'origine, la cible du rappel va reconnaître la demande comme ré-essai même si il a vu la demande avant de déconnecter.

Si la perte de connexion est la dernière associée au canal de retour, alors le serveur DOIT l'indiquer dans le champ sr_status_flags de chaque réponse à SEQUENCE jusqu'à ce que le canal de retour soit rétabli. Il y a deux situations, chacune utilisant des fanions d'état différents : non connexité pour le canal de retour de la session et non connexité pour le canal de retour de toute session du client. Voir au paragraphe 18.46 la description des fanions appropriés dans sr_status_flags.

2.10.13.2.5 Perte de contexte GSS

Le serveur DEVRAIT surveiller quand le nombre de brides RPCSEC_GSS allouées au canal de retour atteint un, et quand cette seule bride est proche de l'expiration (c'est-à-dire, entre une et deux périodes de temps de prêt) et l'indiquer dans le champ sr_status_flags de toutes les réponses à SEQUENCE. Le serveur DOIT indiquer quand toutes les brides RPCSEC_GSS allouées au canal de retour ont expiré via le champ sr_status_flags de toutes les réponses à SEQUENCE.

2.10.14 NFS parallèle et sessions

Un client et un serveur peuvent être une mise en œuvre non pNFS, une mise en œuvre de serveur de métadonnées, une mise en œuvre de serveur de données, ou deux ou trois types de mises en œuvre. Les fanions EXCHGID4_FLAG_USE_NON_PNFS, EXCHGID4_FLAG_USE_PNFS_MDS, et EXCHGID4_FLAG_USE_PNFS_DS (non mutuellement exclusifs) sont passés dans les arguments EXCHANGE_ID et permettent au client d'indiquer comment il veut utiliser les sessions créées sous l'identifiant de client, et permettent au serveur d'indiquer comment il va permettre aux sessions d'être utilisées. Voir au paragraphe 13.1 les considérations sur les sessions de pNFS.

3. Constantes et types de données du protocole

La syntaxe et la sémantique pour décrire les types de données du protocole NFSv4.1 sont définies dans les documents XDR [RFC4506] et RPC [RFC5531]. Les paragraphes qui suivent s'appuient sur les types de données XDR pour définir les constantes, types, et structures spécifiques de ce protocole. La liste complète des types de données XDR est dans la [RFC5662].

3.1 Constantes de base

```
const NFS4_FHSIZE = 128 ;
const NFS4_VERIFIER_SIZE = 8 ;
const NFS4_OPAQUE_LIMIT = 1024 ;
const NFS4_SESSIONID_SIZE = 16 ;

const NFS4_INT64_MAX = 0x7fffffffffffffff ;
const NFS4_UINT64_MAX = 0xffffffffffffffff ;
const NFS4_INT32_MAX = 0x7fffffff ;
const NFS4_UINT32_MAX = 0xffffffff ;

const NFS4_MAXFILELEN = 0xffffffffffffffff ;
const NFS4_MAXFILEOFF = 0xfffffffffffffffe ;
```

Sauf notation contraire, toutes ces constantes sont définies en octets.

* NFS4_FHSIZE est la taille maximum d'une bride de fichier.
* NFS4_VERIFIER_SIZE est la taille fixée d'un vérificateur.
* NFS4_OPAQUE_LIMIT est la taille maximum de certaines informations opaques.
* NFS4_SESSIONID_SIZE est la taille fixée d'un identifiant de session.
* NFS4_INT64_MAX est la valeur maximum d'un entier signé de 64 bits.

* NFS4_UINT64_MAX est la valeur maximum d'un entier non signé de 64 bits.
* NFS4_INT32_MAX est la valeur maximum d'un entier signé de 32 bits.
* NFS4_UINT32_MAX est la valeur maximum d'un entier non signé de 32 bits.
* NFS4_MAXFILELEN est la longueur maximum d'un fichier régulier.
* NFS4_MAXFILEOFF est le décalage maximum dans un fichier régulier.

3.2 Types de données de base

Voici les types de données de base NFSv4.1.

Type de données	Définition
int32_t	typedef int int32_t
uint32_t	typedef unsigned int uint32_t
int64_t	typedef hyper int64_t
uint64_t	typedef unsigned hyper uint64_t
attrlist4	typedef opaque attrlist4<> ; Utilisé pour les attributs de fichier/répertoire.
bitmap4	typedef uint32_t bitmap4<> ; Utilisé dans le codage de dispositif d'attribut.
changeid4	typedef uint64_t changeid4 ; Utilisé dans la définition de change_info4.
clientid4	typedef uint64_t clientid4 ; Référence abrégée à l'identification de client.
count4	typedef uint32_t count4 ; Divers paramètres de compte (READ, WRITE, COMMIT).
length4	typedef uint64_t length4 ; Logneur d'une gamme d'octets dans un fichier.
mode4	typedef uint32_t mode4 ; Type de données d'attribut de mode.
nfs_cookie4	typedef uint64_t nfs_cookie4 ; Valeur de mouchard opaque pour READDIR.
nfs_fh4	typedef opaque nfs_fh4<NFS4_FHSIZE> ; Définition de bride de fichier.
nfs_ftype4	enum nfs_ftype4 ; Divers types de fichier définis.
nfsstat4	enum nfsstat4 ; Retourne la valeur pour les opérations.
offset4	typedef uint64_t offset4 ; Diverses désignations de décalage (READ, WRITE, LOCK, COMMIT).
qop4	typedef uint32_t qop4 ; Désignation de la qualité de protection dans SECINFO.
sec_oid4	typedef opaque sec_oid4<> ; Identifiant d'objet de sécurité. Le type de données sec_oid4 n'est pas réellement opaque. À la place, il contient un IDENTIFIANT D'OBJET ASN.1 comme utilisé par GSS-API dans l'argument mech_type à GSS_Init_sec_context. Voir les détails dans la [RFC2743].
sequenceid4	typedef uint32_t sequenceid4 ; Numéro de séquence utilisé pour diverses opérations de session (EXCHANGE_ID, CREATE_SESSION, SEQUENCE, CB_SEQUENCE).
seqid4	typedef uint32_t seqid4 ; Identifiant de séquence utilisé pour le verrouillage.
sessionid4	typedef opaque sessionid4[NFS4_SESSIONID_SIZE] ; Identifiant de session.
slotid4	typedef uint32_t slotid4 ; Outil de séquençage pour divers opérations de session (SEQUENCE, CB_SEQUENCE).
utf8string	typedef opaque utf8string<> ; Codage UTF-8 pour les chaînes.
utf8str_cis	typedef utf8string utf8str_cis ; Chaîne UTF-8 insensible à la casse.
utf8str_cs	typedef utf8string utf8str_cs ; Chaîne UTF-8 sensible à la casse.
utf8str_mixed	typedef utf8string utf8str_mixed ; Chaînes UTF-8 avec préfixe sensible à la casse et suffixe insensible à la casse.
component4	typedef utf8str_cs component4 ; Représente les composants du nom de chemin.
inktext4	typedef utf8str_cs linktext4 ; Contenu de lien symbolique ("lien symbolique" est défini dans [IEEE-1003.1]).
pathname4	typedef component4 pathname4<> ; Représente le nom de chemin pour fs_locations.
verifier4	typedef opaque verifier4[NFS4_VERIFIER_SIZE] ; Vérificateur utilisé pour diverses opérations (COMMIT, CREATE, EXCHANGE_ID, OPEN, READDIR, WRITE). NFS4_VERIFIER_SIZE est défini comme 8.

Tableau 1

Fin des types de données de base

3.3 Types de données structurées

3.3.1 nfstime4

```
struct nfstime4 {
      int64_t       seconds;
      uint32_t      nseconds;
};
```

Le type de données nfstime4 donne le nombre de secondes et nanosecondes depuis minuit ou zéro heure le 1er janvier 1970 en temps universel coordonné (UTC). Les valeurs positives pour le champ des secondes notent des dates après le 1er janvier 1970 à zéro heure. Les valeurs négatives pour le champ des secondes notent des dates avant le 1er janvier 1970 à zéro heure. Dans les deux cas, le champ nseconds est à ajouter au champ des secondes pour la représentation de l'heure finale. Par exemple, si l'heure à représenter est une demie seconde avant le 1er janvier 1970 à zéro heure, le champ des secondes va avoir une valeur de un négatif (-1) et le champ nseconds va avoir une valeur de une demie seconde (500 000 000). Les valeurs supérieures à 999 999 999 pour le champ nseconds sont invalides.

Ce type de données est utilisé pour passer les informations d'heure et de date. Un serveur convertit de et vers sa représentation locale du temps quand il traite les valeurs de temps, en préservant autant de précision que possible. Si la précision de horodatages mémorisés pour un objet de système de fichiers est moindre que celle définie, une perte de précision peut survenir. L'ajout d'un protocole de maintenance de l'heure est RECOMMANDÉ pour réduire la dérive temporelle entre client et serveur.

3.3.2 time_how4

```
enum time_how4 {
      SET_TO_SERVER_TIME4 = 0,
      SET_TO_CLIENT_TIME4 = 1
};
```

3.3.3 settime4

```
union settime4 switch (time_how4 set_it) {
  cas SET_TO_CLIENT_TIME4 :  nfstime4       time;
  défaut : vide ;
};
```

Les types de données time_how4 et settime4 sont utilisés pour régler les horodatages dans les attributs d'objet de fichier. Si set_it est SET_TO_SERVER_TIME4, alors le serveur utilise sa représentation locale du temps pour la valeur du temps.

3.3.4 specdata4

```
struct specdata4 {
  uint32_t specdata1                    ; /* numéro d'appareil majeur */
  uint32_t specdata2                    ; /* numéro d'appareil mineur */
};
```

Ce type de données représente les numéros d'appareil pour les types de fichier d'appareil NF4CHR et NF4BLK.

3.3.5 fsid4

```
struct fsid4 {
      uint64_t      major;
      uint64_t      minor;
};
```

3.3.6 change_policy4

```
struct change_policy4 {
      uint64_t       cp_major;
      uint64_t       cp_minor;
};
```

Le type de données change_policy4 est utilisé pour l'attribut RECOMMANDÉ change_policy. Il fournit une indication de séquençage de changement analogue à l'attribut de changement. Pour permettre au serveur de présenter une valeur valide à travers les réinitialisations de serveur sans exiger une mémorisation persistante, deux quantités de 64 bits sont utilisées, dont l'une est un identifiant d'instance de serveur et la seconde est incrémentée de manière non persistante, au sein d'une instance de serveur donnée.

3.3.7 fattr4

```
struct fattr4 {
    bitmap4        attrmask;
    attrlist4      attr_vals;
};
```

Le type de données fattr4 est utilisé pour représenter les attributs de fichier et répertoire.

Le gabarit binaire est un dispositif compté d'entiers de 32 bits utilisé pour contenir les valeurs des bits. La position de l'entier dans le dispositif qui contient n bits peut être calculée à partir de l'expression (n / 32), et son bit dans cet entier est (n mod 32).

```
                    0             1
+-----------+-----------+-----------+--
|  compte   | 31  ..  0 | 63  .. 32 |
+-----------+-----------+-----------+--
```

3.3.8 change_info4

```
struct change_info4 {
    bool          atomique ;
    changeid4     avant ;
    changeid4     après ;
};
```

Ce type de données est utilisé avec les opérations CREATE, LINK, OPEN, REMOVE, et RENAME pour faire savoir au client la valeur de l'attribut de changement pour le répertoire dans lequel réside l'objet système de fichiers cible.

3.3.9 netaddr4

```
struct netaddr4 {                          /* voir la structure rpcb dans la RFC 1833 */
    string na_r_netid<>;                   /* identifiant de réseau */
    string na_r_addr<>;                    /* adresse universelle */
};
```

Le type de données netaddr4 est utilisé pour identifier les points d'extrémité de transport de réseau. Les champs na_r_netid et na_r_addr contiennent respectivement un netid et une uaddr. Les concepts de netid et uaddr sont définis dans la [RFC5665]. Les formats de netid et uaddr pour TCP sur IPv4 et TCP sur IPv6 sont définis dans la [RFC5665], spécifiquement les Tableaux 2 et 3 et les paragraphes 5.2.3.3 et 5.2.3.4.

3.3.10 state_owner4

```
struct state_owner4 {
    clientid4      clientid;
    opaque         owner<NFS4_OPAQUE_LIMIT>;
```

```
};

typedef state_owner4 open_owner4;
typedef state_owner4 lock_owner4;
```

Le type de données state_owner4 est le type de base pour open_owner4 (paragraphe 3.3.10.1) et lock_owner4 (paragraphe 3.3.10.2).

3.3.10.1 open_owner4

Ce type de données est utilisé pour identifier le propriétaire de l'état OPEN.

3.3.10.2 lock_owner4

Cette structure est utilisée pour identifier le propriétaire de l'état de verrouillage de gamme d'octets.

3.3.11 open_to_lock_owner4

```
struct open_to_lock_owner4 {
    seqid4          open_seqid;
    stateid4        open_stateid;
    seqid4          lock_seqid;
    lock_owner4     lock_owner;
};
```

Ce type de données est utilisé pour la première opération LOCK faite pour un open_owner4. Il fournit à la fois le open_stateid et le lock_owner, de telle façon que la transition est faite à partir d'une séquence open_stateid valide à celle de la nouvelle séquence lock_stateid. Utiliser ce mécanisme évite la confirmation de la paire lock_owner/lock_seqid car il est lié à l'état établi sous la forme du open_stateid/open_seqid.

3.3.12 stateid4

```
struct stateid4 {
    uint32_t      seqid;
    opaque        autre[12];
};
```

Ce type de données est utilisé pour les divers mécanismes de partage d'états entre le client et le serveur. Le client ne modifie jamais une valeur de type de données identifiant d'état. La valeur de début du champ "seqid" est indéfinie. Il est exigé du serveur qu'il incrémente le champ "seqid" de un à chaque transition du identifiant d'état. Ceci est important car le client va inspecter le seqid dans les identifiants d'état OPEN pour déterminer l'ordre de traitement de OPEN fait par le serveur.

3.3.13 layouttype4

```
enum layouttype4 {
      LAYOUT4_NFSV4_1_FILES     = 0x1,
      LAYOUT4_OSD2_OBJECTS      = 0x2,
      LAYOUT4_BLOCK_VOLUME  = 0x3
};
```

Ce type de données indique quel type de disposition est utilisé. Le serveur de fichiers annonce les types de disposition qu'il prend en charge à travers l'attribut de système de fichiers fs_layout_type (paragraphe 5.12.1). Un client demande les dispositions d'un type particulier dans LAYOUTGET, et traite ces dispositions dans sa logique spécifique de type de disposition.

Le type de données layouttype4 est long de 32 bits. La gamme représentée par le type de disposition est partagée en trois parties. Le type 0x0 est réservé. Les types dans la gamme 0x00000001-0x7FFFFFFF sont globalement uniques et sont

alloués en accord avec la description du paragraphe 22.5 ; ils sont tenus par l'IANA. Les types dans la gamme 0x80000000-0xFFFFFFFF sont spécifiques du site et seulement pour utilisation privée.

L'énumération LAYOUT4_NFSV4_1_FILES spécifie que le type de disposition de fichier NFSv4.1, tel que défini à la Section 13, est à utiliser. L'énumération LAYOUT4_OSD2_OBJECTS spécifie que la disposition d'objet, telle que définie dans la [RFC5664], est à utiliser. De même, l'énumération LAYOUT4_BLOCK_VOLUME spécifie que la disposition de bloc/volume, telle que définie dans la [RFC5663], est à utiliser.

3.3.14 deviceid4

```
const NFS4_DEVICEID4_SIZE = 16 ;

typedef opaque  deviceid4[NFS4_DEVICEID4_SIZE] ;
```

Les informations de disposition incluent des identifiants d'appareil qui spécifient un appareil de mémorisation à travers une bride compacte. Les informations d'adressage et de type sont obtenues avec l'opération GETDEVICEINFO. Il n'est pas garanti que les identifiants d'appareil soient valides à travers les redémarrages de serveur de métadonnées. Un identifiant d'appareil est unique par identifiant de client et type de disposition. Voir au paragraphe 12.2.10 plus de détails.

3.3.15 device_addr4

```
struct device_addr4 {
    layouttype4         da_layout_type;
    opaque              da_addr_body<>;
};
```

L'adresse d'appareil est utilisée pour établir un canal de communication avec l'appareil de mémorisation. Différents types de disposition vont exiger des types de données différents pour définir comment ils communiquent avec les appareils de mémorisation. Le champ opaque da_addr_body est interprété sur la base du champ da_layout_type spécifié.

Le présent document définit l'adresse d'appareil pour la disposition de fichier NFSv4.1 (voir le paragraphe 13.3) qui identifie un appareil de mémorisation par adresse réseau IP et numéro d'accès. C'est suffisant pour que les clients communiquent avec les appareils de mémorisation NFSv4.1, et peut être suffisant aussi pour d'autres types de dispositions. Les types d'appareil pour les appareils de mémorisation fondés sur l'objet et les appareils de mémorisation de bloc (par exemple, les étiquettes de volume d'interface de système de petit ordinateur (SCSI, *Small Computer System Interface*)) sont définis par leurs spécifications de disposition respectives.

3.3.16 layout_content4

```
struct layout_content4 {
      layouttype4  loc_type;
      opaque       loc_body<>;
};
```

Le champ loc_body est interprété sur la base du type de disposition (loc_type). Le présent document définit le loc_body pour le type de disposition de fichier NFSv4.1 ; voir le paragraphe 13.3 pour sa définition.

3.3.17 layout4

```
struct layout4 {
    offset4             lo_offset;
    length4             lo_length;
    layoutiomode4       lo_iomode;
    layout_content4     lo_content;
};
```

Le type de données layout4 définit une disposition pour un fichier. Les données spécifiques du type de disposition sont opaques dans lo_content. Comme les dispositions sont subdivisibles, le décalage et la longueur avec la bride de fichier du

fichier, l'identifiant de client, le iomode, et le type de disposition identifient la disposition.

3.3.18 layoutupdate4

```
struct layoutupdate4 {
    layouttype4         lou_type;
    opaque              lou_body<>;
};
```

Le type de données layoutupdate4 est utilisé par le client pour retourner des informations de disposition mises à jour au serveur de métadonnées via l'opération LAYOUTCOMMIT (paragraphe 18.42). Ce type de données fournit un canal pour repasser les informations spécifiques du type de disposition (dans le champ lou_body) au serveur de métadonnées. Par exemple, pour le type de disposition bloc/volume, cela pourrait inclure la liste des blocs réservés qui ont été écrits. Le contenu de l'argument opaque lou_body est déterminé par le type de disposition. La disposition NFSv4.1 fondée sur le fichier n'utilise pas ce type de données; si lou_type est LAYOUT4_NFSV4_1_FILES, le champ lou_body DOIT avoir une longueur de zéro.

3.3.19 layouthint4

```
struct layouthint4 {
  layouttype4         loh_type;
  opaque              loh_body<>;
};
```

Le type de données layouthint4 est utilisé par le client pour passer une indication sur le type de disposition qu'il voudrait créer pour un fichier particulier. C'est le type de données spécifié par l'attribut layout_hint décrit au paragraphe 5.12.4. Le serveur de métadonnées peut ignorer l'indication ou peut ignorer sélectivement certains champs dans l'indication. Cette indication devrait être fournie au moment de la création au titre des attributs initiaux dans OPEN. Le champ loh_body est spécifique du type de disposition (loh_type). La disposition NFSv4.1 fondée sur le fichier utilise le type de données nfsv4_1_file_layouthint4 comme défini au paragraphe 13.3.

3.3.20 layoutiomode4

```
enum layoutiomode4 {
      LAYOUTIOMODE4_READ  = 1,
      LAYOUTIOMODE4_RW    = 2,
      LAYOUTIOMODE4_ANY   = 3.
};
```

Le iomode spécifie si le client a l'intention de juste lire ou de lire et écrire les données représentées par la disposition. Bien que le iomode LAYOUTIOMODE4_ANY NE DOIVE PAS être utilisé dans les arguments à l'opération LAYOUTGET, il PEUT être utilisé dans les arguments aux opérations LAYOUTRETURN et CB_LAYOUTRECALL. Le iomode YOUTIOMODE4_ANY spécifie que les dispositions relevant des deux iomodes LAYOUTIOMODE4_READ et LAYOUTIOMODE4_RW sont retournées ou rappelées, respectivement. L'utilisation par le serveur de métadonnées du iomode peut dépendre du type de disposition utilisé. Les appareils de mémorisation PEUVENT valider les accès d'entrée/sortie par rapport au iomode et rejeter les accès invalides.

3.3.21 nfs_impl_id4

```
struct nfs_impl_id4 {
    utf8str_cis   nii_domain;
    utf8str_cs    nii_name;
    nfstime4      nii_date;
};
```

Ce type de données est utilisé pour identifier les détails de mise en œuvre du client et du serveur. Le champ nii_domain est le nom de domaine du DNS avec lequel la mise en œuvre est associée. Le champ nii_name est le nom de produit de la mise en œuvre et est de forme complètement libre. Il est RECOMMANDÉ que le nii_name soit utilisé pour distinguer

l'architecture de machine, les plates-formes de machine, les révisions, les versions, et les niveaux de réparations. Le champ nii_date est l'horodatage de quand l'instance de logiciel a été publiée ou construite.

3.3.22 threshold_item4

```
struct threshold_item4 {
    layouttype4    thi_layout_type;
    bitmap4        thi_hintset;
    opaque         thi_hintlist<>;
};
```

Ce type de données contient une liste d'indications spécifiques d'un type de disposition pour aider le client à déterminer quand il devrait envoyer les entrées/sorties directement à travers le serveur de métadonnées plutôt que les appareils de mémorisation. Le type de données consiste en le type de la disposition (thi_layout_type) un gabarit binaire (thi_hintset) décrivant l'ensemble d'undications prises en charge par le serveur (elles peuvent différer selon le type de disposition) et une liste d'indications (thi_hintlist) dont le contenu est déterminé par le gabarit binaire du hintset. Voir l'attribut mdsthreshold pour les détails.

Le champ thi_hintset est un gabarit binaire des valeurs suivantes :

Nom	N°	Type de données	Description
threshold4_read_size	0	length4	Si la longueur d'un fichier est inférieure à la valeur de threshold4_read_size, alors il est RECOMMANDÉ que le client lise le fichier via le MDS et non un appareil de mémorisation.
threshold4_write_size	1	length4	Si la longueur d'un fichier est inférieure à la valeur de threshold4_write_size, alors il est RECOMMANDÉ que le client écrive sur le fichier via le MDS et non un appareil de mémorisation.
threshold4_read_iosize	2	length4	Pour les tailles d'entrées/sortie de lecture en dessous de ce seuil, il est RECOMMANDÉ de lire les données par le MDS.
threshold4_write_iosize	3	length4	Pour les tailles d'entrées/sortie d'écriture en dessous de ce seuil, il est RECOMMANDÉ d'écrire les données à travers le MDS.

Tableau 2

3.3.23 mdsthreshold4

```
struct mdsthreshold4 {
    threshold_item4 mth_hints<>;
};
```

Ce type de données contient un dispositif d'éléments de type de données threshold_item4, dont chacun est valide pour un type de disposition particulier. Un dispositif est nécessaire parce que un serveur peut prendre en charge plusieurs types de dispositions pour un seul fichier.

4. Brides de fichier

La bride de fichier dans le protocole NFS est un identifiant unique par serveur pour un objet de système de fichiers. Le contenu de la bride de fichier est opaque au client. Donc, le serveur est responsable de traduire la bride de fichier en une représentation interne de l'objet de système de fichiers.

4.1 Obtention de la première bride de fichier

Les opérations du protocole NFS sont définies en termes de une ou plusieurs brides de fichier. Donc, le client a besoin d'une bride de fichier pour initier la communication avec le serveur. Avec le protocole NFSv3 [RFC1813], il existe un protocole auxiliaire pour obtenir cette première bride de fichier. Le protocole MOUNT , numéro de programme RPC 100005, fournit le mécanisme de traduction d'un nom de chemin de système de fichiers fondé sur une chaîne en une bride

de fichier, qui peut alors être utilisée par les protocoles NFS.

Le protocole MOUNT a des déficiences dans le domaine de la sécurité et dans son utilisation via des pare-feu. C'est une des raisons pour laquelle l'utilisation de la bride de fichier publique a été introduite dans les [RFC2054] et [RFC2055]. Avec l'utilisation de la bride de fichier publique combinée avec l'opération LOOKUP dans le protocole NFSv3, il a été démontré que le protocole MOUNT est inutile pour une interaction viable entre client et serveur NFS.

Donc, le protocole NFSv4.1 ne va pas utiliser de protocole auxiliaire pour la traduction des noms de chemin fondés sur une chaîne en une bride de fichier. Deux brides de fichier spéciales vont être utilisées comme points de départ pour le client NFS.

4.1.1 Bride de fichier racine

La première de ces brides de fichier spéciales est la bride de fichier ROOT *(racine)*. La bride de fichier ROOT est la racine "conceptuelle" de l'espace de noms de système de fichiers chez le serveur NFS. Le client utilise ou commence avec la bride de fichier ROOT en employant l'opération PUTROOTFH. L'opération PUTROOTFH donne pour instruction au serveur de régler la bride de fichier "courante" à la racine de l'arborescence de fichiers du serveur. Une fois cette opération PUTROOTFH utilisée, le client peut alors traverser la totalité de l'arborescence de fichiers du serveur avec l'opération LOOKUP. Une discussion complète de l'espace de noms de serveur figure à la Section 7.

4.1.2 Bride de fichier publique

La seconde bride de fichier spéciale est la bride de fichier PUBLIC. À la différence de la bride de fichier ROOT, la bride de fichier PUBLIC peut être liée à ou représenter un objet de système de fichiers arbitraire chez le serveur. Le serveur est responsable de ce lien. Il se peut que la bride de fichier PUBLIC et la bride de fichier ROOT se réfèrent au même objet de système de fichiers. Cependant, il appartient au logiciel administratif chez le serveur et aux politiques de l'administrateur du serveur de définir le lien de la bride de fichier PUBLIC et de l'objet de système de fichiers du serveur. Le client ne peut faire aucune hypothèse sur ce lien. Le client utilise la bride de fichier PUBLIC via l'opération PUTPUBFH.

4.2 Types de bride de fichier

Dans le protocole NFSv3, il y avait un type de bride de fichier avec un seul ensemble de sémantique. Ce type de bride de fichier est appelé "persistant" dans NFSv4.1. La sémantique d'une bride de fichier persistante reste la même qu'avant. Un nouveau type de bride de fichier introduit dans NFSv4.1 est la bride de fichier "volatile", qui tente de s'accommoder de certains environnements de serveur.

Le type de bride de fichier volatile a été introduit pour traiter les questions de fonctions de serveur ou de mise en œuvre qui rendent infaisable une mise en œuvre correcte de bride de fichier persistante. Certains environnements de serveur ne fournissent pas un niveau invariant de système de fichier qui puisse être utilisé pour construire une bride de fichier persistante. Le système de fichiers du serveur sous-jacent ne peut pas fournir l'invariant ou les interfaces de programmation de système de fichiers du serveur ne peuvent pas fournir l'accès à l'invariant nécessaire. Les brides de fichier volatiles peuvent faciliter la fonction de mise en œuvre de serveur comme la gestion de mémorisation hiérarchique ou la réorganisation ou migration de système de fichiers. Cependant, la bride de fichier volatile augmente la charge de mise en œuvre pour le client.

Comme le client va devoir traiter différemment les brides de fichier persistantes et volatiles, un attribut de fichier est défini, qui peut être utilisé par le client pour déterminer les types de bride de fichier retournés par le serveur.

4.2.1 Propriétés générales d'une bride de fichier

La bride de fichier contient toutes les informations dont le serveur a besoin pour distinguer un fichier individuel. Pour le client, la bride de fichier est opaque. Le client mémorise les brides de fichier pour les utiliser dans les demandes ultérieures et peut comparer l'égalité de deux brides de fichier provenant du même serveur en faisant une comparaison octet par octet. Cependant, le client NE DOIT PAS interpréter autrement le contenu des brides de fichier. Si deux brides de fichier provenant du même serveur sont égales, elles DOIVENT se référer au même fichier. les serveurs DEVRAIENT essayer de maintenir une correspondance biunivoque entre les brides de fichier et les fichiers, mais cela n'est pas exigé. Les clients DOIVENT utiliser les comparaisons de brides de fichier seulement pour améliorer les performances, pas pour un comportement correct. Tous les clients doivent être prêts à des situations dans lesquelles il ne peut pas être déterminé si

deux brides de fichier notent le même objet et dans ce cas, éviter de faire des hypothèses invalides qui pourraient causer un comportement incorrect. Une discussion de la comparaison de bride de fichier et des attributs dans le contexte de la mise en antémémoire des données est présentée au paragraphe 10.3.4.

Par exemple, dans le cas où deux noms de chemin différents quand ils sont traversés au serveur se terminent au même objet de système de fichiers, le serveur DEVRAIT retourner la même bride de fichier pour chaque chemin. Cela peut se produire si une liaison physique (voir [IEEE-1003.1]) est utilisée pour créer deux noms de fichier qui se réfèrent au même objet de fichier sous-jacent et ses données associées. Par exemple, si les chemins /a/b/c et /a/d/c se réfèrent au même fichier, le serveur DEVRAIT retourner la même bride de fichier pour les deux traversées de nom de chemin.

4.2.2 Bride de fichier persistante

Une bride de fichier persistante est définie comme ayant une valeur fixe pour la durée de vie de l'objet de système de fichiers auquel elle se réfère. Une fois que le serveur a créé la bride de fichier pour un objet de système de fichiers, le serveur DOIT accepter la même bride de fichier pour l'objet pendant la durée de vie de l'objet. Si le serveur redémarre, le serveur NFS DOIT honorer la même valeur de bride de fichier que dans l'instanciation précédente du serveur. De même, si le système de fichiers a migré, le nouveau serveur NFS DOIT honorer la même bride de fichier que l'ancien serveur NFS.

La bride de fichier persistante va devenir périmée ou invalide quand l'objet de système de fichiers est supprimé. Quand une bride de fichier persistante est présentée au serveur et qu'elle se réfère à un objet supprimé, il DOIT retourner une erreur de NFS4ERR_STALE. Une bride de fichier peut devenir périmée quand le système de fichiers contenant l'objet n'est plus disponible. Le système de fichiers peut devenir indisponible si il existe sur un support qui peut être supprimé et que le support n'est plus disponible au serveur ou si le système de fichiers a été détruit en totalité ou si le système de fichiers a simplement été supprimé de l'espace de noms du serveur (c'est-à-dire, démonté dans un environnement UNIX).

4.2.3 Bride de fichier volatile

Une bride de fichier volatile ne partage pas les mêmes caractéristiques de longévité qu'une bride de fichier persistante. Le serveur peut déterminer qu'une bride de fichier volatile n'est plus valide à de nombreux moments. Si le serveur peut définitivement déterminer qu'une bride de fichier volatile se réfère à un objet qui a été supprimé, le serveur devrait retourner NFS4ERR_STALE au client (comme c'est le cas pour les brides de fichier persistantes). Dans tous les autres cas où le serveur détermine qu'une bride de fichier volatile ne peut plus être utilisée, il devrait retourner une erreur de NFS4ERR_FHEXPIRED.

L'attribut EXIGÉ "fh_expire_type" est utilisé par le client pour déterminer quel type de bride de fichier le serveur fournit pour un système de fichiers particulier. Cet attribut est un gabarit binaire avec les valeurs suivantes :

FH4_PERSISTENT : La valeur de FH4_PERSISTENT est utilisée pour indiquer une bride de fichier persistante, qui est valide jusqu'à ce que l'objet soit supprimé du système de fichiers. Le serveur ne va pas retourner NFS4ERR_FHEXPIRED pour cette bride de fichier. FH4_PERSISTENT est défini comme une valeur dans laquelle aucun des bits spécifiés ci-dessous n'est établi.

FH4_VOLATILE_ANY : la bride de fichier peut expirer à tout moment, sauf quand spécifiquement exclu (c'est-à-dire, FH4_NO_EXPIRE_WITH_OPEN).

FH4_NOEXPIRE_WITH_OPEN : peut seulement être établi quand FH4_VOLATILE_ANY est établi. Si ce bit est établi, alors la signification de FH4_VOLATILE_ANY est qualifiée pour exclure toute expiration de la bride de fichier quand elle est ouverte.

FH4_VOL_MIGRATION : la bride de fichier va expirer par suite d'une transition d'un système de fichiers (migration ou duplication) dans les cas où la continuité de l'utilisation de la bride de fichier n'est pas spécifiée par les informations de classe de bride dans l'attribut fs_locations_info. Quand ce bit est établi, les clients sans accès aux informations de fs_locations_info devraient supposer que les brides de fichier vont expirer lors des transitions de système de fichiers.

FH4_VOL_RENAME : la bride de fichier va expirer durant le changement de nom. Cela inclut un changement de nom par le client demandeur ou par tout autre client. Si FH4_VOL_ANY est établi, FH4_VOL_RENAME est redondant.

Les serveurs qui fournissent des brides de fichier volatiles qui peuvent expirer alors qu'elles sont ouvertes exigent un soin particulier à l'égard du traitement des RENAME et REMOVE. Cette situation peut arriver si FH4_VOL_MIGRATION ou

FH4_VOL_RENAME est établi, si FH4_VOLATILE_ANY est établi et FH4_NOEXPIRE_WITH_OPEN ne l'est pas, ou si un système de fichiers non en lecture seule a une cible de transition dans une classe de brides différentes. Dans ces cas, le serveur devrait refuser un RENAME ou REMOVE qui affecterait un fichier OPEN dans un des composants conduisant au fichier OPEN. De plus, le serveur devrait refuser toutes les demandes RENAME ou REMOVE durant la période de grâce, afin de s'assurer que les réclamations de fichiers où des brides de fichier peuvent avoir expiré ne font pas une réclamation pour le mauvais fichier.

Les brides de fichier volatiles conviennent particulièrement pour la mise en œuvre des pseudo systèmes de fichiers utilisés pour ponter des exportations. Voir au paragraphe 7.5 la discussion sur ce sujet.

4.3 Méthode de construction d'une bride de fichier volatile

Une bride de fichier volatile, bien que opaque pour le client, pourrait contenir :

[bit volatile = 1 | heure d'amorçage du serveur | créneau | numéro de génération]

* créneau est un indice dans le tableau de brides de fichier volatiles du serveur

* numéro de génération est le numéro de génération pour l'entrée/créneau du tableau

Quand le client présente une bride de fichier volatile, le serveur fait les vérifications suivantes, qui supposent que la vérification du bit volatile a réussi. Si l'heure d'amorçage du serveur est avant l'heure d'amorçage courante du serveur, on retourne NFS4ERR_FHEXPIRED. Si le créneau est hors gamme, on retourne NFS4ERR_BADHANDLE. Si le numéro de génération ne correspond pas, on retourne NFS4ERR_FHEXPIRED.

Quand le serveur redémarre, le tableau est parti (il est volatile).

Si le bit volatile est 0, alors c'est une bride de fichier persistante avec une structure différente à la suite.

4.4 Récupération du client de l'expiration de la bride de fichier

Si possible, le client DEVRAIT récupérer de la réception d'une erreur NFS4ERR_FHEXPIRED. Le client doit prendre des responsabilités supplémentaires pour se préparer à récupérer de l'expiration d'une bride de fichier volatile. Si le serveur retourne des brides de fichier persistantes, le client n'a pas besoin de ces étapes supplémentaires.

Pour les brides de fichier volatiles, le plus souvent, le client va avoir besoin de mémoriser les noms des composants conduisant et incluant l'objet de système de fichiers en question. Avec ces noms, le client devrait être capable de récupérer en trouvant une bride de fichier dans l'espace de noms qui est encore disponible ou en commençant par la racine de l'espacc de noms du système de fichiers du serveur.

Si la bride de fichier expirée se réfère à un objet qui a été supprimé du système de fichiers, évidemment, le client ne va pas être capable de récupérer la bride de fichier expirée.

Il est aussi possible que la bride de fichier expirée se réfère à un fichier qui a été renommé. Si le fichier a été renommé par un autre client, là encore il est possible que le client original ne soit pas capable de récupérer. Cependant, dans le cas où le client lui-même renomme le fichier et où le fichier est ouvert, il est possible que le client soit capable de récupérer. Le client peut déterminer le nouveau nom de chemin sur la base du traitement de la demande de changement de nom. Le client peut alors régénérer la nouvelle bride de fichier sur la base du nouveau nom de chemin. Le client pourrait aussi utiliser la procédure COMPOUND pour construire une série d'opérations comme :

```
RENAME A B
LOOKUP B
GETFH
```

Noter que la procédure COMPOUND ne fournit pas l'atomicité. Cet exemple réduit seulement les frais généraux de récupération d'une bride de fichier expirée.

5. Attributs de fichier

Pour satisfaire les exigences d'extensibilité et d'interopérabilité accrue avec les plates-formes non UNIX, les attributs doivent être traités de manière souple. La structure NFSv3 fattr3 contient une liste fixe d'attributs que tous les clients et serveurs ne sont pas capables de prendre en charge ou de se soucier. La structure fattr3 ne peut pas être étendue lorsque de nouveaux besoins apparaissent et elle ne fournit pas de moyen d'indiquer la non prise en charge. Avec le protocole NFSv4.1, le client est capable d'interroger le serveur sur les attributs qu'il prend en charge et construire des demandes avec seulement les attributs pris en charge (ou un sous ensemble de ceux-ci).

À cette fin, les attributs sont divisés en trois groupes : EXIGÉS, RECOMMANDÉS, et nommés. Les attributs EXIGÉS et RECOMMANDÉS sont pris en charge dans le protocole NFSv4.1 par un codage spécifique et bien défini et sont identifiés par un numéro. Ils sont demandés en rétablissant un bit dans le vecteur de bits envoyé dans la demande GETATTR ; le serveur répond en incluant un vecteur de bits dans la liste des attributs retournée dans la réponse. De nouveaux attributs EXIGÉS ou RECOMMANDÉS pourront être ajoutés au protocole NFSv4 au titre d'une nouvelle version mineure en publiant une RFC sur la voie de la normalisation qui alloue une valeur au nouveau numéro d'attribut et définit le codage de l'attribut. Voir au paragraphe 2.7 la discussion sur la question.

L'accès aux attributs nommés est fait par la nouvelle opération OPENATTR, qui accède à un répertoire caché d'attributs associés à un objet de système de fichiers. OPENATTR prend une bride de fichier pour l'objet et retourne la bride de fichier pour la hiérarchie d'attributs. La bride de fichier pour les attributs nommés est un objet de répertoire accessible par LOOKUP ou READDIR et contient les fichiers dont les noms représentent les attributs nommés et dont les octets de données sont la valeur de l'attribut. Par exemple :

```
LOOKUP      "foo"       ; cherche le fichier
GETATTR     attrbits
OPENATTR                ; accès aux attributs nommés de foo
LOOKUP      "x11icon"   ; cherche des attributs spécifiques
READ        0,4096      ; lit le flux d'octets
```

Tableau 3

Les attributs nommés sont destinés aux données nécessaires aux applications plutôt qu'à la mise en œuvre de client NFS. Les mises en œuvre de NFS sont vivement encouragées à définir leurs nouveaux attributs comme attributs RECOMMANDÉS en les apportant au processus de normalisation de l'IETF.

L'ensemble d'attributs classés EXIGÉS est délibérément petit car les serveurs ont besoin de faire ce qu'il faut pour les prendre en charge. Un serveur devrait prendre en charge autant d'attributs RECOMMANDÉS que possible mais, par leur définition, le serveur n'est pas obligé de les prendre tous en charge. Les attributs sont réputés EXIGÉS si les données sont à la fois nécessaires à un grand nombre de clients et ne sont pas autrement calculables par le client quand la prise en charge n'est pas fournie sur le serveur.

Noter que le répertoire caché retourné par OPENATTR est une facilité du traitement du protocole. Le client ne devrait pas faire d'hypothèses sur la mise en œuvre par le serveur des attributs nommés ni sur si le système de fichiers sous-jacent chez le serveur a un répertoire d'attributs nommés. Donc, les opérations comme SETATTR et GETATTR sur le répertoire d'attributs nommés sont indéfinies.

5.1 Attributs exigés

Ils DOIVENT être pris en charge par chaque client et serveur NFSv4.1 afin d'assurer un niveau minimum d'interopérabilité. Le serveur DOIT mémoriser et retourner ces attributs, et le client DOIT être capable de fonctionner avec un ensemble d'attributs limité à ces attributs. Avec juste les attributs EXIGÉS, certaine fonctions de client peuvent être entravées ou limitées d'une certaine façon. Un client peut demander qu'un de ces attributs soit retourné en établissant un bit dans la demande GETATTR, et le serveur DOIT retourner sa valeur.

5.2 Attributs recommandés

Ces attributs sont assez bien compris pour garantir la prise en charge dans le protocole NFSv4.1. Cependant, ils peuvent ne pas être pris en charge sur tous les clients et serveurs. Un client peut demander qu'un de ces attributs soit retourné en établissant un bit dans la demande GETATTR mais doit traiter le cas où le serveur ne le retourne pas. Un client PEUT

demander quel ensemble d'attributs le serveur prend en charge et NE DEVRAIT PAS demander les attributs que le serveur ne prend pas en charge. Un serveur devrait être tolérant aux demandes d'attributs non pris en charge et simplement ne pas les retourner plutôt que de considérer la demande comme une erreur. On s'attend à ce que les serveurs prennent en charge tous les attributs qu'ils peuvent prendre en charge sans difficulté et n'échouent seulement à prendre en charge que les attributs qui sont difficiles à prendre en charge dans leur environnement de fonctionnement. Un serveur devrait fournir les attributs chaque fois que il n'a pas à "mentir" au client. Par exemple, une heure de modification de fichier devrait être soit une heure précise, soit ne devrait pas être prise en charge par le serveur. Certaines fois cela va être difficile pour les clients, mais un client est mieux placé pour décider si et comment fabriquer ou construire un attribut plutôt que de le faire sans lui.

5.3 Attributs nommés

Ces attributs ne sont pas pris en charge par un codage direct dans le protocole NFSv4 mais sont accédés par des noms de chaîne plutôt que des numéros et correspondent à un flux non interprété d'octets mémorisés avec l'objet de système de fichiers. L'accès à cet espace de nom pour ces attributs peut être fait en utilisant l'opération OPENATTR. L'opération OPENATTR retourne une bride de fichier pour un "répertoire d'attributs nommés" virtuel, et plus de lecture et modification de l'espace de noms peuvent être faites en utilisant des opérations qui fonctionnent sur des répertoires plus typiques. En particulier, READDIR peut être utilisé pour obtenir une liste de ces attributs nommés, et LOOKUP et OPEN peuvent choisir un attribut particulier. La création d'un nouvel attribut nommé peut être le résultat d'un OPEN qui spécifie la création d'un fichier.

Une fois qu'un OPEN est fait, les attributs nommés peuvent être examinés et changés par des opérations READ et WRITE normales utilisant les brides de fichier et les identifiants d'état retournés par OPEN.

Les attributs nommés et le répertoire des attributs nommés peuvent avoir leurs propres attributs (non nommés). Chacun de ces objets DOIT avoir tous les attributs EXIGÉS et peut avoir des attributs RECOMMANDÉS supplémentaires. Cependant, l'ensemble des attributs pour les attributs nommés et le répertoire des attributs nommés ne doit pas être, et normalement ne va pas être, aussi grand que pour les autres objets dans ce système de fichiers.

Les attributs nommés et le répertoire des attributs nommés pourraient être la cible de délégations (dans le cas du répertoire des attributs nommés, ce vont être des délégations de répertoire). Cependant, comme accorder des délégations est à la discrétion du serveur, un serveur n'a pas besoin de prendre en charge de délégations sur les attributs nommés ou le répertoire des attributs nommés.

Il est RECOMMANDÉ que les serveurs prennent en charge des attributs nommés arbitraires. Un client ne devrait pas dépendre de la capacité de mémoriser des attributs nommés dans le système de fichiers du serveur. Si un serveur prend en charge les attributs nommés, un client qui est aussi capable de les traiter devrait être capable de copier les données et les métadonnées d'un fichier avec une complète transparence d'une localisation à une autre ; cela impliquerait que les noms permis pour les entrées régulières de répertoire sont valides aussi pour les noms d'attribut nommé.

Dans NFSv4.1, la structure des répertoires d'attributs nommés est restreinte d'un certain nombre de façons, afin d'empêcher le développement de mises en œuvre non interopérables dans lesquelles certains serveurs prennent en charge une structure générale de répertoire pleinement hiérarchique pour les attributs nommés tandis que d'autres prennent en charge une structure limitée mais adéquate pour les attributs nommés. Dans ces environnements, les clients ou applications pourraient en venir à dépendre d'extensions non portables. Les restrictions sont :

* CREATE n'est pas permis dans un répertoire des attributs nommés. Donc, des objets comme des liaisons symboliques et des fichiers spéciaux ne sont pas permis comme attributs nommés. De plus, des répertoires ne peuvent pas être créés dans un répertoire des attributs nommés, de sorte qu'une structure non hiérarchique des attributs nommés pour un seul objet est permise.

* Si OPENATTR est fait sur un répertoire des attributs nommés ou sur un attribut nommé, le serveur DOIT retourner NFS4ERR_WRONG_TYPE.

* Faire un RENAME d'un attribut nommé à un répertoire différent des attributs nommés ou à un répertoire ordinaire (c'est-à-dire, d'attribut non nommé) n'est pas permis.

* Créer des liaisons physiques entre répertoires d'attributs nommés ou entre répertoires d'attributs nommés et répertoires ordinaires n'est pas permis.

Les noms des attributs ne vont pas être contrôlés par le présent document ou d'autres documents sur la vois de la

normalisation de l'IETF. Voir la discussion du paragraphe 22.2.

5.4 Classification des attributs

Chaque attribut EXIGÉ et RECOMMANDÉ peut être classé dans une de trois catégories : par serveur (c'est-à-dire, la valeur de l'attribut va être la même pour tous les objets de fichier qui partagent le même propriétaire de serveur ; voir le paragraphe 2.5 pour la définition de propriétaire de serveur) par système de fichiers (c'est-à-dire, la valeur de l'attribut va être la même pour certains ou tous les objets de fichier qui partagent même attribut fsid (paragraphe 5.8.1.9) et propriétaire de serveur) ou par objet de système de fichiers. Noter qu'il est possible que certains attributs par système de fichiers puissent varier au sein du système de fichiers, selon la valeur de l'attribut "homogeneous" (paragraphe 5.8.2.16). Noter que les attributs time_access_set et time_modify_set ne sont pas mentionnés dans ce paragraphe parce que ils sont des attributs en écriture seule correspondant à time_access et time_modify, et sont utilisés dans une instance spéciale de SETATTR.

* L'attribut par serveur est lease_time.

* Les attributs par système de fichiers sont :
  support_attrs, suppattr_exclcreat, fh_expire_type, link_support, symlink_support, unique_handles, aclsupport, cansettime, case_insensitive, case_preserving, chown_restricted, files_avail, files_free, files_total, fs_locations, homogeneous, maxfilesize, maxname, maxread, maxwrite, no_trunc, space_avail, space_free, space_total, time_delta, change_policy, fs_status, fs_layout_type, fs_locations_info, fs_charset_cap.

* Les attributs par objet de système de fichiers sont :
  type, change, size, named_attr, fsid, rdattr_error, filehandle, acl, archive, fileid, hidden, maxlink, mimetype, mode, numlinks, owner, owner_group, rawdev, space_used, system, time_access, time_backup, time_create, time_metadata, time_modify, mounted_on_fileid, dir_notif_delay, dirent_notif_delay, dacl, sacl, layout_type, layout_hint, layout_blksize, layout_alignment, mdsthreshold, retention_get, retention_set, retentevt_get, retentevt_set, retention_hold, mode_set_masked.

Pour quota_avail_hard, quota_avail_soft, et quota_used, voir leur définition ci-dessous pour le classement approprié.

5.5 Attributs Set-Only et Get-Only

Certains attributs EXIGÉS et RECOMMANDÉS sont établis seulement ; c'est-à-dire, ils peuvent être établis via SETATTR mais non restitués via GETATTR. De même, certains attributs EXIGÉS et RECOMMANDÉS sont établis seulement ; c'est-à-dire, ils peuvent être restitués via GETATTR mais non établis via SETATTR. Si un client tente d'établir un get-only ou d'obtenir des attributs set-only, le serveur DOIT retourner NFS4ERR_INVAL.

5.6 Attributs exigés - liste et références de définition

La liste des attributs EXIGÉS apparaît au Tableau 4. La signification des colonnes du tableau est :
Nom : nom de l'attribut.
Id : numéro alloué à l'attribut. En cas de conflit entre le numéro alloué et la [RFC5662], cette dernière est probablement d'autorité, mais devrait être résolu avec un errata au présent document et/ou à la [RFC5662]. Voir [Errata] pour le traitement des errata.
Type de données : type de données XDR de l'attribut.
Acc : accès permis à l'attribut. R signifie lecture seule (GETATTR peut restituer, SETATTR ne peut pas établir). W signifie écriture seule (SETATTR peut établir, GETATTR ne peut pas restituer). R W signifie lecture/écriture (GETATTR peut restituer, SETATTR peut établir).
Défini dans : paragraphe de la présente spécification qui décrit l'attribut.

Nom	Id	Type de données	Acc	Défini dans
supported_attrs	0	bitmap4	R	paragraphe 5.8.1.1
type	1	nfs_ftype4	R	paragraphe 5.8.1.2
fh_expire_type	2	uint32_t	R	paragraphe 5.8.1.3
change	3	uint64_t	R	paragraphe 5.8.1.4
size	4	uint64_t	R W	paragraphe 5.8.1.5
link_support	5	bool	R	paragraphe 5.8.1.6
symlink_support	6	bool	R	paragraphe 5.8.1.7
named_attr	7	bool	R	paragraphe 5.8.1.8

fsid	8	fsid4	R	paragraphe 5.8.1.9
unique_handles	9	bool	R	paragraphe 5.8.1.10
lease_time	10	nfs_lease4	R	paragraphe 5.8.1.11
rdattr_error	11	enum	R	paragraphe 5.8.1.12
handle	19	nfs_fh4	R	paragraphe 5.8.1.13
suppattr_exclcreat	75	bitmap4	R	paragraphe 5.8.1.14

Tableau 4

5.7 Attributs recommandés - liste et références de définition

Les attributs RECOMMANDÉS sont définis au Tableau 5. La signification des en-têtes de colonnes est la même qu'au Tableau 4 ; voir le paragraphe 5.6 pour les significations.

Nom	Id	Type de données	Acc	Défini dans :
acl	12	nfsace4<>	R W	paragraphe 6.2.1
aclsupport	13	uint32_t	R	paragraphe 6.2.1.2
archive	14	bool	R W	paragraphe 5.8.2.1
cansettime	15	bool	R	paragraphe 5.8.2.2
case_insensitive	16	bool	R	paragraphe 5.8.2.3
case_preserving	17	bool	R	paragraphe 5.8.2.4
change_policy	60	chg_policy4	R	paragraphe 5.8.2.5
chown_restricted	18	bool	R	paragraphe 5.8.2.6
dacl	58	nfsacl41	R W	paragraphe 6.2.2
dir_notif_delay	56	nfstime4	R	paragraphe 5.11.1
dirent_notif_delay	57	nfstime4	R	paragraphe 5.11.2
fileid	20	uint64_t	R	paragraphe 5.8.2.7
files_avail	21	uint64_t	R	paragraphe 5.8.2.8
files_free	22	uint64_t	R	paragraphe 5.8.2.9
files_total	23	uint64_t	R	paragraphe 5.8.2.10
fs_charset_cap	76	uint32_t	R	paragraphe 5.8.2.11
fs_layout_type	62	layouttype4<>	R	paragraphe 5.12.1
fs_locations	24	fs_locations	R	paragraphe 5.8.2.12
fs_locations_info	67	fs_locations_info4	R	paragraphe 5.8.2.13
fs_status	61	fs4_status	R	paragraphe 5.8.2.14
hidden	25	bool	R W	paragraphe 5.8.2.15
homogeneous	26	bool	R	paragraphe 5.8.2.16
layout_alignment	66	uint32_t	R	paragraphe 5.12.2
layout_blksize	65	uint32_t	R	paragraphe 5.12.3
layout_hint	63	layouthint4	W	paragraphe 5.12.4
layout_type	64	layouttype4<>	R	paragraphe 5.12.5
maxfilesize	27	uint64_t	R	paragraphe 5.8.2.17
maxlink	28	uint32_t	R	paragraphe 5.8.2.18
maxname	29	uint32_t	R	paragraphe 5.8.2.19
maxread	30	uint64_t	R	paragraphe 5.8.2.20
maxwrite	31	uint64_t	R	paragraphe 5.8.2.21
mdsthreshold	68	mdsthreshold4	R	paragraphe 5.12.6
mimetype	32	utf8str_cs	R W	paragraphe 5.8.2.22
mode	33	mode4	R W	paragraphe 6.2.4
mode_set_masked	74	mode_masked4	W	paragraphe 6.2.5
mounted_on_fileid	55	uint64_t	R	paragraphe 5.8.2.23
non_trunc	34	bool	R	paragraphe 5.8.2.24
numlinks	35	uint32_t	R	paragraphe 5.8.2.25
owner	36	utf8str_mixed	R W	paragraphe 5.8.2.26
owner_group	37	utf8str_mixed	R W	paragraphe 5.8.2.27
quota_avail_hard	38	uint64_t	R	paragraphe 5.8.2.28
quota_avail_soft	39	uint64_t	R	paragraphe 5.8.2.29
quota_used	40	uint64_t	R	paragraphe 5.8.2.30
rawdev	41	specdata4	R	paragraphe 5.8.2.31
retentevt_get	71	retention_get4	R	paragraphe 5.13.3

retentevt_set	72	retention_set4	W	paragraphe 5.13.4
retention_get	69	retention_get4	R	paragraphe 5.13.1
retention_hold	73	uint64_t	R W	paragraphe 5.13.5
retention_set	70	retention_set4	W	paragraphe 5.13.2
sacl	59	nfsacl41	R W	paragraphe 6.2.3
space_avail	42	uint64_t	R	paragraphe 5.8.2.32
space_free	43	uint64_t	R	paragraphe 5.8.2.33
space_total	44	uint64_t	R	paragraphe 5.8.2.34
space_used	45	uint64_t	R	paragraphe 5.8.2.35
system	46	bool	R W	paragraphe 5.8.2.36
time_access	47	nfstime4	R	paragraphe 5.8.2.37
time_access_set	48	settime4	W	paragraphe 5.8.2.38
time_backup	49	nfstime4	R W	paragraphe 5.8.2.39
time_create	50	nfstime4	R W	paragraphe 5.8.2.40
time_delta	51	nfstime4	R	paragraphe 5.8.2.41
time_metadata	52	nfstime4	R	paragraphe 5.8.2.42
time_modify	53	nfstime4	R	paragraphe 5.8.2.43
time_modify_set	54	settime4	W	paragraphe 5.8.2.44

Tableau 5

5.8 Définition des attributs

5.8.1 Définitions des attributs exigés

5.8.1.1 Attribut 0 : supported_attrs

Vecteur binaire qui restituerait tous les attributs EXIGÉS et RECOMMANDÉS qui sont pris en charge pour cet objet. La portée de cet attribut s'applique à tous les objets avec un fsid qui correspond.

5.8.1.2 Attribut 1 : type

Désigne le type d'un objet en termes de un des numéros de constantes spéciales :
* NF4REG désigne un fichier régulier.
* NF4DIR désigne un répertoire.
* NF4BLK désigne un fichier spécial d'appareil de bloc.
* NF4CHR désigne un fichier spécial d'appareil de caractères.
* NF4LNK désigne une liaison symbolique.
* NF4SOCK désigne un fichier spécial de prise désignée.
* NF4FIFO désigne un fichier spécial fifo.
* NF4ATTRDIR désigne un répertoire des attributs nommés.
* NF4NAMEDATTR désigne un attribut nommé.

Dans le texte explicatif et les descriptions d'opération, les phrases suivantes sont utilisées avec la signification ci-dessous :
* La phrase "est un répertoire" signifie que l'attribut de type de l'objet est NF4DIR ou NF4ATTRDIR.
* La phrase "est un fichier spécial" signifie que l'attribut de type de l'objet est NF4BLK, NF4CHR, NF4SOCK, ou NF4FIFO.
* Les phrases "est un fichier ordinaire" et "est un fichier régulier" signifient que l'attribut de type de l'objet est NF4REG ou NF4NAMEDATTR.

5.8.1.3 Attribut 2 : fh_expire_type

Le serveur utilise cela pour spécifier un comportement d'expiration de bride de fichier au client. Voir à la Section 4 une description supplémentaire.

5.8.1.4 Attribut 3 : change

Valeur créée par le serveur que le client peut utiliser pour déterminer si les données de fichier, le contenu de répertoire, ou les attributs de l'objet ont été modifiés. Le serveur peut retourner l'attribut time_metadata de l'objet pour cette valeur d'attribut, mais seulement si l'objet de système de fichiers ne peut pas être mis à jour plus fréquemment que la résolution de time_metadata.

5.8.1.5 Attribut 4 : size

Taille de l'objet en octets.

5.8.1.6 Attribut 5 : link_support

VRAI, si le système de fichiers de l'objet prend en charge les liaisons physiques.

5.8.1.7 Attribut 6 : symlink_support

VRAI, si le système de fichiers de l'objet prend en charge les liaisons symboliques.

5.8.1.8 Attribut 7 : named_attr

VRAI, si cet objet a des attributs nommés. En d'autres termes, l'objet a un répertoire des attributs nommés non vide.

5.8.1.9 Attribut 8 : fsid

Identifiant univoque de système de fichiers pour le système de fichiers qui détient cet objet. L'attribut fsid a des composants majeurs et mineurs, dont chacun est du type de données uint64_t.

5.8.1.10 Attribut 9 : unique_handles

VRAI, si il est garanti que deux brides de fichier distinctes se réfèrent à deux objets différents de système de fichiers.

5.8.1.11 Attribut 10 : lease_time

Durée du prêt au serveur en secondes.

5.8.1.12 Attribute 11 : rdattr_error

Erreur retournée d'une tentative de restituer des attributs durant une opération READDIR.

5.8.1.13 Attribut 19 : filehandle

Bride de fichier de cet objet (principalement pour les demandes READDIR).

5.8.1.14 Attribut 75 : suppattr_exclcreat

Vecteur de bits qui va établir tous les attributs EXIGÉS et RECOMMANDÉS qui sont pris en charge par la méthode EXCLUSIVE4_1 de création de fichier via l'opération OPEN. La portée de cet attribut s'applique à tous les objets avec un fsid qui correspond.

5.8.2 Définition des attributs recommandés non catégorisés

Les définitions de la plupart des attributs RECOMMANDÉS suivent. Les collections qui partagent une catégorie commune sont définies dans d'autres paragraphes.

5.8.2.1 Attribut 14 : archive

VRAI, si ce fichier a été archivé depuis la dernière modification (déconseillé en faveur de time_backup).

5.8.2.2 Attribut 15 : cansettime

VRAI, si le serveur est capable de changer les heures pour un objet de système de fichiers comme spécifié dans une opération SETATTR.

5.8.2.3 Attribut 16 : case_insensitive

VRAI, si les comparaisons de nom de fichier sur ce système de fichiers sont insensibles à la casse.

5.8.2.4 Attribute 17 : case_preserving

VRAI, si la casse du nom de fichier sur ce système de fichiers est préservée.

5.8.2.5 Attribute 60 : change_policy

Valeur créée par le serveur que le client peut utiliser pour déterminer si certaines politiques de serveur relatives au système de fichiers courant ont subi un changement. Si la valeur reste la même, alors le client peut être sûr que les valeurs des attributs relatifs aux champs fs_location et fss_type de l'attribut fs_status n'ont pas changé. Par ailleurs, un changement dans cette valeur n'implique pas nécessairement un changement de politique. Il appartient au client d'interroger le serveur pour déterminer si une politique pertinente pour lui a changé. Voir les détails au paragraphe 3.3.6.

Cet attribut DOIT changer quand la valeur retournée par l'attribut fs_locations ou fs_locations_info change, quand un système de fichiers passe de lecture seule à écrivable ou vice versa, ou quand l'ensemble admis de nuances de sécurité pour le système de fichiers ou une de ses parties est changé.

5.8.2.6 Attribut 18 : chown_restricted

Si cet attribut est VRAI, le serveur va rejeter toute demande de changer le propriétaire ou le groupe associé à un fichier si l'appelant n'est pas un utilisateur privilégié (par exemple, "root" dans les environnements de fonctionnement UNIX ou, dans Windows 2000, le privilège "Take Ownership").

5.8.2.7 Attribut 20 : fileid

Numéro qui identifie de façon univoque le fichier dans le système de fichiers.

5.8.2.8 Attribut 21 : files_avail

Créneaux de fichier disponibles pour cet utilisateur sur le système de fichiers contenant cet objet -- ce devrait être la plus petite limite pertinente.

5.8.2.9 Attribut 22 : files_free

Créneaux de fichier libres sur le système de fichiers contenant cet objet -- ce devrait être la plus petite limite pertinente.

5.8.2.10 Attribut 23: files_total

Total des créneaux de fichier sur le système de fichiers contenant cet objet.

5.8.2.11 Attribut 76 : fs_charset_cap

Capacités de jeux de caractères pour ce système de fichiers. Voir au paragraphe 14.4.

5.8.2.12 Attribute 24 : fs_locations

Localisations où ce système de fichiers peut être trouvé. Si le serveur retourne NFS4ERR_MOVED comme erreur, cet attribut DOIT être pris en charge. Voir les détails au paragraphe 11.16.

5.8.2.13 Attribut 67 : fs_locations_info

Pleine fonction de localisation de système de fichiers. Voir les détails au paragraphe 11.17.2.

5.8.2.14 Attribut 61 : fs_status

Informations génériques de type de système de fichiers. Voir les détails au paragraphe 11.18.

5.8.2.15 Attribut 25 hidden

VRAI, si le fichier est considéré comme caché à l'égard de l'API Windows.

5.8.2.16 Attribut 26 homogeneous

VRAI, si le système de fichiers de cet objet est homogène ; c'est-à-dire, tous les objets dans le système de fichiers (tous les objets sur le serveur avec le même fsid) ont des valeurs communes pour tous les attributs par système de fichier.

5.8.2.17 Attribut 27 : maxfilesize

Taille de fichier maximum prise en charge pour le système de fichiers de cet objet.

5.8.2.18 Attribut 28 : maxlink

Nombre maximum de liaisons pour cet objet.

5.8.2.19 Attribut 29 : maxname

Taille maximum de nom de fichier prise en charge pour cet objet.

5.8.2.20 Attribut 30 : maxread

Quantité maximum de données que l'opération READ va retourner pour cet objet.

5.8.2.21 Attribut 31 : maxwrite

Quantité maximum de données que l'opération WRITE va accepter pour cet objet. Cet attribut DEVRAIT être pris en charge si le fichier est écrivable. L'absence de cet attribut peut conduire à ce que le client gaspille la bande passante ou n'obtienne pas les meilleures performances.

5.8.2.22 Attribut 32 : mimetype

Type/sous type du corps MIME de cet objet.

5.8.2.23 Attribut 55 : mounted_on_fileid

Comme fileid, mais si la bride de fichier cible est la racine d'un système de fichiers, cet attribut représente l'identifiant de fichier *(fileid)* du répertoire sous-jacent.
Les environnements de fonctionnement fondés sur UNIX connectent un système de fichiers dans l'espace de noms en connectant (montant) le système de fichiers sur l'objet de fichier existant (le point de montage, généralement un répertoire) d'un système de fichiers existant. Quand le répertoire parent du point de montage est lu par une API comme readdir(), les résultats en retour sont les entrées du répertoire, chacune avec un nom de composant et un identifiant de fichier. Le fileid de l'entrée de répertoire du point de montage va être différent du fileid que retourne l'invocation de système stat(). L'invocation de système stat() retourne le fileid de la racine du système de fichiers monté, tandis que readdir() retourne le fileid que stat() aurait retourné avant qu'un système de fichiers soit monté au point de montage.

À la différence de NFSv3, NFSv4.1 permet que la demande LOOKUP d'un client traverse d'autres systèmes de fichiers. Le client détecte la traversée de système de fichiers chaque fois que l'argument de bride de fichier de LOOKUP a un attribut fsid différent de celui de la bride de fichier retournée par LOOKUP. Un client fondé sur UNIX va considérer cela comme un "croisement de point de montage". UNIX a un schéma traditionnel pour permettre à un processus de déterminer son répertoire de travail actuel. Cela s'appuie sur le readdir() du parent d'un point de montage et le stat() du point de montage qui retourne des identifiants de fichier comme décrit précédemment. L'attribut mounted_on_fileid correspond au fileid que readdir() aurait retourné comme décrit précédemment.

Alors que le client NFSv4.1 pourrait simplement fabriquer un fileid correspondant à ce que mounted_on_fileid fournit (et si le serveur ne prend pas en charge mounted_on_fileid, le client n'a pas le choix) il y a un risque que le client génère un fileid en conflit avec un déjà alloué à un autre objet dans le système de fichiers. À la place, si le serveur peut fournir le mounted_on_fileid, les problèmes potentiels pour le fonctionnement du client dans ce domaine sont éliminés.

Si le serveur détecte qu'il n'y a pas de point de montage à l'objet de fichier cible, alors la valeur pour mounted_on_fileid qu'il retourne est la même que celle de l'attribut fileid.

L'attribut mounted_on_fileid est RECOMMANDÉ, de sorte que le serveur DEVRAIT le fournir si possible, et pour un serveur fondé sur UNIX, c'est direct. Généralement, mounted_on_fileid va être demandé durant une opération READDIR, et dans ce cas, il est trivial (au moins pour les serveurs fondés sur UNIX) de retourner mounted_on_fileid car il est égal au fileid d'une entrée de répertoire retournée par readdir(). Si mounted_on_fileid est demandé dans une opération GETATTR, le serveur devrait respecter un invariant qui lui a retourné une valeur égale à l'entrée d'objet de fichier dans le répertoire parent de l'objet, c'est-à-dire, ce que readdir() aurait retourné. Certains environnements de fonctionnement permettent qu'une série de deux systèmes de fichiers ou plus soit montée sur un seul point de montage. Dans ce cas, pour que le

serveur respecte l'invariant sus-mentionné, il va devoir trouver le point de montage de base, et non les points de montage intermédiaires.

5.8.2.24 Attribut 34 : no_trunc

Si cet attribut est VRAI, alors si le client utilise un nom de fichier plus long que name_max, une erreur va être retournée au lieu du nom qui a été tronqué.

5.8.2.25 Attribut 35 : numlinks

Nombre de liaisons physiques à cet objet.

5.8.2.26 Attribut 36 : owner

Nom de la chaîne du propriétaire de cet objet.

5.8.2.27 Attribut 37 : owner_group

Nom de la chaîne du groupe propriétaire de cet objet.

5.8.2.28 Attribut 38 : quota_avail_hard

Valeur en octets qui représente la quantité d'espace de disque supplémentaire au delà de l'allocation actuelle qui peut être allouée à ce fichier ou répertoire avant que plus d'allocations soient refusées. Il est compris que cet espace peut être consommé par des allocations à d'autres fichiers ou répertoires.

5.8.2.29 Attribut 39 : quota_avail_soft

Valeur en octets qui représente la quantité d'espace de disque supplémentaire qui peut être allouée à ce fichier ou répertoire avant que l'utilisateur puisse raisonnablement être averti. On comprend que cet espace peut être consommé par des allocations à d'autres fichiers ou répertoires bien qu'il y ait une règle sur les autres fichiers ou répertoires possibles.

5.8.2.30 Attribut 40 : quota_used

Valeur en octets qui représente la quantité d'espace de disque utilisé par ce fichier ou répertoire et éventuellement un certain nombre d'autres fichiers ou répertoires similaires, où l'ensemble des "similaires" respecte au moins le critère que l'allocation d'espace à tout fichier ou répertoire dans l'ensemble va réduire le "quota_avail_hard" de chaque autre fichier ou répertoire de l'ensemble.

Noter qu'il peut y avoir un certain nombre d'ensembles de fichiers ou répertoires distincts mais se chevauchant pour lesquels une valeur de quota_used est maintenue, par exemple, "tous les fichiers avec un certain propriétaire", "tous les fichiers avec un certain groupe de propriétaires", etc. Le serveur est libre de choisir un de ces ensembles quand il fournit le contenu de l'attribut quota_used, mais devrait le faire de façon répétable. La règle peut être configurée par système de fichiers ou peut être de "choisir l'ensemble avec le plus petit quota".

5.8.2.31 Attribut 41 : rawdev

Numéro d'appareil brut de fichier de type NF4BLK ou NF4CHR. Le numéro d'appareil est partagé en numéro majeur et numéro mineur. Si l'attribut de type du fichier n'est pas NF4BLK ou NF4CHR, la valeur retournée NE DEVRAIT PAS être considérée comme utile.

5.8.2.32 Attribut 42 : space_avail

Espace de disque en octets disponible à cet utilisateur sur le système de fichiers contenant cet objet -- ce devrait être la plus petite limite pertinente.

5.8.2.33 Attribut 43 : space_free

Espace de disque libre en octets sur le système de fichiers contenant cet objet -- ce devrait être la plus petite limite pertinente.

5.8.2.34 Attribut 44 : space_total

Espace de disque total en octets sur le système de fichiers contenant cet objet.

5.8.2.35 Attribut 45 : space_used

Nombre d'octets de système de fichiers alloué à cet objet.

5.8.2.36 Attribut 46 : system

Cet attribut est VRAI si ce fichier est un fichier "system" à l'égard de l'environnement de fonctionnement Windows.

5.8.2.37 Attribut 47 : time_access

L'attribut time_access représente l'heure du dernier accès à l'objet par une opération READ envoyée au serveur. La notion de ce qu'est un "accès" dépend de l'environnement de fonctionnement du serveur et/ou de la sémantique du système de fichiers du serveur. Par exemple, pour les serveurs qui suivent la sémantique d'interface de système d'exploitation portable (POSIX, *Portable Operating System Interface*) time_access va être mis à jour seulement par les opérations READ et READDIR et aucune des opérations qui modifient le contenu de l'objet [IEEE-1003.1]. Bien sûr, le réglage de l'attribut time_access_set correspondant est une autre façon de modifier l'attribut time_access.

Chaque fois que l'objet de fichier réside sur un système de fichiers écrivable, le serveur devrait faire de son mieux pour enregistrer time_access dans une mémorisation stable. Cependant, pour atténuer les effets de cela sur les performances, et plus spécialement chaque fois que le serveur effectue la lecture du contenu de l'objet à partir de sa mémoire tampon, le serveur PEUT mettre en mémoire tampon les mises à jour de l'heure d'accès et les écrire tranquillement dans une mémorisation stable. Il est aussi acceptable de donner aux administrateurs du serveur l'option de désactiver les mises à jour de time_access.

5.8.2.38 Attribut 48 : time_access_set

Règle l'heure du dernier accès à l'objet. Utilisé seulement par SETATTR.

5.8.2.39 Attribut 49 : time_backup

Heure de la dernière sauvegarde de l'objet.

5.8.2.40 Attribut 50 : time_create

Heure de création de l'objet. Cet attribut n'a aucune relation avec l'attribut traditionnel de fichier UNIX "ctime" ou "change time".

5.8.2.41 Attribut 51 : time_delta

Plus petite granularité de temps utile du serveur.

5.8.2.42 Attribut 52 : time_metadata

Heure de la dernière modification de métadonnées de l'objet.

5.8.2.43 Attribut 53 : time_modify

Heure de la dernière modification de l'objet.

5.8.2.44 Attribut 54 : time_modify_set

Règle l'heure de la dernière modification de l'objet. Utilisé seulement par SETATTR.

5.9 Interprétation de owner et de owner_group

Les attributs RECOMMANDÉS "owner" et "owner_group" (et aussi les utilisateurs et les groupes dans l'attribut "acl") sont représentés par une chaîne UTF-8. Pour éviter une représentation liée à une mise en œuvre sous-jacente particulière chez le client ou serveur, l'utilisation de la chaîne UTF-8 a été choisie. Noter que le paragraphe 6.1 de la [RFC2624] fournit des raisons supplémentaires. On s'attend à ce que le client et le serveur aient leur propre représentation locale de "owner" et

"owner_group" qui est utilisée pour la mémorisation locale ou la présentation à l'utilisateur final. Donc, on s'attend à ce que quand ces attributs sont transférés entre le client et le serveur, la représentation locale soit traduite en une syntaxe de la forme "utilisateur@dns_domaine". Cela permettra à un client et un serveur qui n'utilisent pas la même représentation locale de traduire en une syntaxe commune qui puisse être interprétée par les deux.

De même, les principaux de sécurité peuvent être représentés de différentes façons par différents mécanismes de sécurité. Les serveurs traduisent normalement ces représentations dans un format commun, généralement celui utilisé par la mémorisation locale, pour servir de moyen d'identification des utilisateurs correspondant à ces principaux de sécurité. Quand ces identifiants locaux sont traduits sous la forme de l'attribut "owner", associé aux fichiers créé par ces principaux, ils identifient, dans un format commun, les utilisateurs associés à chaque ensemble correspondant de principaux de sécurité.

La traduction utilisée pour interpréter les chaînes owner et group n'est pas spécifiée au titre de ce protocole. Cela permet d'employer diverses solutions. Par exemple, un tableau local de traduction peut être consulté qui transpose un identifiant numérique en la syntaxe utilisateur@dns_domaine. Un service de noms peut aussi être utilisé pour accomplir la traduction. Un serveur peut fournir un service plus général, non limité par une traduction particulière (qui traduirait seulement un ensemble limité de chaînes possibles) en mémorisant les attributs owner et owner_group dans une mémorisation locale sans aucune traduction ou il peut augmenter une méthode de traduction en mémorisant la chaîne entière pour les attributs pour lesquels aucune traduction n'est disponible tout en utilisant la représentation locale pour les cas dans lesquels une traduction est disponible.

Les serveurs qui ne fournissent pas la prise en charge de toutes les valeurs possibles des attributs owner et owner_group DEVRAIENT retourner une erreur (NFS4ERR_BADOWNER) quand une chaîne est présentée qui n'a pas de traduction, lorsque la valeur est à établir pour un SETATTR des attributs owner, owner_group, ou acl. Quand un serveur accepte une valeur de owner ou owner_group comme valide sur un SETATTR (et de même pour les chaînes owner et group dans une acl) il promet de retourner cette même chaîne quand un GETATTR correspondant est fait. Les changements de configuration (y compris les changements de la transposition de chaîne en la représentation locale) et les traductions de noms mal construites (celles qui contiennent un alias) peuvent rendre cette promesse impossible à honorer. Les serveurs devraient faire les efforts appropriés pour éviter une situation dans laquelle ces attributs ont leurs valeurs changées quand un changement non réel de propriété s'est produit.

La portion "dns_domaine" de la chaîne owner est destinée à être un nom de domaine du DNS, par exemple, utilisateur@exemple.org. Les serveurs devraient accepter comme valide un ensemble d'utilisateurs pour au moins un domaine. Un serveur peut traiter les autres domaines comme n'ayant pas de traduction valide. Un service plus général est fourni quand un serveur est capable d'accepter des utilisateurs pour plusieurs domaines, ou pour tous les domaines, sous réserve des contraintes de sécurité.

Dans le cas où il n'y a pas de traduction disponible au client ou au serveur, la valeur de l'attribut va être construite sans le "@". Donc, l'absence du @ dans l'attribut owner ou owner_group signifie qu'aucune traduction n'était disponible chez l'envoyeur et que le receveur de l'attribut ne devrait pas utiliser cette chaîne comme base de traduction dans son propre format interne. Même si la valeur de l'attribut ne peut pas être traduite, elle peut quand même être utile. Dans le cas d'un client, la chaîne d'attribut peut être utilisée pour l'affichage local du propriétaire.

Pour fournir un plus grand degré de compatibilité avec NFSv3, qui identifiait les utilisateurs et les groupes par des identifiants d'utilisateur et des identifiants de groupe de 32 bits non signés, les chaînes de propriétaire et de groupe qui consistent en valeurs numériques décimales sans zéros en tête peuvent avoir une interprétation particulière par les clients et serveurs qui choisissent de les prendre en charge. Le receveur peut traiter une telle chaîne d'utilisateur ou groupe comme représentant le même utilisateur que celui qui serait représenté par un uid ou gid NFSv3 ayant la valeur numérique correspondante. Un serveur n'est pas obligé d'accepter une telle chaîne, mais peut retourner à la place une erreur NFS4ERR_BADOWNER. Pour éviter d'utiliser ce mécanisme à la place de la traduction d'utilisateur et de groupe, afin qu'un client pusse passer tous les propriétaires et groupes en forme numérique, un serveur DEVRAIT retourner une erreur NFS4ERR_BADOWNER quand il y a une traduction valide pour l'utilisateur ou propriétaire désigné de cette façon. Dans ce cas, le client doit utiliser la chaîne nom@domaine appropriée et non la forme spéciale pour la compatibilité.

La chaîne de propriétaire "nobody" *(personne)* peut être utilisée pour désigner un utilisateur anonyme, qui va être associée à un fichier créé par un principal de sécurité qui ne peut pas être transposé par les moyens normaux en l'attribut owner. Les utilisateurs et les mises en œuvre de NFSv4.1 NE DEVRAIENT PAS utiliser "nobody" pour désigner un utilisateur réel dont l'accès n'est pas anonyme.

5.10 Attributs de casse de caractères

À l'égard des attributs case_insensitive et case_preserving, chaque caractère UCS-4 (que code UTF-8) peut être transposé conformément à l'Appendice B.2 de la [RFC3454]. Pour le traitement général des caractères et les questions d'internationalisation, voir la Section 14.

5.11 Attributs de notification de répertoire

Comme décrit au paragraphe 18.39, le client peut demander un délai minimum pour les notifications de changements aux attributs, mais le serveur est libre d'ignorer ce que le client demande. Le client peut déterminer à l'avance quels délais de notification le serveur va accepter en envoyant une opération GETATTR pour l'un ou l'autre ou les deux de deux attributs de notification de répertoire. Quand le client invoque l'opération GET_DIR_DELEGATION et demande des notifications de changement d'attribut, il devrait demander des délais de notification qui ne sont pas moins que les valeurs d'attribut fournies par le serveur.

5.11.1 Attribut 56 : dir_notif_delay

L'attribut dir_notif_delay est le nombre minimum de secondes dont le serveur va retarder avant de notifier au client un changement des attributs du répertoire.

5.11.2 Attribut 57 : dirent_notif_delay

L'attribut dirent_notif_delay est le nombre minimum de secondes dont le serveur va retarder avant de notifier au client un changement à un objet de fichier qui a une entrée dans le répertoire.

5.12 Définitions d'attributs pNFS

5.12.1 Attribut 62 : fs_layout_type

L'attribut fs_layout_type (voir le paragraphe 3.3.13) s'applique à un système de fichiers et indique quels types de disposition sont pris en charge par le système de fichiers. Quand le client rencontre un nouveau fsid, le client DEVRAIT obtenir la valeur pour l'attribut fs_layout_type associé au nouveau système de fichiers. Cet attribut est utilisé par le client pour déterminer si les types de disposition pris en charge par le serveur correspondent à une des types de disposition pris en charge par le client.

5.12.2 Attribut 66 : layout_alignment

Quand un client détient des dispositions sur les fichiers d'un système de fichiers, l'attribut layout_alignment indique l'alignement préféré pour les entrées/sorties aux fichiers sur ce système de fichiers. Lorsque possible, le client devrait envoyer des opérations READ et WRITE avec des décalages qui sont tous des multiples de l'attribut layout_alignment.

5.12.3 Attribut 65 : layout_blksize

Quand un client détient des dispositions sur les fichiers d'un système de fichiers, l'attribut layout_blksize indique la taille de bloc préférée pour les entrées/sorties aux fichiers sur ce système de fichiers. Lorsque possible, le client devrait envoyer des opérations READ avec un argument count qui est un multiple entier de layout_blksize, et des opérations WRITE avec un argument data d'une taille qui est un multiple entier de layout_blksize.

5.12.4 Attribut 63 : layout_hint

L'attribut layout_hint (voir le paragraphe 3.3.19) peut être établi sur de nouveaux fichiers créés pour influencer le choix du serveur de métadonnées pour la disposition du fichier. Si possible, cet attribut est un de ceux établis dans les attributs initiaux dans l'opération OPEN. Le serveur de métadonnées peut choisir d'ignorer cet attribut. L'attribut layout_hint est un sous ensemble de la structure de disposition retournée par LAYOUTGET. Par exemple, au lieu de spécifier des appareils particuliers, cela serait utilisé pour suggérer la largeur de bande d'un fichier. La mise en œuvre de serveur détermine quels champs dans la disposition vont être utilisés.

5.12.5 Attribut 64 : layout_type

Cet attribut fait la liste des types de disposition disponibles pour un fichier. La valeur retournée par le serveur est seulement pour information. Le client va utiliser l'opération LAYOUTGET pour obtenir les informations nécessaires afin d'effectuer les entrées/sorties, par exemple, les informations d'appareil spécifiques pour le fichier et sa disposition.

5.12.6 Attribut 68 : mdsthreshold

Cet attribut est une indication fournie par un serveur utilisée pour communiquer au client quand il est plus efficace d'envoyer des opérations READ et WRITE au serveur de métadonnées ou au serveur de données. Les deux types de seuils décrits sont les seuils de taille de fichier et les seuils de taille d'entrée/sortie. Si la taille d'un fichier est inférieure au seuil de taille de fichier, l'accès aux données DEVRAIT être envoyé au serveur de métadonnées. Si une demande d'entrée/sortie a une longueur inférieure au seuil de taille d'entrée/sortie, l'entrée/sortie DEVRAIT être envoyée au serveur de métadonnées. Chaque type de seuil est spécifié séparément pour la lecture et l'écriture.

Le serveur PEUT fournir les deux types de seuils pour un fichier. Si les deux tailles de fichier et taille d'entrée/sortie sont fournies, le client DEVRAIT atteindre ou excéder les deux seuils avant d'envoyer ses demandes de lecture ou écriture au serveur de données. Autrement, si seulement un des seuils spécifiés est atteint ou excédé, les demandes d'entrée/sortie sont envoyées au serveur de métadonnées.

Pour chaque type de seuil, une valeur de zéro indique qu'aucune demande READ ou WRITE ne devrait être envoyée au serveur de métadonnées, tandis qu'une valeur toute de uns indique que toutes les demandes READ ou WRITE devraient être envoyées au serveur de métadonnées.

L'attribut est disponible par bride de fichier. Si la bride de fichier courante se réfère à un fichier ou répertoire non pNFS, le serveur de métadonnées devrait retourner un attribut représentatif du système de fichiers de la bride de fichier. Il est suggéré que cet attribut soit interrogé au titre de l'opération OPEN. Du fait de changements dynamiques du système, le client ne devrait pas supposer que l'attribut va rester constant pour une période spécifique ; donc, il devrait être rafraîchi périodiquement.

5.13 Attributs de rétention

La rétention est un concept par lequel un objet fichier peut être placé dans un état immuable, non supprimable, non renommable pour une durée fixe ou infinie. Une fois dans cet état "retenu", le fichier ne peut pas être déplacé hors de l'état jusqu'à la fin de la période de rétention.

Quand la rétention est activée, la rétention DOIT s'étendre aux données du fichier, et au nom du fichier. Le serveur PEUT étendre la rétention à toute autre propriété du fichier, y compris aux sous ensembles d'attributs EXIGÉS, RECOMMANDÉS, et nommés, avec les exceptions mentionnées dans cette section.

Les serveurs PEUVENT prendre ou non en charge la rétention sur tout type d'objet de fichier.

Les cinq attributs de rétention sont expliqués dans les paragraphes qui suivent.

5.13.1 Attribut 69 : retention_get

Si la rétention est activée pour le fichier associé, la valeur de cet attribut représente l'heure de début de la rétention de l'objet de fichier. La valeur de cet attribut est seulement lisible avec l'opération GETATTR et NE DOIT PAS être modifiée par l'opération SETATTR (paragraphe 5.5). La valeur de l'attribut consiste en :

```
const RET4_DURATION_INFINITE = 0xffffffffffffffff ;
struct retention_get4 {
    uint64_t       rg_duration ;
    nfstime4       rg_begin_time<1> ;
};
```

Le champ rg_duration est la durée en secondes qui indique pendant combien de temps le fichier va être conservé une fois la rétention activée. Le champ rg_begin_time est un dispositif de jusqu'à une valeur de temps absolu. Si le dispositif est de longueur zéro, un temps de rétention sans début a été établi, et la rétention n'est pas activée. Si rg_duration est égal à

RET4_DURATION_INFINITE, le fichier, une fois la rétention activée, va être conservé pour une durée infinie.

Si (aussitôt que) rg_duration est zéro, alors rg_begin_time va être de longueur zéro, et là encore, la rétention n'est pas (n'est plus) activée.

5.13.2 Attribut 70 : retention_set

Cet attribut est utilisé pour régler la durée de rétention et facultativement activer la rétention pour l'objet de fichier associé. Cet attribut n'est modifiable que via l'opération SETATTR et NE DOIT PAS être restitué par l'opération GETATTR (paragraphe 5.5). Cet attribut correspond à retention_get. La valeur de l'attribut consiste en :

```
struct retention_set4 {
    bool        rs_enable ;
    uint64_t    rs_duration<1> ;
};
```

Si le client règle rs_enable à VRAI, alors il active la rétention sur l'objet de fichier avec le début du temps de rétention qui commence à la date et heure courante du serveur. La durée de la rétention peut aussi être fournie si le dispositif rs_duration est de longueur un. La durée est le temps en secondes depuis le début de la rétention, et si il est réglé à RET4_DURATION_INFINITE, le fichier est à conserver pour toujours. Si la rétention est activée, sans durée de rétention spécifiée ni dans ce SETATTR ni dans un précédent, la durée est par défaut de zéro seconde. Le serveur PEUT restreindre l'activation de rétention ou la durée de rétention sur la base de la permission ACL ACE4_WRITE_RETENTION. L'activation de rétention NE DOIT PAS empêcher l'activation de rétention fondée sur l'événement ou la modification de l'attribut retention_hold.

Les règles suivantes s'appliquent aux deux attributs retention_set et retentevt_set.

* Tant que la rétention n'est pas activée, il est permis au client de diminuer la durée.

* La durée peut toujours être réglée à une valeur égale ou supérieure, même si la rétention est activée. Noter que une fois que la rétention est activée, la durée réelle (comme retournée par les attributs retention_get ou retentevt_get ; voir le paragraphe 5.13.1 ou le paragraphe 5.13.3) est décomptée de façon constante jusqu'à zéro (une unité par seconde) sauf si la durée était réglée à RET4_DURATION_INFINITE. Donc, il ne va pas être possible au client d'étendre précisément la durée sur un fichier dont la rétention est activée.

* Lorsque la rétention est activée, les tentatives de désactiver la rétention ou de diminuer la durée de rétention DOIVENT échouer avec l'erreur NFS4ERR_INVAL.

* Si le principal qui tente de changer retention_set ou retentevt_set n'a pas les permissions ACE4_WRITE_RETENTION, la tentative DOIT échouer avec NFS4ERR_ACCESS.

5.13.3 Attribut 71 : retentevt_get

Obtient la durée de rétention fondée sur l'événement, et si elle est activée, l'heure de début de la rétention fondée sur l'événement de l'objet de fichier. Cet attribut est comme retention_get, mais se réfère à une rétention fondée sur l'événement. L'événement qui déclenche la rétention fondée sur l'événement n'est pas défini par la spécification NFSv4.1.

5.13.4 Attribut 72 : retentevt_set

Règle la durée de la rétention fondée sur l'événement, et facultativement active la rétention fondée sur l'événement sur l'objet de fichier. Cet attribut correspond à retentevt_get et est comme retention_set, mais se réfère à la rétention fondée sur l'événement. Quand la rétention fondée sur l'événement est établie, le fichier DOIT être conservé même si la rétention non fondée sur l'événement a été établie, et que la durée de la rétention non fondée sur l'événement a été atteinte. À l'inverse, quand la rétention non fondée sur l'événement a été établie, le fichier DOIT être conservé même si la rétention fondée sur l'événement a été établie, et la durée de la rétention fondée sur l'événement a été atteinte. Le serveur PEUT restreindre l'activation de la rétention fondée sur l'événement ou la durée de la rétention fondée sur l'événement sur la base de la permission ACL ACE4_WRITE_RETENTION. L'activation de la rétention fondée sur l'événement NE DOIT PAS empêcher l'activation de la rétention non fondée sur l'événement ou la modification de l'attribut retention_hold.

5.13.5 Attribut 73 : retention_hold

Obtient ou établit des prises de rétention administrative, une prise par position de bit.

Cet attribut permet de une à 64 prises administratives, une prise par bit sur l'attribut. Si retention_hold n'est pas zéro, alors le fichier NE DOIT PAS être supprimé, renommé, ou modifié, même si la durée sur la rétention fondée sur l'événement activé ou non fondée sur l'événement a été atteinte. Le serveur PEUT restreindre la modification de retention_hold sur la base de la permission d'ACL ACE4_WRITE_RETENTION_HOLD. L'activation de prises de rétention administrative n'empêche pas l'activation de la rétention fondée sur l'événement ou non fondée sur l'événement.

Si le principal qui tente de changer retention_hold n'a pas les permissions ACE4_WRITE_RETENTION_HOLD, la tentative DOIT échouer avec l'erreur NFS4ERR_ACCESS.

6. Attributs de contrôle d'accès

Les listes de contrôle d'accès (ACL, *Access Control List*) sont des attributs de fichier qui spécifient un contrôle d'accès de granularité fine. Cette section couvre les attributs de fichier "acl", "dacl", "sacl", "aclsupport", "mode", et "mode_set_masked" et leurs interactions. Noter que les attributs de fichier peuvent s'appliquer à tout objet de système de fichiers.

6.1 Buts

Les ACL et les modes représentent deux modèles bien établis pour spécifier les permissions. Ce paragraphe spécifie les exigences qui tentent d'atteindre les buts suivants :

* Si un serveur prend en charge l'attribut mode, il devrait fournir une sémantique raisonnable aux clients qui établissent et restituent seulement l'attribut mode.

* Si un serveur prend en charge les attributs d'ACL, il devrait fournir une sémantique raisonnable aux clients qui établissent et restituent seulement ces attributs.

* Sur les serveurs qui prennent en charge l'attribut mode, si des attributs d'ACL n'ont jamais été établis sur un objet, via héritage ou explicitement, le comportement devrait être celui de style UNIX traditionnel.

* Sur les serveurs qui prennent en charge l'attribut mode, si des attributs d'ACL ont été précédemment établis sur un objet, explicitement ou via héritage :
  - Établir seulement l'attribut mode devrait effectivement contrôler les permissions traditionnelles de style UNIX de lecture, écriture, et exécution sur owner, owner_group, et autres.
  - Établir seulement l'attribut mode devrait fournir une sécurité raisonnable. Par exemple, établir un mode de 000 devrait être suffisant pour assurer que les futures opérations OPEN pour OPEN4_SHARE_ACCESS_READ ou OPEN4_SHARE_ACCESS_WRITE par tout principal échouent, sans considération d'une ACL précédemment existante ou héritée.

* NFSv4.1 peut introduire une sémantique différente relative aux attributs mode et ACL, mais il ne rend pas invalide les mises en œuvre précédemment existantes. De plus, ce paragraphe fournit des éclaircissements fondés sur les précédentes mises en œuvre et les discussions autour d'elles.

* Sur les serveurs qui prennent en charge les deux attributs mode et acl ou dacl, le serveur doit assurer la cohérence entre les deux. La valeur de l'attribut mode (à l'exception des trois bits de poids fort décrits au paragraphe 6.2.4) doit être déterminée entièrement par la valeur de l'ACL, de sorte que l'utilisation du mode n'est jamais exigée pour autre chose qu'établir les trois bits de poids fort. Voir au paragraphe 6.4.1 les exigences exactes.

* Quand un attribut mode est établi sur un objet, les attributs d'ACL peuvent devoir être modifiés afin de ne pas entrer en conflit avec le nouveau mode. Dans ce cas, il est souhaitable que l'ACL conserve autant d'informations que possible. Cela inclut des informations sur l'héritage, les ACE AUDIT et ALARM, et les permissions accordées et refusées qui ne sont pas en conflit avec le nouveau mode.

6.2 Discussion des attributs de fichier

6.2.1 Attribut 12 : acl

L'attribut ACL de NFSv4.1 contient un dispositif d'entrées de contrôle d'accès (ACE, *Access Control Entry*) qui sont associées à l'objet de système de fichiers. Bien que le client puisse établir et obtenir l'attribut acl, le serveur est responsable de l'utilisation de l'ACL pour effectuer le contrôle d'accès. Le client peut utiliser les opérations OPEN ou ACCESS pour vérifier l'accès sans modifier ou lire les données ou métadonnées.

La structure d'ACE de NFS est définie comme suit :

```
typedef uint32_t       acetype4;
typedef uint32_t       aceflag4;
typedef uint32_t       acemask4;

struct nfsace4 {
      acetype4         type;
      aceflag4         flag;
      acemask4         access_mask;
      utf8str_mixed    who;
};
```

Pour déterminer si une demande réussit, le serveur traite chaque entrée de nfsace4 dans l'ordre. Seules les ACE qui ont un "who" qui correspond au demandeur sont considérées. Chaque ACE est traitée jusqu'à ce que tous les bits de l'accès du demandeur soient dans l'état ALLOWED *(permis)*. Une fois qu'un bit (voir ci-dessous) a été ALLOWED par une ACCESS_ALLOWED_ACE, il n'est plus considéré dans le traitement des ACE ultérieures. Si un ACCESS_DENIED_ACE est rencontré où l'accès du demandeur a encore un bit non ALLOWED en commun avec le "access_mask" *(gabarit d'accès)* de l'ACE, la demande est rejetée. Quand l'ACL est entièrement traitée, si il y a des bits dans le gabarit du demandeur qui n'ont pas été ALLOWED ou DENIED, l'accès est refusé.

À la différence des types d'ACE ALLOW et DENY, les types d'ACE ALARM et AUDIT n'affectent pas l'accès d'un demandeur, et sont plutôt pour des événement déclencheurs par suite de la tentative d'accès d'un demandeur. Donc, les ACE AUDIT et ALARM sont traités seulement après le traitement des ACE ALLOW et DENY.

Le modèle d'ACL NFSv4.1 est assez riche. Certaines plates-formes de serveur peuvent fournir une fonction de contrôle d'accès qui va au delà de l'attribut mode de style UNIX, mais qui n'est pas aussi riche que le modèle d'ACL de NFS. Afin que les utilisateurs puissent tirer parti de cette fonction plus limitée, le serveur peut prendre en charge les attributs acl en transposant son modèle d'ACL en modèle d'ACL NFSv4.1. Les serveurs doivent s'assurer que l'ACL qu'ils mémorisent ou appliquent est au moins aussi strict que l'ACL NFSv4 qui a été établi. Il est tentant d'accomplir cela en rejetant toute ACL qui sort du petit ensemble qui peut être représenté de façon précise. Cependant, cette approche peut rendre les ACL inutilisables sans une connaissance particulière côté client de la transposition du serveur, ce qui contrevient à l'objectif d'avoir un protocole commun d'ACL NFSv4. Donc, les serveurs devraient accepter toute ACL qu'ils peuvent sans compromettre la sécurité. Pour aider à réaliser cela, les serveurs peuvent faire une exception spéciale, dans le cas de bits de permission non pris en charges, à la règle que les bits non ALLOWED ou DENIED par une ACL doivent être refusés. Par exemple, un serveur de style UNIX pourrait choisir de permettre en silence des permissions d'attribut read même quand une ACL ne permet pas explicitement ces permissions. (Une ACL qui refuse explicitement la permission des attributs read devrait quand même être rejetée.)

La situation est compliquée par le fait qu'un serveur peut avoir plusieurs modules qui appliquent les ACL. Par exemple, l'application pour l'accès NFSv4.1 peut être différente de, mais pas plus faible, que l'application pour l'accès local, et les deux peuvent être différentes de l'application pour l'accès par d'autres protocoles comme le bloc de message de serveur (SMB, *Server Message Block*). Donc il peut être utile qu'un serveur accepte une ACL même si tous ses modules ne sont pas capables de la prendre en charge.

Le principe directeur à l'égard de l'accès NFSv4 est que le serveur ne doit pas accepter des ACL qui font paraître l'accès au fichier plus restrictif qu'il n'est réellement.

6.2.1.1 Type ACE

Les constantes utilisées pour le champ type (acetype4) sont les suivantes :

```
const ACE4_ACCESS_ALLOWED_ACE_TYPE  = 0x00000000;
const ACE4_ACCESS_DENIED_ACE_TYPE   = 0x00000001;
const ACE4_SYSTEM_AUDIT_ACE_TYPE    = 0x00000002;
const ACE4_SYSTEM_ALARM_ACE_TYPE    = 0x00000003;
```

Seuls les bits ALLOWED et DENIED peuvent être utilisés dans l'attribut dacl, et seuls les bits AUDIT et ALARM peuvent être utilisés dans l'attribut sacl. Les quatre sont permis dans l'attribut acl.

Valeur	Abréviation	Description
ACE4_ACCESS_ALLOWED_ACE_TYPE	ALLOW	Accorde explicitement l'accès défini dans acemask4 au fichier ou répertoire.
ACE4_ACCESS_DENIED_ACE_TYPE	DENY	Refuse explicitement l'accès défini dans acemask4 au fichier ou répertoire.
ACE4_SYSTEM_AUDIT_ACE_TYPE	AUDIT	Enregistre (d'une façon qui dépend du système) toute tentative d'accès à un fichier ou répertoire qui utilise une des méthodes d'accès spécifiées dans acemask4.
ACE4_SYSTEM_ALARM_ACE_TYPE	ALARM	Génère une alarme (d'une façon qui dépend du système) quand une tentative d'accès est faite à un fichier ou répertoire pour les méthodes d'accès spécifiées dans acemask4.

Tableau 6

La colonne "Abréviation" note comment on se réfère aux types dans la suite de ce paragraphe.

6.2.1.2 Attribut 13 : aclsupport

Un serveur n'a pas besoin de prendre en charge tous les types d'ACE ci-dessus. Cet attribut indique quels types d'ACE sont pris en charge pour le système de fichiers en cours. Les constantes de gabarit binaire utilisées pour représenter les définitions ci-dessus dans l'attribut aclsupport sont les suivantes :

```
const ACL4_SUPPORT_ALLOW_ACL   = 0x00000001 ;
const ACL4_SUPPORT_DENY_ACL    = 0x00000002 ;
const ACL4_SUPPORT_AUDIT_ACL   = 0x00000004 ;
const ACL4_SUPPORT_ALARM_ACL   = 0x00000008 ;
```

Les serveurs qui prennent en charge le type d'ACE ALLOW ou DENY DEVRAIENT prendre en charge les deux types d'ACE ALLOW et DENY.

Les clients ne devraient pas tenter d'établir une ACE sauf si le serveur revendique la prise en charge de ce type d'ACE. Si le serveur reçoit une demande d'établir une ACE qu'il ne peut pas mémoriser, il DOIT rejeter la demande avec NFS4ERR_ATTRNOTSUPP. Si le serveur reçoit une demande d'établir une ACE qu'il peut mémoriser mais ne peut pas appliquer, le serveur DEVRAIT rejeter la demande avec NFS4ERR_ATTRNOTSUPP.

La prise en charge de tout attribut d'ACL est facultative (bien que RECOMMANDÉE). Cependant, un serveur qui prend en charge un des nouveaux attributs d'ACL (dacl ou sacl) DOIT permettre d'utiliser les nouveaux attributs d'ACL pour accéder à touts les types d'ACE qu'il prend en charge. En d'autres termes, si ce serveur prend en charge les ACE ALLOW ou DENY, alors il DOIT prendre en charge l'attribut dacl, et si il prend en charge les ACE AUDIT ou ALARM, alors il DOIT prend en charge l'attribut sacl.

6.2.1.3 Gabarit d'accès ACE

Les constantes de gabarit binaire utilisées pour le champ "gabarit d'accès" sont les suivantes :

```
const ACE4_READ_DATA                = 0x00000001;
const ACE4_LIST_DIRECTORY           = 0x00000001;
const ACE4_WRITE_DATA               = 0x00000002;
const ACE4_ADD_FILE                 = 0x00000002;
const ACE4_APPEND_DATA              = 0x00000004;
const ACE4_ADD_SUBDIRECTORY         = 0x00000004;
const ACE4_READ_NAMED_ATTRS         = 0x00000008;
const ACE4_WRITE_NAMED_ATTRS        = 0x00000010;
```

```
const ACE4_EXECUTE                = 0x00000020;
const ACE4_DELETE_CHILD           = 0x00000040;
const ACE4_READ_ATTRIBUTES        = 0x00000080;
const ACE4_WRITE_ATTRIBUTES       = 0x00000100;
const ACE4_WRITE_RETENTION        = 0x00000200;
const ACE4_WRITE_RETENTION_HOLD   = 0x00000400;
const ACE4_DELETE                 = 0x00010000;
const ACE4_READ_ACL               = 0x00020000;
const ACE4_WRITE_ACL              = 0x00040000;
const ACE4_WRITE_OWNER            = 0x00080000;
const ACE4_SYNCHRONIZE            = 0x00100000;
```

Noter que certains gabarits ont des valeurs coïncidentes, par exemple, ACE4_READ_DATA et ACE4_LIST_DIRECTORY. Les entrées de gabarit ACE4_LIST_DIRECTORY, ACE4_ADD_FILE, et ACE4_ADD_SUBDIRECTORY sont destinées à être utilisées avec des objets de répertoire, tandis que ACE4_READ_DATA, ACE4_WRITE_DATA, et ACE4_APPEND_DATA sont destinées à être utilisées avec des objets non de répertoire.

6.2.1.3.1 Discussion des attributs de gabarit

ACE4_READ_DATA
Opérations affectées : READ, OPEN
Discussion : permission de lire les données du fichier. Les serveurs DEVRAIENT permettre à un utilisateur la capacité de lire les données du fichier quand seul le bit de gabarit d'accès ACE4_EXECUTE est permis.

ACE4_LIST_DIRECTORY
Opérations affectées : READDIR
Discussion : permission de faire la liste des contenus d'un répertoire.

ACE4_WRITE_DATA
Opérations affectées : WRITE, OPEN, SETATTR de taille
Discussion : permission de modifier les données d'un fichier.

ACE4_ADD_FILE
Opérations affectées : CREATE, LINK, OPEN, RENAME
Discussion : permission d'ajouter un nouveau fichier dans un répertoire. L'opération CREATE est affectée quand nfs_ftype4 est NF4LNK, NF4BLK, NF4CHR, NF4SOCK, ou NF4FIFO. (NF4DIR n'est pas mentionné parce que il est couvert par ACE4_ADD_SUBDIRECTORY.) OPEN est affecté quand il est utilisé pour créer un fichier régulier. LINK et RENAME sont toujours affectés.

ACE4_APPEND_DATA
Opérations affectées : WRITE, OPEN, SETATTR de taille
Discussion : la capacité de modifier les données d'un fichier, mais seulement en commençant à la fin du fichier (EOF). Cela permet la notion de fichier en ajout seulement, en permettant ACE4_APPEND_DATA et en refusant ACE4_WRITE_DATA au même utilisateur ou groupe. Si un fichier a une ACL telle que celle décrite ci-dessus et si une demande WRITE est faite pour ailleurs que EOF, le serveur DEVRAIT retourner NFS4ERR_ACCESS.

ACE4_ADD_SUBDIRECTORY
Opérations affectées : CREATE, RENAME
Discussion : permission de créer un sous répertoire dans un répertoire. L'opération CREATE est affectée quand nfs_ftype4 est NF4DIR. L'opération RENAME est toujours affectée.

ACE4_READ_NAMED_ATTRS
Opérations affectées : OPENATTR
Discussion : permission de lire les attributs nommés d'un fichier ou de chercher le répertoire des attributs nommés. OPENATTR est affecté quand il n'est pas utilisé pour créer un répertoire des attributs nommés. C'est quand 1) createdir est VRAI, mais un répertoire des attributs nommés existe déjà, ou 2) createdir est FAUX.

ACE4_WRITE_NAMED_ATTRS
Opérations affectées : OPENATTR
Discussion : permission d'écrire les attributs nommés d'un fichier ou de créer un répertoire des attributs nommés.

OPENATTR est affecté quand il est utilisé pour créer un répertoire des attributs nommés. C'est quand createdir est VRAI et qu'aucun répertoire des attributs nommés n'existe. La capacité de vérifier si un répertoire des attributs nommés existe ou non dépend de la capacité de le chercher ; donc, les utilisateurs ont aussi besoin de la permission ACE4_READ_NAMED_ATTRS afin de créer un répertoire des attributs nommés.

ACE4_EXECUTE
Opérations affectées : READ, OPEN, REMOVE, RENAME, LINK, CREATE
Discussion : permission d'exécuter un fichier. Les serveurs DEVRAIENT permettre à un utilisateur la capacité de lire les données du fichier quand seul le bit de gabarit d'accès ACE4_EXECUTE est permis. C'est parce que il n'y a pas de moyen d'exécuter un fichier sans en lire le contenu. Bien qu'un serveur puisse traiter de façon identique les bits ACE4_EXECUTE et ACE4_READ_DATA quand il décide de permettre une opération READ, il DEVRAIT aussi permettre que les deux bits soient établis indépendamment dans les ACL, et DOIT distinguer entre eux quand il répond aux opérations ACCESS. En particulier, les serveurs NE DEVRAIENT PAS changer en silence un des deux bits quand l'autre est établi, car cela rendrait impossible au client d'appliquer correctement la distinction entre les permissions read et execute. Par exemple, à la suite d'un SETATTR de l'ACL suivante : nfsuser:ACE4_EXECUTE:ALLOW

Un GETATTR suivant de ACL pour ce fichier DEVRAIT retourner : nfsuser:ACE4_EXECUTE:ALLOW

Plutôt que : nfsuser:ACE4_EXECUTE/ACE4_READ_DATA:ALLOW

ACE4_EXECUTE
Opérations affectées : LOOKUP
Discussion : permission de traverser/chercher un répertoire.

ACE4_DELETE_CHILD
Opérations affectées : REMOVE, RENAME
Discussion : permission de supprimer un fichier ou répertoire au sein d'un répertoire. Voir au paragraphe 6.2.1.3.2 des informations sur l'interaction de ACE4_DELETE et ACE4_DELETE_CHILD.

ACE4_READ_ATTRIBUTES
Opérations affectées : GETATTR d'attributs d'objet de système de fichiers, VERIFY, NVERIFY, READDIR
Discussion : capacité de lire les attributs de base (non ACL) d'un fichier. Sur un système UNIX, les attributs de base peuvent être vus comme les attributs de niveau stat. Permettre le bit de gabarit d'accès signifierait que l'entité peut exécuter "ls -l" et stat. Si une opération READDIR demande des attributs, ce gabarit doit être admis pour que le READDIR réussisse.

ACE4_WRITE_ATTRIBUTES
Opérations affectées : SETATTR de time_access_set, time_backup, time_create, time_modify_set, mimetype, hidden, system
Discussion : permission de changer l'heure associée à un fichier ou répertoire en une valeur arbitraire. C'est aussi la permission de changer les attributs mimetype, hidden, et system. Un utilisateur qui a ACE4_WRITE_DATA ou ACE4_WRITE_ATTRIBUTES va avoir la permission de régler les temps associés à un fichier à l'heure actuelle du serveur.

ACE4_WRITE_RETENTION
Opérations affectées : SETATTR de retention_set, retentevt_set.
Discussion : permission de modifier les durées de rétention fondée sur l'événement et non fondée sur l'événement. Aussi permission d'activer la rétention fondée sur l'événement et non fondée sur l'événement. Un serveur PEUT se comporter de telle façon que l'établissement de ACE4_WRITE_ATTRIBUTES permette ACE4_WRITE_RETENTION.

ACE4_WRITE_RETENTION_HOLD
Opérations affectées : SETATTR de retention_hold.
Discussion : permission de modifier les prise de rétention administrative. Un serveur PEUT transposer ACE4_WRITE_ATTRIBUTES en ACE_WRITE_RETENTION_HOLD.

ACE4_DELETE
Opérations affectées : REMOVE
Discussion : permission de supprimer le fichier ou répertoire. Voir au paragraphe 6.2.1.3.2 des informations sur l'inter action de ACE4_DELETE et ACE4_DELETE_CHILD.

ACE4_READ_ACL

Opérations affectées : GETATTR de acl, dacl, ou sacl, NVERIFY, VERIFY
Discussion : permission de lire l'ACL.

ACE4_WRITE_ACL
Opérations affectées : SETATTR de acl et mode
Discussion : permission d'écrire les attributs acl et mode.

ACE4_WRITE_OWNER
Opérations affectées : SETATTR de owner et owner_group
Discussion : permission d'écrire les attributs owner et owner_group. Sur les systèmes UNIX, c'est la capacité d'exécuter chown() et chgrp().

ACE4_SYNCHRONIZE
Opérations affectées : NONE
Discussion : permission d'utiliser l'objet de fichier comme une primitive de synchronisation pour une communication inter processus. Cette permission n'est pas appliquée ou interprétée par le serveur NFSv4.1 au nom du client. Normalement, la permission ACE4_SYNCHRONIZE n'a de signification que sur les systèmes locaux de fichiers, c'est-à-dire, des systèmes de fichiers non accédés via NFSv4.1. La raison de l'existence du bit de permission est que certains environnements d'exploitation, comme Windows, utilisent ACE4_SYNCHRONIZE. Par exemple, si un client copie un fichier qui a ACE4_SYNCHRONIZE établi à partir d'un système local de fichiers à un serveur NFSv4.1, et que plus tard il copie le fichier venant du serveur NFSv4.1 sur un système local de fichiers, il est probable que si ACE4_SYNCHRONIZE était établi dans le fichier original, le client va vouloir qu'il soit établi dans la seconde copie. La première copie ne va pas avoir la permission établie sauf si le serveur NFSv4.1 a le moyen d'établir le bit ACE4_SYNCHRONIZE. La seconde copie n'aura pas la permission établie si le serveur NFSv4.1 n'a pas de moyen de restituer le bit ACE4_SYNCHRONIZE.

Les mises en œuvre de serveur n'ont pas besoin de fournir la granularité de contrôle qui est impliquée par cette liste de gabarits. Par exemple, les systèmes fondés sur POSIX pourraient ne pas distinguer ACE4_APPEND_DATA (la capacité d'ajouter à un fichier) de ACE4_WRITE_DATA (la capacité de modifier les contenus existants) ; les deux gabarits seraient liés à une seule permission "écrire" [IEEE-1003.1]. Quand un tel serveur retourne des attributs au client, il va montrer les deux ACE4_APPEND_DATA et ACE4_WRITE_DATA si et seulement si la permission d'écriture est activée.

Si un serveur reçoit une demande SETATTR qu'il ne peut pas mettre en œuvre de façon précise, il devrait aller dans la direction d'un accès plus restreint, sauf dans les cas discutés précédemment de execute et read. Par exemple, supposons qu'un serveur ne puisse pas distinguer l'écrasement de données et l'ajout de nouvelles données, comme décrit au paragraphe précédent. Si un client soumet une ACE ALLOW lorsque ACE4_APPEND_DATA est établi mais ACE4_WRITE_DATA ne l'est pas (ou vice versa) le serveur devrait soit désactiver ACE4_APPEND_DATA, soit rejeter la demande avec NFS4ERR_ATTRNOTSUPP.

6.2.1.3.2 ACE4_DELETE ou ACE4_DELETE_CHILD

Deux bits de gabarit d'accès gouvernent la capacité de supprimer une entrée de répertoire : ACE4_DELETE sur l'objet lui-même (la "cible") et ACE4_DELETE_CHILD sur le répertoire contenant (le "parent").

De nombreux systèmes prennent aussi le "bit collant" (MODE4_SVTX) sur un répertoire pour permettre de quitter la liaison seulement à un utilisateur qui possède la cible ou le parent ; sur certains de ces systèmes la décision dépend aussi de si la cible est écrivable.

Les serveurs DEVRAIENT permettre de quitter la liaison si ACE4_DELETE est permis sur la cible, ou si ACE4_DELETE_CHILD est permis sur le parent. (Noter que ceci est vrai même si le parent ou la cible refuse explicitement une de ces permissions.)

Si les ACL en question ne permettent explicitement ni ne refusent une des conditions si-dessus, et si MODE4_SVTX n'est pas établi sur le parent, alors le serveur DEVRAIT permettre la suppression si et seulement si ACE4_ADD_FILE est permis. Dans le cas où MODE4_SVTX est établi, le serveur peut aussi exiger que le suppresseur possède le parent ou la cible, ou peut exiger que la cible soit écrivable.

Cela permet aux serveurs de prendre en charge quelque chose d'assez proche de la sémantique traditionnelle de style UNIX, avec ACE4_ADD_FILE à la place du bit écrire.

6.2.1.4 Fanion ACE

Les constantes de gabarit binaire utilisées pour le champ Fanions sont les suivantes :

```
const ACE4_FILE_INHERIT_ACE             = 0x00000001;
const ACE4_DIRECTORY_INHERIT_ACE        = 0x00000002;
const ACE4_NO_PROPAGATE_INHERIT_ACE     = 0x00000004;
const ACE4_INHERIT_ONLY_ACE             = 0x00000008;
const ACE4_SUCCESSFUL_ACCESS_ACE_FLAG   = 0x00000010;
const ACE4_FAILED_ACCESS_ACE_FLAG       = 0x00000020;
const ACE4_IDENTIFIER_GROUP             = 0x00000040;
const ACE4_INHERITED_ACE                = 0x00000080;
```

Un serveur n'a pas besoin de prendre en charge ces fanions. Si le serveur prend en charge des fanions similaires, mais pas exactement les mêmes, à ces fanions, la mise en œuvre peut définir une transposition entre les fanions définis par le protocole et les fanions définis par la mise en œuvre.

Par exemple, supposons qu'un client essaye d'établir une ACE avec ACE4_FILE_INHERIT_ACE établi mais pas ACE4_DIRECTORY_INHERIT_ACE. Si le serveur ne prend pas en charge une forme d'héritage d'ACL, le serveur devrait rejeter la demande avec NFS4ERR_ATTRNOTSUPP. Si le serveur ne prend pas en charge un seul fanion "inherit ACE" qui s'applique aux fichiers et répertoires, le serveur peut rejeter la demande (c'est-à-dire, exiger du client qu'il établisse les deux fanions d'héritage de fichier et de répertoire). Le serveur peut aussi accepter la demande et désactiver en silence le fanion ACE4_DIRECTORY_INHERIT_ACE.

6.2.1.4.1 Discussion des bits de fanion

ACE4_FILE_INHERIT_ACE
Tout fichier non répertoire dans tout sous répertoire va faire hériter cette ACE.

ACE4_DIRECTORY_INHERIT_ACE
Peut être placé dans un répertoire et indique que cette ACE devrait être ajoutée à chaque nouveau répertoire créé.
Si ce fanion est établi dans une ACE dans un attribut d'ACL à établir sur un objet de système de fichiers non répertoire, l'opération qui tente d'établir l'ACL DEVRAIT échouer avec NFS4ERR_ATTRNOTSUPP.

ACE4_NO_PROPAGATE_INHERIT_ACE
Peut être placé dans un répertoire. Ce fanion dit au serveur que l'héritage de cette ACE devrait s'arrêter à tout répertoire fils nouvellement créé.

ACE4_INHERIT_ONLY_ACE
Peut être placé dans un répertoire mais ne s'applique pas au répertoire ; les ACE ALLOW et DENY avec ce bit établi n'affectent pas l'accès au répertoire, et les ACE AUDIT et ALARM avec ce bit établi ne déclenchent pas d'événement d'enregistrement ou d'alarme. De telles ACE ne prennent effet qu'une fois qu'elles sont appliquées (avec ce bit remis à zéro) aux nouveaux fichiers et répertoires créés comme spécifié par les fanions ACE4_FILE_INHERIT_ACE et ACE4_DIRECTORY_INHERIT_ACE. Si ce fanion est présent sur une ACE, mais si ni ACE4_DIRECTORY_INHERIT_ACE ni ACE4_FILE_INHERIT_ACE n'est présent, alors une opération qui tente d'établir un tel attribut DEVRAIT échouer avec NFS4ERR_ATTRNOTSUPP.

ACE4_SUCCESSFUL_ACCESS_ACE_FLAG et ACE4_FAILED_ACCESS_ACE_FLAG
Les bits fanions ACE4_SUCCESSFUL_ACCESS_ACE_FLAG (SUCCESS) et ACE4_FAILED_ACCESS_ACE_FLAG (FAILED) ne peuvent être établis que sur les types d'ACE ACE4_SYSTEM_AUDIT_ACE_TYPE (AUDIT) et ACE4_SYSTEM_ALARM_ACE_TYPE (ALARM). Si durant le traitement de l'ACL du fichier, le serveur rencontre une ACE AUDIT ou ALARM correspondant au principal qui tente le OPEN, le serveur note ce fait, et la présence, si il en est, des fanions SUCCESS et FAILED rencontrés dans l'ACE AUDIT ou ALARM. Une fois que le serveur a achevé le traitement de l'ACL, il note alors si l'opération a réussi ou échoué. Si l'opération a réussi, et si le fanion SUCCESS était établi pour une ACE AUDIT ou ALARM correspondante, alors l'événement AUDIT ou ALARM approprié s'est produit. Si l'opération a échoué, et si le fanion FAILED était établi pour l'ACE AUDIT ou ALARM correspondante, alors l'événement AUDIT ou ALARM approprié s'est produit. L'un ou l'autre ou les deux de SUCCESS ou FAILED peut être établi, mais si ni l'un ni l'autre n'est établi, l'ACE AUDIT ou ALARM n'est pas utile. Le traitement décrit précédemment s'applique aux opérations ACCESS même quand elles retournent NFS4_OK. Pour les besoins de AUDIT et ALARM, on considère qu'une opération ACCESS est un "échec" si elle échoue à retourner un bit qui était demandé et pris en charge.

ACE4_IDENTIFIER_GROUP

Indique que le "who" se réfère à un GROUP comme défini dans UNIX ou à un GROUP ACCOUNT comme défini dans Windows. les clients et serveurs DOIVENT ignorer le fanion ACE4_IDENTIFIER_GROUP sur les ACE avec une valeur de who égale à un des identifiants spéciaux mentionnés au paragraphe 6.2.1.5.

ACE4_INHERITED_ACE
Indique que cette ACE est héritée d'un répertoire parent. Un serveur qui prend en charge l'héritage automatique va placer ce fanion sur toutes les ACE héritées du répertoire parent quand il crée un nouvel objet. Les applications de client vont utiliser cela pour effectuer l'héritage automatique. Les clients et serveurs DOIVENT mettre ce bit à zéro dans l'attribut acl ; il peut seulement être utilisé dans les attributs dacl et sacl.

6.2.1.5 Who d'ACE

Le champ "who" d'une ACE est un identifiant qui spécifie le ou les principaux à qui l'ACE s'applique. Il peut se référer à un utilisateur ou à un groupe, avec le bit fanion ACE4_IDENTIFIER_GROUP qui spécifie lequel.

Il y a plusieurs identifiants spéciaux qui doivent être compris universellement, plutôt que dans le contexte d'un domaine DNS particulier. Certains de ces identifiants ne peuvent pas être compris quand un client NFS accède au serveur, mais ont une signification quand un processus local accède au fichier. La capacité d'afficher et modifier ces permissions est permise sur NFS, même si aucune des méthodes d'accès sur le serveur ne comprend les identifiants.

Who	Description
OWNER	Le propriétaire du fichier.
GROUP	Le groupe associé au fichier.
EVERYONE	Tout le monde, y compris le propriétaire et le groupe propriétaire.
INTERACTIVE	Accès à partir d'un terminal interactif.
NETWORK	Accès via le réseau.
DIALUP	Accès via un utilisateur qui fait le numéro du serveur.
BATCH	Accès à partir d'une tâche en lots.
ANONYMOUS	Accès sans aucune authentification.
AUTHENTICATED	Tout utilisateur authentifié (par opposition à ANONYMOUS).
SERVICE	Accès à partir d'un service système.

Tableau 7

Pour éviter les conflits, ces identifants spéciaux sont distingués par un "@" ajouté et devraient apparaître sous la forme "xxxx@" (sans nom de domaine après le "@") par exemple, ANONYMOUS@.

Le fanion ACE4_IDENTIFIER_GROUP DOIT être ignoré sur les entrées avec ces identifiants spéciaux. Quand on code des entrées avec ces identifiants spéciaux, le fanion ACE4_IDENTIFIER_GROUP DEVRAIT être réglé à zéro.

6.2.1.5.1 Discussion de EVERYONE@

Il est important de noter que "EVERYONE@" n'est pas équivalent de l'entité UNIX "autre". C'est parce que, par définition, un "autre" UNIX n'inclut pas le propriétaire ou le groupe propriétaire d'un fichier. "EVERYONE@" signifie littéralement tout le monde, y compris le propriétaire ou groupe propriétaire.

6.2.2 Attribut 58 : dacl

L'attribut dacl est comme l'attribut acl, mais dacl permet juste les ACE ALLOW et DENY. L'attribut dacl prend en charge l'héritage automatique (voir le paragraphe 6.4.3.2).

6.2.3 Attribut 59 : sacl

L'attribut sacl est comme l'attribut acl, mais sacl permet juste les ACE AUDIT et ALARM. L'attribut sacl prend en charge l'héritage automatique (voir le paragraphe 6.4.3.2).

6.2.4 Attribut 33 : mode

L'attribut mode de NFSv4.1 se fonde sur les bits mode de UNIX. Les bits suivants sont définis :

```
const MODE4_SUID = 0x800;       /* établit l'identifiant d'utilisateur à l'exécution */
const MODE4_SGID = 0x400;       /* établit l'identifiant de groupe à l'exécution */
const MODE4_SVTX = 0x200;       /* sauvegarde le texte même après usage */
const MODE4_RUSR = 0x100;       /* permission de lecture : propriétaire */
const MODE4_WUSR = 0x080;       /* permission d'écriture : propriétaire */
const MODE4_XUSR = 0x040;       /* permission d'exécuter : propriétaire */
const MODE4_RGRP = 0x020;       /* permission de lecture : groupe */
const MODE4_WGRP = 0x010;       /* permission d'écriture : groupe */
const MODE4_XGRP = 0x008;       /* permission d'exécuter : groupe */
const MODE4_ROTH = 0x004;       /* permission de lecture : autre */
const MODE4_WOTH = 0x002;       /* permission d'écriture : autre */
const MODE4_XOTH = 0x001;       /* permission d'exécuter : autre */
```

Les bits MODE4_RUSR, MODE4_WUSR, et MODE4_XUSR s'appliquent au principal identifié dans l'attribut owner. Les bits MODE4_RGRP, MODE4_WGRP, et MODE4_XGRP s'appliquent aux principaux identifiés dans l'attribut owner_group mais qui ne sont pas identifiés dans l'attribut owner. Les bits MODE4_ROTH, MODE4_WOTH, et MODE4_XOTH s'appliquent à tout principal qui ne correspond pas à celui de l'attribut owner et n'a pas un groupe qui correspond à celui de l'attribut owner_group.

Les bits dans un mode autre que ceux spécifiés ci-dessus ne sont pas définis dans le présent protocole. Un serveur NE DOIT PAS retourner des bits autres que ceux définis ci-dessus dans une opération GETATTR ou READDIR, et il DOIT retourner NFS4ERR_INVAL si des bits autres que ceux définis ci-dessus sont établis dans une opération SETATTR, CREATE, OPEN, VERIFY, ou NVERIFY.

6.2.5 Attribut 74 : mode_set_masked

L'attribut mode_set_masked est un attribut en écriture seule qui permet que des bits individuels dans l'attribut mode soient établis ou réinitialisés, sans en changer d'autres. Il permet, par exemple, que les bits MODE4_SUID, MODE4_SGID, et MODE4_SVTX soient modifiés tout en laissant inchangés tous bits parmi les neuf de mode de moindre poids dédiés aux permissions.

Dans les instances où les neuf bits de moindre poids sont laissés inchangés, alors ni l'attribut acl ni l'attribut dacl ne devraient être automatiquement modifiés comme discuté au paragraphe 6.4.1.

L'attribut mode_set_masked consiste en deux mots, chacun sous la forme mode4. Le premier consiste en la valeur à appliquer à la valeur de mode courante et la seconde est un gabarit. Seul les bits réglés à un dans le mot de gabarit sont changés (établis ou remis à un) dans le mode du fichier. Tous les autres bits dans le mode restent inchangés. Les bits dans le premier mot qui correspondent aux bits qui sont à zéro dans le gabarit sont ignorés, sauf les bits indéfinis dont la validité est vérifiée et peut résulter en une erreur NFS4ERR_INVAL comme décrit ci-dessous.

L'attribut mode_set_masked est seulement valide dans une opération SETATTR. Si il est utilisé dans une opération CREATE ou OPEN, le serveur DOIT retourner NFS4ERR_INVAL.

Les bits non définis comme valides dans l'attribut mode ne sont pas non plus valides dans le mot de l'attribut mode_set_masked. Le serveur DOIT retourner NFS4ERR_INVAL si un de ces bits est réglé à un dans un SETATTR. Si le mode et les attributs mode_set_masked sont tous deux spécifiés dans le même SETATTR, le serveur DOIT aussi retourner NFS4ERR_INVAL.

6.3 Méthodes courantes

Les exigences de ce paragraphe sont référencées dans les paragraphes suivants, en particulier au paragraphe 6.4.

6.3.1 Interprétation d'une ACL

6.3.1.1 Considérations de serveur

Le serveur utilise l'algorithme décrit au paragraphe 6.2.1 pour déterminer si une ACL permet l'accès à un objet. Cependant, l'ACL pourrait n'être pas la seule pour déterminer l'accès. Par exemple :

* Dans le cas d'un système de fichiers exporté en lecture seule, le serveur peut refuser l'accès en écriture même si une ACL d'un objet l'accorde.

* Les mises en œuvre de serveur PEUVENT accorder les permissions ACE4_WRITE_ACL et ACE4_READ_ACL pour empêcher que survienne une situation dans laquelle il a une façon non valide de modifier l'ACL.

* Tous les serveurs vont permettre à un utilisateur la capacité de lire les données du fichier quand seule la permission d'exécuter est accordée (c'est-à-dire, si l'ACL refuse à l'utilisateur l'accès ACE4_READ_DATA et lui permet ACE4_EXECUTE, le serveur va permettre à l'utilisateur de lire les données du fichier).

* De nombreux serveurs ont la notion d'outrepassement du propriétaire dans laquelle il est permis au propriétaire de l'objet d'outrepasser les accès qui sont refusés par l'ACL. Cela peut être utile, par exemple, pour permettre aux utilisateurs de continuer d'accéder aux fichiers ouverts sur lesquels les permissions ont changé.

* De nombreux serveurs ont la notion de "super utilisateur" qui a des privilèges supérieurs à un utilisateur ordinaire. Un super utilisateur peut être capable de lire ou écrire des données ou métadonnées de façons qui ne seraient pas permises par l'ACL.

* Un attribut de rétention pourrait aussi bloquer un accès par ailleurs permis par les ACL (voir le paragraphe 5.13).

6.3.1.2 Considérations de client

Les clients NE DEVRAIENT PAS faire leurs propres vérifications d'accès sur la base de leur interprétation de l'ACL, mais plutôt utiliser les opérations OPEN et ACCESS pour faire les vérifications d'accès. Cela permet au client d'agir sur le résultat de la détermination par le serveur de si l'accès devrait ou non être accordé sur la base de son interprétation de l'ACL.

Les clients doivent avoir connaissance de situations dans lesquelles l'ACL d'un objet va définir un certain accès même si le serveur ne l'applique pas. En général, mais en particulier dans ces situations, le client a besoin de faire sa part dans l'application de l'accès comme défini par l'ACL. Pour ce faire, le client PEUT envoyer l'opération ACCESS appropriée avant de servir la demande de l'utilisateur ou de l'application afin de déterminer si l'utilisateur ou l'application devrait se voir accorder l'accès demandé. Pour des exemples dans lesquels l'ACL peut définir des accès que le serveur n'applique pas, voir le paragraphe 6.3.1.1.

6.3.2 Calcul d'un attribut de mode provenant d'une ACL

La méthode suivante peut être utilisée pour calculer les bits MODE4_R*, MODE4_W*, et MODE4_X* d'un attribut de mode, sur la base d'une ACL.

D'abord, pour chacun des identifiants spéciaux OWNER@, GROUP@, et EVERYONE@, évaluer l'ACL dans l'ordre, en considérant seulement les ACE ALLOW et DENY pour l'identifiant EVERYONE@ et pour l'identifiant considéré. Le résultat de l'évaluation va être un gabarit d'ACL NFSv4 montrant exactement quels bits sont permis à cet identifiant.

Puis traduire le gabarit calculé pour OWNER@, GROUP@, et EVEONE@ dans les bits de mode pour, respectivement, l'utilisateur, les groupes, et les autres, comme suit :

1. Établir le bit read (MODE4_RUSR, MODE4_RGRP, ou MODE4_ROTH) si et seulement si ACE4_READ_DATA est établi dans le gabarit correspondant.
2. Établir le bit écrire (MODE4_WUSR, MODE4_WGRP, ou MODE4_WOTH) si et seulement si ACE4_WRITE_DATA et ACE4_APPEND_DATA sont tous les deux établis dans le gabarit correspondant.
3. Établir le bit execute (MODE4_XUSR, MODE4_XGRP, ou MODE4_XOTH) si et seulement si ACE4_EXECUTE est établi dans le gabarit correspondant.

6.3.2.1 Discussion

Certaines mises en œuvre de serveur ajoutent aussi des bits permis aux utilisateur et groupes nommés aux bits de groupe (MODE4_RGRP, MODE4_WGRP, et MODE4_XGRP).

Il est déconseillé aux mises en œuvre de faire cela, parce que il a été trouvé que cela cause de la confusion pour les utilisateurs qui voient refuser l'accès aux membres d'un groupe d'un fichier que les bits de mode paraissent permettre. (La présence des ACE DENY peut aussi conduire à un tel comportement, mais les ACE DENY sont supposés être utilisés plus rarement.)

La même confusion de l'utilisateur vue quand aller chercher le mode résulte aussi si l'établissement du mode ne contrôle pas effectivement les permissions pour le propriétaire, le groupe, et les autres utilisateurs ; cela motive les exigences qui suivent.

6.4 Exigences

Le serveur qui prend en charge à la fois mode et ACL doit veiller à synchroniser les bits MODE4_*USR, MODE4_*GRP, et MODE4_*OTH avec les ACE qui ont les champs who respectifs de "OWNER@", "GROUP@", et "EVERYONE@". De cette façon, le client peut voir si des permissions d'accès sémantiquement équivalentes existent si le client demande les attributs owner, owner_group, et mode ou juste l'ACL.

Dans ce paragraphe, beaucoup est pris des méthodes du paragraphe 6.3.2. De nombreuses exigences se réfèrent à ce paragraphe. Mais noter que les méthodes ont les comportements spécifiés avec des "DEVRAIT". C'est intentionnel, pour éviter d'invalider les mises en œuvre existantes qui calculent le mode en accord avec le projet retiré d'ACL POSIX (projet 17 de 1003.1e) plutôt que par les permissions réelles sur owner, group, et autre.

6.4.1 Réglage des attributs Mode et/ou ACL

Dans le cas où un serveur prend en charge l'attribut sacl ou dacl, en plus de l'attribut acl, le serveur DOIT faire échouer une demande d'établir l'attribut acl simultanément avec un attribut dacl ou sacl. L'erreur à donner est NFS4ERR_ATTRNOTSUPP.

6.4.1.1 Établissement de Mode et pas de ACL

Quand un des neuf bits de mode de moindre poids est sujet à changer, soit parce que l'attribut mode a été établi, soit parce que l'attribut mode_set_masked a été établi et que le gabarit incluait un ou plusieurs bits des neuf bits de mode de moindre poids, et qu'aucun attributACL n'est explicitement établi, les attributs acl et dacl doivent être modifiés en accord avec la valeur mise à jour de ces bits. Cela doit arriver même si la valeur des bits de moindre poids est la même après que mode est établi comme avant.

Noter qu'aucune ACE AUDIT ou ALARM (donc toutes ACE dans l'attribut sacl) n'est affectée par des changements de mode.

Dans les cas où les bits de permissions sont sujets à changement, les attributs acl et dacl DOIVENT être modifiés de telle façon que le mode calculé via la méthode du paragraphe 6.3.2 donne les neuf bits de moindre poids (MODE4_R*, MODE4_W*, MODE4_X*) de l'attribut mode comme modifié par le changement d'attribut. Les attributs ACL DEVRAIENT aussi être modifiés de telle façon que :

1. Si MODE4_RGRP n'est pas établi, ACE4_READ_DATA NE DEVRAIT PAS être accordé aux entités explicitement mentionnées dans l'ACL autres que OWNER@ et EVERYONE@.

2. Si MODE4_WGRP n'est pas établi, ACE4_WRITE_DATA ou ACE4_APPEND_DATA NE DEVRAIT PAS être accordé aux entités explicitement mentionnées dans l'ACL autres que OWNER@ et EVERYONE@.

3. Si MODE4_XGRP n'est pas établi, ACE4_EXECUTE NE DEVRAIT PAS être accordé aux entités explicitement mentionnées dans l'ACL autres que OWNER@ et EVERYONE@.

Les bits de gabarit d'accès autres que ceux mentionnés ci-dessus, apparaissant dans les ACE ALLOW, PEUVENT aussi être désactivés.

Noter que les ACE avec le fanion ACE4_INHERIT_ONLY_ACE établi n'affectent pas les permissions de l'ACL elle-même, ni les ACE de type AUDIT et ALARM. À ce titre, il est souhaitable de laisser ces ACE non modifiées quand on modifie les attributs d'ACL.

Noter aussi que l'exigence peut être satisfaite en éliminant acl et dacl, en faveur d'une ACL qui représente le mode et seulement le mode. Ceci est permis, mais il est préférable qu'un serveur préserve autant de l'ACL que possible sans violer les exigences ci-dessus. Éliminer l'ACL rend effectivement impossible qu'un fichier créé avec un attribut mode hérite d'une ACL (voir le paragraphe 6.4.3).

6.4.1.2 Établissement de ACL et pas de mode

Quand on établit les attributs acl ou dacl et pas mode ou mode_set_masked, les bits de permission du mode doivent être déduits de l'ACL. Dans ce cas, l'attribut ACL DEVRAIT être établi comme donné. Les neuf bits de mondre poids de l'attribut mode (MODE4_R*, MODE4_W*, MODE4_X*) DOIVENT être modifiés pour correspondre au résultat de la méthode du paragraphe 6.3.2. Les trois bits de poids fort de mode (MODE4_SUID, MODE4_SGID, MODE4_SVTX) DEVRAIENT rester inchangés.

6.4.1.3 Établissement de ACL et de Mode

Quand on établit à la fois les deux attributs de mode (cela inclut l'utilisation de l'attribut mode ou de l'attribut mode_set_masked) et les attributs acl ou dacl dans la même opération, les attributs DOIVENT être appliqués dans cet ordre : mode (ou mode_set_masked) puis ACL. L'attribut relatif au mode est établi comme donné, puis l'attribut ACL est réglé comme donné, éventuellement en changeant le mode final, comme décrit au paragraphe 6.4.1.2.

6.4.2 Restitution des attributs Mode et/ou ACL

Ce paragraphe s'applique seulement aux serveurs qui prennent en charge les deux attributs mode et ACL.

Certaines mises en œuvre de serveur peuvent avoir un concept de "objets sans ACL", ce qui signifie que toutes les permissions sont accordées et refusées en accord avec l'attribut mode et qu'aucun attribut ACL n'est mémorisé pour cet objet. Si un attribut ACL est demandé à un tel serveur, le serveur DEVRAIT retourner une ACL qui n'est pas en conflit avec le mode ; c'est-à-dire, l'ACL retournée DEVRAIT représenter les neuf bits de moindre poids de l'attribut mode (MODE4_R*, MODE4_W*, MODE4_X*) comme décrit au paragraphe 6.3.2.

Pou d'autres mises en œuvre de serveur, l'attribut ACL est toujours présent pour chaque objet. Ces serveurs DEVRAIENT mémoriser au moins les trois bits de poids fort de l'attribut mode (MODE4_SUID, MODE4_SGID, MODE4_SVTX). Le serveur DEVRAIT retourner un attribut mode si il en est demandé un, et les neuf bits de moindre poids de mode (MODE4_R*, MODE4_W*, MODE4_X*) DOIVENT correspondre au résultat de l'application de la méthode du paragraphe 6.3.2 à l'attribut ACL.

6.4.3 Création de nouveaux objets

Si un serveur prend en charge tous les attributs ACL, il peut utiliser les attributs ACL sur le répertoire parent pour calculer un attribut ACL initial pour un objet nouvellement créé. Cela va être appelé l'ACL héritée dans ce paragraphe. L'acte d'ajout d'une ou plusieurs ACE à l'ACL héritée qui sont fondées sur les ACE dans l'ACL du répertoire parent va être appelé l'héritage d'une ACE dans ce paragraphe.

Les mises en œuvre devraient normaliser ce que le comportement de CREATE et OPEN doit être selon la présence ou l'absence des attributs mode et ACL.

1. Si juste le mode est donné dans l'appel : dans ce cas, l'héritage DEVRAIT avoir lieu, mais le mode DOIT être appliqué à l'ACL héritée comme décrit au paragraphe 6.4.1.1, modifiant ainsi l'ACL.

2. Si juste l'ACL est donnée dans l'appel : dans ce cas, l'héritage NE DEVRAIT PAS avoir lieu, et l'ACL définie dans le CREATE ou OPEN va être établie sans modification, et le mode modifié comme au paragraphe 6.4.1.2.

3. Si mode et ACL sont donnés tous les deux dans l'appel : dans ce cas, l'héritage NE DEVRAIT PAS avoir lieu, et les deux attributs vont être réglés comme décrit au paragraphe 6.4.1.3.

4. Si ni mode ni ACL ne sont donnés dans l'appel : dans le cas où un objet est créé sans attribut initial du tout, par

exemple, une opération OPEN avec un opentype4 de OPEN4_CREATE et un createmode4 de EXCLUSIVE4, l'héritage NE DEVRAIT PAS avoir lieu (noter que EXCLUSIVE4_1 est un meilleur choix de createmode4, car il permet des attributs initiaux). À la place, le serveur DEVRAIT établir des permissions de refuser tout accès à l'objet nouvellement créé. On s'attend à ce que le client approprié établisse les attributs désirés dans une opération SETATTR ultérieure, et le serveur DEVRAIT permettre que cette opération réussisse, sans considération des permissions avec lesquelles l'objet est créé. Par exemple, une ACL vide refuse toutes les permissions, mais le serveur devrait permettre que le SETATTR du propriétaire réussisse même si WRITE_ACL est implicitement refusé.

Dans d'autres cas, l'héritage DEVRAIT avoir lieu, et aucune modification de l'ACL ne va se produire. L'attribut mode, si il est pris en charge, DOIT être comme calculé au paragraphe 6.3.2, avec les bits MODE4_SUID, MODE4_SGID, et MODE4_SVTX à zéro. Si aucune ACE héritable n'existe sur le répertoire parent, les règles pour créer les attributs acl, dacl, ou sacl sont définies par la mise en œuvre. Si l'attribut dacl ou sacl est pris en charge, alors le fanion ACL4_DEFAULTED DEVRAIT être établi sur les nouveaux attributs créés.

6.4.3.1 ACL héritée

Si l'objet créé n'est pas un répertoire, l'ACL héritée NE DEVRAIT PAS hériter des ACE provenant de l'ACL du répertoire parent sauf si le fanion ACE4_FILE_INHERIT_FLAG est établi.

Si l'objet créé est un répertoire, l'ACL héritée devrait hériter de toutes les ACE héritables provenant du répertoire parent, c'est-à-dire, celles qui ont le fanion ACE4_FILE_INHERIT_ACE ou ACE4_DIRECTORY_INHERIT_ACE établi. Si l'ACE héritable a ACE4_FILE_INHERIT_ACE établi mais ACE4_DIRECTORY_INHERIT_ACE à zéro, l'ACE héritée sur le répertoire nouvellement créé DOIT avoir le fanion ACE4_INHERIT_ONLY_ACE établi pour empêcher que le répertoire soit affecté par des ACE destinées à des non répertoires.

Quand un nouveau répertoire est créé, le serveur PEUT partager toute ACE héritée qui est à la fois héritable et effective (en d'autres termes, qui n'a ni ACE4_INHERIT_ONLY_ACE ni ACE4_NO_PROPAGATE_INHERIT_ACE établis) en deux ACE, une sans fanion d'héritage et une avec ACE4_INHERIT_ONLY_ACE établi. (Dans le cas d'un attribut dacl ou sacl, ces deux ACE DEVRAIENT aussi avoir le fanion ACE4_INHERITED_ACE établi.) Cela rend plus simple de modifier les permissions effectives sur le répertoire sans modifier l'ACE qui va être héritée sur les enfants du nouveau répertoire.

6.4.3.2 Héritage automatique

L'attribut acl consiste seulement en un dispositif d'ACE, mais les attributs sacl (paragraphe 6.2.3) et dacl (paragraphe 6.2.2) incluent aussi un champ fanion additionnel.

```
struct nfsacl41 {
      aclflag4       na41_flag ;
      nfsace4        na41_aces<> ;
} ;
```

Le champ Fanion s'applique au sacl ou dacl entier ; trois valeurs de fanion sont définies :

```
const ACL4_AUTO_INHERIT     = 0x00000001 ;
const ACL4_PROTECTED        = 0x00000002 ;
const ACL4_DEFAULTED        = 0x00000004 ;
```

et tous les autres bits doivent être mis à zéro. Le fanion ACE4_INHERITED_ACE peut être établi dans les ACE de sacl ou dacl (tandis qu'il doit toujours être à zéro dans acl).

Ensemble, ces caractéristiques permettent à un serveur de prendre en charge l'héritage automatique, qu'on explique maintenant plus en détails.

Les ACE héritables sont normalement héritées par les objets fils seulement au moment où les objets fils sont créés ; les modifications ultérieures aux ACE héritables ne résultent pas en modifications aux ACE héritées sur les descendants.

Cependant, dacl et sacl fournissent un mécanisme FACULTATIF qui permet à une application cliente de propager les changements aux ACE héritables à une hiérarchie entière de répertoires.

Un serveur qui prend cela en charge effectue l'héritage à la création de l'objet de la façon normale, et DEVRAIT établir le fanion ACE4_INHERITED_ACE sur toute ACE héritée lorsque elle est ajoutée au nouvel objet.

Une application cliente comme un éditeur d'ACL peut alors propager les changements aux ACE héritables sur un répertoire en traversant de façon récurrente les descendants de ce répertoire et en modifiant chaque ACL rencontrée pour supprimer toutes les ACE avec le fanion ACE4_INHERITED_ACE et pour les remplacer par les nouvelles ACE héritables (elles aussi avec le fanion ACE4_INHERITED_ACE établi). Elle utilise les fanions existants d'héritage d'ACE pour décider de façon évidente quelles ACE propager. (Noter qu'elle peut rencontrer d'autres ACE héritables quand elle descend la hiérarchie de répertoires et que celles-ci vont aussi devoir être prises en compte dans la propagation des ACE héritables aux autres descendants.)

La portée de cette propagation peut être limitée de deux façons : d'abord, l'héritage automatique n'est pas effectué à partir de toute ACL de répertoire qui a le fanion ACL4_AUTO_INHERIT à zéro, et ensuite, l'héritage automatique s'arrête partout où une ACL avec le fanion ACL4_PROTECTED est établi, empêchant la modification de cette ACL et aussi (si l'ACL est établie sur un répertoire) de l'ACL sur tous les descendants de l'objet.

Cette propagation est effectuée indépendamment pour les attributs sacl et dacl ; donc, les fanions ACL4_AUTO_INHERIT et ACL4_PROTECTED peuvent être établis indépendamment pour le sacl et le dacl, et la propagation d'un type d'acl peut continuer à descendre une hiérarchie même lorsque la propagation de l'autre acl s'est arrêtée.

Les nouveaux objets devraient être créés avec une dacl et une sacl qui ont toutes les deux le fanion ACL4_PROTECTED à zéro et le fanion ACL4_AUTO_INHERIT établi à la même valeur que celle, respectivement, sur la sacl ou dacl de l'objet parent.

Les deux attributs dacl et sacl sont RECOMMANDÉS, et un serveur peut prendre en charge l'un sans prendre en charge l'autre.

Un serveur qui prend en charge à la fois le vieil attribut acl et un ou les deux des nouveaux attributs dacl ou sacl doit le faire d'une façon telle que les trois attributs restent cohérents les uns avec les autres. Donc, les ACE rapportées dans l'attribut acl devraient être l'union des ACE rapportées dans les attributs dacl et sacl, sauf que le fanion ACE4_INHERITED_ACE provenant des ACE dans l'acl doit être à zéro. Et bien sûr, un client qui interroge seulement l'acl va être incapable de déterminer les valeurs des champs de fanion de sacl ou dacl.

Quand un client effectue un SETATTR pour l'attribut acl, le serveur DEVRAIT régler le fanion ACL4_PROTECTED à vrai sur le sacl et le dacl. En utilisant l'attribut acl, par opposition aux attributs dacl ou sacl, le client signale qu'il ne peut pas comprendre l'héritage automatique, et donc ne peut pas garantir d'établir une ACL pour laquelle l'héritage automatique aurait un sens.

Quand une application cliente interroge une ACL, la modifie, et l'établit à nouveau, il devrait laisser inchangées toutes les ACE marquées avec ACE4_INHERITED_ACE, dans leur ordre original, à la fin de l'ACL. Si l'application est incapable de faire cela, il devrait établir le fanion ACL4_PROTECTED. Ce comportement n'est pas appliqué par les serveurs, mais la violation de cette règle peut conduire à des résultats inattendus quand les applications effectuent l'héritage automatique.

Si un serveur prend aussi en charge l'attribut mode, il DEVRAIT régler le mode d'une façon telle que les ACE héritées soient inchangées, dans leur ordre original, à la fin de l'ACL. Si il n'est pas capable de le faire, il DEVRAIT établir le fanion ACL4_PROTECTED sur le dacl du fichier.

Finalement, dans le cas où la demande qui crée un nouveau fichier ou répertoire n'établit pas aussi les permissions pour ce fichier ou répertoire, et il n'y a pas aussi des ACE à hériter du répertoire du parent, alors le choix du serveur de l'ACL pour le nouvel objet dépend de la mise en œuvre. Dans ce cas, le serveur DEVRAIT établir le fanion ACL4_DEFAULTED sur l'ACL qu'il choisit pour le nouvel objet. Une application qui effectue l'héritage automatique prend le fanion ACL4_DEFAULTED comme le signe que l'ACL devrait être complètement remplacée par celle générée en utilisant les règles de l'héritage automatique.

7. Espace de noms d'un seul serveur

Cette section décrit l'espace de noms NFSv4 d'un seul serveur. Les espaces de noms d'un seul serveur peuvent être présentés directement aux clients, ou ils peuvent être utilisés comme base pour former de plus grands espaces de noms multi serveurs (par exemple, au niveau du site ou de l'organisation) pour être présentés aux clients, comme décrit à la Section 11.

7.1 Exportations de serveur

Sur un serveur UNIX, l'espace de noms décrit tous les fichiers accessibles par des noms de chemin sous le répertoire racine ou "/". Sur un serveur Windows, l'espace de noms constitue tous les fichiers sur des disques nommés par des lettres de disque transposées. Les administrateurs de serveur NFS rendent rarement disponible la totalité de l'espace de noms de système de fichiers du serveur aux clients NFS. Plus souvent, des portions de l'espace de noms sont rendues disponibles via une caractéristique "export". Dans les précédentes versions du protocole NFS, la bride de fichier racine pour chaque export est obtenue du protocole MOUNT ; le client envoie une chaîne qui identifie le noms de l'exportation au sein de l'espace de noms et le serveur retourne la bride de fichier racine pour cette exportation. Le protocole MOUNT fournit aussi une procédure EXPORTS qui énumère les exportations du serveur.

7.2 Navigation dans les exportations

Le protocole NFSv4.1 fournit une bride de fichier racine que les clients peuvent utiliser pour obtenir les brides de fichier pour les exportations d'un serveur particulier, via une série d'opérations LOOKUP au sein d'un COMPOUND, pour traverser un chemin. Une pratique courante d'utilisateur est d'utiliser une interface d'utilisateur graphique (peut-être une fenêtre de dialogue de fichier "Open") pour trouver un fichier via une progression à travers une arborescence de répertoire. Le client doit être capable de se déplacer d'une exportation à une autre via des opérations LOOKUP progressives à un seul composant.

Ce style de navigation n'est pas bien pris en charge par le protocole NFSv3. Dans NFSv3, le client s'attend à ce que toutes les opérations LOOKUP restent dans un seul système de fichiers de serveur. Par exemple, l'attribut device ne va pas changer. Cela empêche un client de prendre les chemins d'espace de noms que cette portée exporte.

Dans le cas de NFSv3, un auto monteur sur le client peut obtenir une photographie de l'espace de noms du serveur en utilisant la procédure EXPORTS du protocole MOUNT. Si il comprend la syntaxe de nom de chemin du serveur, il peut créer une image de l'espace de noms du serveur sur le client. Les parties de l'espace de noms qui ne sont pas exportées par le serveur sont remplies avec des répertoires qui pourraient être construits de façon similaire à un "pseudo système de fichiers" (voir le paragraphe 7.3) NFSv4.1 qui permet à l'utilisateur de naviguer d'un système de fichiers monté à un autre. Il y a un inconvénient à cette représentation de l'espace de noms du serveur sur le client : il est statique. Si l'administrateur du serveur ajoute une nouvelle exportation, le client ne va pas le savoir.

7.3 Système de pseudo fichiers de serveur

Les serveurs NFSv4.1 évitent cette incohérence de l'espace de noms en présentant toutes les exportations pour un serveur donné dans le cadre d'un seul espace de noms pour ce serveur. Un client NFSv4.1 utilise les opérations LOOKUP et READDIR pour naviguer sans interruption d'une exportation à une autre.

Lorsque il y a des portions de l'espace de noms du serveur qui ne sont pas exportées, les clients ont besoin d'un moyen de traverser ces portions pour atteindre les systèmes de fichiers réellement exportés. Une technique que les serveurs peuvent utiliser pour fournir cela est de ponter la portion non exportée de l'espace de noms via un "pseudo système de fichiers" qui fournit une vue des seuls répertoires exportés. Un pseudo système de fichiers a un unique fsid et se comporte comme un système de fichiers normal, en lecture seule.

Sur la base de la construction de l'espace de noms du serveur, il est possible que plusieurs pseudo systèmes de fichiers existent. Par exemple,

```
/a          pseudo système de fichiers
/a/b         système de fichiers réel
/a/b/c       pseudo système de fichiers
/a/b/c/d     système de fichiers réel
```

Chacun des pseudo systèmes de fichiers est considéré comme une entité séparée et donc DOIT avoir son propre fsid, unique parmi tous les fsid pour ce serveur.

7.4 Racines multiples

Certains environnements d'exploitation sont parfois décrits comme ayant "plusieurs racines". Dans de tels environnements, les systèmes de fichiers individuels sont couramment représentés par des noms de disque ou de volume. Les serveurs

NFSv4 pour ces plates-formes peuvent construire un pseudo système de fichiers par dessus ces noms de racines afin que les lettres de disques ou les noms de volumes soient simplement des noms de répertoire dans la pseudo racine.

7.5 Volatilité des brides de fichier

La nature du pseudo système de fichiers du serveur est qu'il est une représentation logique du ou des systèmes de fichiers disponibles sur le serveur. Donc, le pseudo système de fichiers est très probablement construit dynamiquement quand le serveur est instancié pour la première fois. On s'attend à ce que le pseudo système de fichiers ne puisse pas avoir une contrepartie sur le disque à partir de laquelle des brides de fichier persistantes pourraient être construites. Bien qu'il soit préférable que le serveur fournisse des brides de fichier persistantes pour le pseudo système de fichiers, le client NFS devrait s'attendre à ce que les brides de fichier de pseudo système de fichiers soient volatiles. Cela peut être confirmé en vérifiant l'attribut "fh_expire_type" associé pour les brides de fichier en question. Si les brides de fichier sont volatiles, le client NFS doit être prêt à récupérer une valeur de bride de fichier (par exemple, avec une série d'opérations LOOKUP) quand il reçoit une erreur de NFS4ERR_FHEXPIRED.

Parce qu'il est assez probable que les serveurs vont mettre en œuvre des pseudo systèmes de fichiers en utilisant des brides de fichier volatiles, les clients doivent être prêts pour elles, plutôt que de supposer que toutes les brides de fichier vont être persistantes.

7.6 Racine exportée

Si le système de fichiers racine du serveur est exporté, on pourrait conclure qu'un pseudo système de fichiers est inutile. Il n'en est pas nécessairement ainsi. Supposons le systèmes de fichiers suivant sur un serveur :

```
/      fs1  (exporté)
/a     fs2  (non exporté)
/a/b   fs3  (exporté)
```

Parce que fs2 n'est pas exporté, fs3 ne peut pas être atteint avec de simples LOOKUP. Le serveur doit ponter le trou avec un pseudo système de fichiers.

7.7 Croisement de point culminant

L'environnement du système de fichiers de serveur peut être construit d'une façon telle qu'un système de fichiers contienne un répertoire qui est "couvert" ou monté par un second système de fichiers. Par exemple :

```
/a/b        (système de fichiers 1)
/a/b/c/d    (système de fichiers 2)
```

Le pseudo système de fichiers pour ce serveur peut être construit de façon à ressembler à :

```
/           (bouche trou/non exporté)
/a/b        (système de fichiers 1)
/a/b/c/d    (système de fichiers 2)
```

Il est de la responsabilité du serveur de présenter un pseudo système de fichiers complet au client. Si le client envoie une demande LOOKUP pour le chemin /a/b/c/d, la réponse du serveur est la bride de fichier de la racine du système de fichiers /a/b/c/d. Dans les précédentes versions de NFS, le serveur aurait répondu avec la bride de fichier de répertoire /a/b/c/d dans le système de fichiers /a/b.

Le client NFS va être capable de déterminer si il traverse un point de montage de serveur par un changement de valeur de l'attribut "fsid".

7.8 Politique de sécurité et présentation d'espace de noms

Parce que les clients NFSv4 possèdent la capacité de changer les mécanismes de sécurité utilisés, après avoir déterminé ce qui est permis, en utilisant SECINFO et SECINFO_NONAME, le serveur NE DEVRAIT PAS présenter une vue différente de l'espace de noms sur la base du mécanisme de sécurité utilisé par un client. À la place, il devrait présenter une vue

cohérente et retourner NFS4ERR_WRONGSEC si une tentative est faite d'accéder aux données avec un mécanisme de sécurité inapproprié.

Si des considérations de sécurité rendent nécessaire de cacher l'existence d'un système de fichiers particulier, par opposition à toutes les données en son sein, le serveur peut appliquer une politique de sécurité de ressources partagées dans l'espace de noms du serveur aux composants des ancêtres de la ressource. Par exemple :

```
/                       (bouche-trou/non exporté)
/a/b                    (système de fichiers 1)
/a/b/MonProjetSecret    (système de fichiers 2)
```

Le répertoire /a/b/MonProjetSecret est un système de fichiers réel et est la ressource partagée. Supposons que la politique de sécurité pour /a/b/MonProjetSecret soit Kerberos avec intégrité et qu'il soit désiré de limiter la connaissance de l'existence de ce système de fichiers. Dans ce cas, le serveur devrait appliquer la même politique de sécurité à /a/b. Cela permet que la connaissance de l'existence d'un système de fichiers soit sécurisée quand c'est désirable.

Dans le cas de l'utilisation de plusieurs mécanismes de sécurité disjoints dans les ressources du serveur, appliquer cette sorte de politique résulterait en ce que le système de fichiers de niveau supérieur ne serait pas accessible en utilisant toute nuance de sécurité. Donc, cette sorte de configuration n'est pas compatible avec la dissimulation de l'existence (par opposition au contenu) pour des clients qui utilisent plusieurs ensembles disjoints de nuances de sécurité.

Dans d'autres circonstances, une politique désirable est pour la sécurité d'un objet particulier dans l'espace de noms du serveur pour inclure l'union de tous les mécanismes de sécurité de tous les descendants directs. Une pratique courante et convenable, sauf si de fortes exigences de sécurité en décident autrement, est de rendre le pseudo système de fichiers entier accessible par tous les mécanismes de sécurité valides.

Lorsque il y a des soucis sur la sécurité des données dans le réseau, les clients devraient utiliser de forts mécanismes de sécurité pour accéder au pseudo système de fichiers afin d'empêcher des attaques par interposition.

8. Gestion d'état

L'intégration du verrouillage dans le protocole NFS fait qu'il est nécessairement à états pleins. Avec l'inclusion de caractéristiques telles que les réservations partagées, les délégations de fichier et répertoire, les dispositions rappelables, et la prise en charge du verrouillage obligatoire de gamme d'octets, le protocole devient substantiellement plus dépendant d'une gestion appropriée de l'état que la combinaison traditionnelle de NFS et du gestionnaire de verrou de réseau (NLM, *Network Lock Manager*) [XNFS]. Ces caractéristiques incluent des facilités de verrouillage étendues, qui fournissent des mesures d'exclusion inter-clients, mais l'état offre aussi des caractéristiques non directement fournissables en utilisant un modèle sans état. Trois composants rendent cet état gérable :

* une claire division entre client et serveur,
* la capacité de détecter fiablement les incohérences d'état entre client et serveur,
* des mécanismes simples et robustes de récupération.

Dans ce modèle, le serveur possède les informations d'état. Le client demande des changements dans les verrous et le serveur répond avec les changements faits. Les changement non initiés par le client dans l'état de verrouillage sont peu fréquents. Le client reçoit une notification rapide de ces changements et peut ajuster sa vue de l'état de verrouillage pour refléter les changements du serveur.

Les éléments d'état individuels créés par le serveur et passés au client à sa demande sont représentés par des identifiants d'état *(stateid)* de 128 bits. Cet identifiant d'état peut représenter un fichier ouvert particulier, un ensemble de verrous de gamme d'octets détenus par un propriétaire particulier, ou une délégation rappelable de privilèges d'accès à un fichier d'une façon particulière ou à une localisation particulière.

Dans tous les cas, il y a une transition depuis les informations les plus générales qui représentent un client comme un tout jusqu'à l'identifiant d'état léger final utilisé pour la plupart des interactions de verrouillage de client et serveur. Les détails de cette transition vont varier avec le type d'objet mais elle commence toujours avec un identifiant de client.

8.1 Identifiants de client et de session

Un client doit établir un identifiant de client (voir le paragraphe 2.4) et alors un ou plusieurs identifiants de session *(sessionid)* (voir le paragraphe 2.10) avant d'effectuer toute opération d'ouverture, de verrou de gamme d'octets, de délégation, ou obtenir une disposition pour un objet de fichier. Chaque identifiant de session est associé à un identifiant de client spécifique, et donc sert d'abrégé de référence à un client NFSv4.1.

Pour certains types d'interactions de verrouillage, le client va représenter un certain nombre d'entités internes de verrouillage appelées des "propriétaires", qui correspondent normalement aux processus internes au client. Pour les autres types d'objets relatifs au verrouillage, comme les délégations et dispositions, aucune de ces entités intermédiaires n'est fournie, et les objets relatifs au verrouillage sont considérés être transférés directement entre le serveur et un client unitaire.

8.2 Définition de stateid

Quand le serveur accorde un verrou de n'importe quel type (incluant des ouvertures, verrous de gamme d'octets, délégations, et dispositions) il répond avec un unique identifiant d'état qui représente un ensemble de verrous (souvent un seul verrou) pour le même fichier, du même type, et partageant les mêmes caractéristiques de propriétaire. Donc, les ouvertures du même fichier par des propriétaires d'ouverture différents ont chacune un identifiant d'état qui les identifient. De même, chaque ensemble de verrous de gamme d'octets sur un fichier possédé par un propriétaire de verrou spécifique a son propre identifiant d'état pour l'identifier. Les délégations et dispositions ont aussi des identifiants d'état associés par lesquels ils peuvent être référencés. L'identifiant d'état est utilisé comme abrégé de référence à un verrou ou ensemble de verrous, et étant donné un identifiant d'état, le serveur peut déterminer le ou les propriétaires d'état associés (dans le cas d'une paire propriétaire d'ouverture/propriétaire de verrou) et la bride de fichier associée. Quand des identifiants d'état sont utilisés, la bride de fichier en cours doit être celle associée à cet identifiant d'état.

Tous les identifiants d'état associés à un identifiant de client donné sont associés à un prêt commun qui représente la prétention de ces identifiants d'état et les objets qu'ils représentent pour être maintenus par le serveur. Voir au paragraphe 8.3 la discussion du prêt.

Le serveur peut allouer des identifiants d'état indépendamment pour différents clients. Un identifiant d'état avec le même schéma de bits pour un client peut désigner un ensemble entièrement différent de verrous pour un client différent. L'identifiant d'état est toujours interprété par rapport à l'identifiant de client associé à la session en cours. Les identifiants d'état s'appliquent à toutes les sessions associées à l'identifiant de client donné, et le client peut utiliser un identifiant d'état obtenu d'une session sur une autre session associée au même identifiant de client.

8.2.1 Types de stateid

À l'exception des identifiants d'état spéciaux (voir le paragraphe 8.2.3) chaque identifiant d'état représente des objets de verrouillage d'un ensemble de types définis par le protocole NFSv4.1. Noter que dans tous ces cas, lorsque on parle de garantie, il est compris qu'il y a des situations comme un redémarrage de client, ou une révocation de verrou, qui permettent d'annuler la garantie.

* Les identifiants d'état peuvent représenter des ouvertures de fichiers : chaque identifiant d'état dans ce cas représente l'état OPEN pour un triplet donné {identifiant de client/propriétaire d'ouverture/bride de fichier}. De tels identifiants d'état sont sujets à changement (avec l'incrément de seqid de l'identifiant d'état qui en découle) en réponse aux OPEN qui résultent en des opérations de mise à niveau et de OPEN_DOWNGRADE.

* Les identifiants d'état peuvent représenter des ensembles de verrous de gamme d'octets : tous les verrous détenus sur un fichier particulier par un propriétaire particulier et contenus sous l'égide d'une ouverture de fichier particulière sont associés à un seul identifiant d'état avec le seqid qui est incrémenté chaque fois que des opérations LOCK et LOCKU affectent cet ensemble de verrous.

* Les identifiants d'état peuvent représenter des délégations de fichier, qui sont des garanties rappelables par le serveur au client que les autres clients ne vont pas référencer ou modifier un fichier particulier, jusqu'à ce que la délégation soit retournée. Dans NFSv4.1, les délégations de fichier peuvent être obtenues sur les fichiers réguliers et non réguliers. Un identifiant d'état représente une seule délégation détenue par un client pour une bride de fichier particulière.

* Les identifiants d'état peuvent représenter des délégations de répertoire, qui sont des garanties rappelables par le serveur au client que les autres clients ne vont pas modifier le répertoire, jusqu'à ce que la délégation soit retournée. Un identifiant d'état représente une seule délégation détenue par un client pour une bride de fichier particulière d'un

répertoire.

* Les identifiants d'état peuvent représenter des dispositions, qui sont des garanties rappelables par le serveur au client que des fichiers particuliers peuvent être accédés via un autre protocole d'accès aux données à des localisations spécifiques. Un tel accès est limité à des ensembles particuliers de gammes d'octets et peut se faire jusqu'à ce que ces gammes d'octets soient réduites ou que la disposition soit retournée.

Un identifiant d'état représente l'ensemble de toutes les dispositions détenues par un client particulier pour une bride de fichier particulière avec un type de disposition donné. Le seqid est mis à jour lorsque les dispositions de cet ensemble de gammes d'octets changent, via des opérations qui changent l'identifiant d'état de la disposition comme LAYOUTGET et LAYOUTRETURN.

8.2.2 Structure d'identifiant d'état

Les identifiants d'état sont divisés en deux champs, un champ de 96 bits "autre" qui identifie l'ensemble spécifique de verrous, et une valeur de séquence de 32 bits "seqid". Sauf dans le cas d'identifiant d'état spéciaux (voir le paragraphe 8.2.3) une valeur particulière du champ "autre" note un ensemble de verrous du même type (par exemple, des verrous de gamme d'octets, des ouvertures, délégations, ou dispositions) pour un fichier ou répertoire spécifique, et partageant les mêmes caractéristiques de propriétaire. Le seqid désigne une instance spécifique d'un tel ensemble de verrous, et est incrémenté pour indiquer les changements dans cet ensemble de verrous, soit par l'ajout soit par la suppression de verrous à l'ensemble, un changement dans la gamme d'octets auxquels ils s'appliquent, ou un relèvement ou une dégradation du type d'un ou plusieurs verrous.

Quand un tel ensemble de verrous est créé, le serveur retourne un identifiant d'état avec une valeur de seqid de un. Sur les opérations suivantes qui modifient l'ensemble de verrous, le serveur est obligé d'incrémenter de un le champ "seqid" chaque fois qu'il retourne un identifiant d'état pour la même combinaison de propriétaire d'état/fichier/type et qu'il y a un changement dans l'ensemble de verrous réellement désignés. Dans ce cas, le serveur va retourner un identifiant d'état avec un même champ "autre" que précédemment utilisé pour cette combinaison de propriétaire d'état/fichier/type, avec un champ "seqid" incrémenté. Ce schéma continue jusqu'à ce que le seqid soit incrémenté à NFS4_UINT32_MAX, et un (pas zéro) est la prochaine valeur de seqid.

L'objet de l'incrémentation du seqid est de permettre au serveur de communiquer au client l'ordre dans lequel les opérations qui ont modifié l'état de verrouillage associé à un identifiant d'état ont été traitées et rendre possible au client d'envoyer des demandes conditionnelles sur l'ensemble de verrous qui n'ont pas changé depuis que l'identifiant d'état en question a été retourné.

Sauf pour les identifiants d'état de disposition (paragraphe 12.5.3) quand un client envoie un identifiant d'état au serveur, il a un choix à l'égard du seqid envoyé. Il peut régler le seqid à zéro pour indiquer au serveur qu'il souhaite utiliser le seqid le plus à jour pour le champ "autre" de cet identifiant d'état. Cela serait le choix courant dans le cas d'un identifiant d'état envoyé avec une opération READ ou WRITE. Il aussi peut régler une valeur non zéro, auquel cas le serveur vérifie si le seqid est le bon. Dans ce cas, le serveur est obligé de retourner NFS4ERR_OLD_STATEID si le seqid est inférieur à la valeur courante et NFS4ERR_BAD_STATEID si le seqid est supérieur à la valeur courante. Cela serait le choix courant dans le cas d'identifiants d'état envoyés avec un CLOSE ou un OPEN_DOWNGRADE. Parce que les OPEN peuvent être envoyés en parallèle pour le même propriétaire, un client pourrait clore un fichier sans savoir qu'une mise à niveau de OPEN a été faite par le serveur, changeant le verrou en question. Si CLOSE a été envoyé avec un seqid de zéro, la mise à niveau de OPEN serait annulée avant que le client ait même reçu une indication qu'une mise à niveau était survenue.

Quand un identifiant d'état est envoyé par le serveur au client au titre d'une opération de rappel, il n'est pas soumis à vérification du seqid en cours et au retour d'un NFS4ERR_OLD_STATEID. C'est parce que le client n'est pas en position de connaître le seqid le plus à jour et donc ne peut pas le vérifier. Sauf mention spéciale, la valeur de seqid pour un identifiant d'état envoyé par le serveur au client au titre d'un rappel est obligée d'être zéro avec NFS4ERR_BAD_STATEID retourné si elle ne l'est pas.

Pour faire des comparaisons entre les seqid, à la fois par le client pour déterminer l'ordre des opérations et par le serveur pour déterminer si le NFS4ERR_OLD_STATEID est à retourner, la possibilité que le seqid soit revenu à zéro après la valeur de NFS4_UINT32_MAX doit être prise en compte. Quand deux valeurs de seqid sont comparées, le compte total de créneaux pour toutes les sessions associées au client en cours est utilisée pour le faire. Quand une valeur de seqid est moins que ce compte total de créneaux et qu'une autre valeur de seqid est supérieure à NFS4_UINT32_MAX moins le compte total de créneaux, la première doit être traitée comme inférieure à la dernière, en dépit du fait qu'elle est numériquement supérieure.

8.2.3 Stateid particuliers

Les valeurs d'identifiant d'état dont le champ "autre" est tout de zéros ou tout de uns sont réservées. Elles ne peuvent pas être allouées par le serveur mais ont une signification particulière définie par le protocole. La signification particulière dépend de si le champ "autre" est tout de zéros ou tout de uns et de la valeur spécifique du champ "seqid".

Les combinaisons suivantes de "autre" et de "seqid" sont définies dans NFSv4.1 :

* Quand "autre" et "seqid" sont tous les deux à zéro, l'identifiant d'état est traité comme un identifiant d'état anonyme spécial, qui peut être utilisé dans les demandes READ, WRITE, et SETATTR pour indiquer l'absence de tout état OPEN associé à la demande. Quand une valeur d'identifiant d'état anonyme est utilisée et qu'un open existant refuse la forme d'accès demandée, alors l'accès va être refusé à la demande. Cet identifiant d'état NE DOIT PAS être utilisé sur des opérations sur les serveurs de données (paragraphe 13.6).

* Quand "autre" et "seqid" sont tous les deux de uns, l'identifiant d'état est un outrepassement spécial de READ. Quand cette valeur est utilisée dans WRITE ou SETATTR, elle est traitée comme la valeur anonyme. Quand elle est utilisée dans READ, le serveur PEUT accorder l'accès, même si l'accès serait normalement refusé aux opérations READ. Cet identifiant d'état NE DOIT PAS être utilisé sur les opérations pour les serveurs de données.

* Quand "autre" est zéro et "seqid" est un, l'identifiant d'état représente l'identifiant d'état actuel, qui est toute valeur du dernier identifiant d'état retourné par une opération dans le COMPOUND. Dans le cas d'un OPEN, l'identifiant d'état retourné pour le fichier ouvert et non pour la délégation est utilisé. L'identifiant d'état passé à l'opération à la place de la valeur spéciale a sa valeur de "seqid" réglée à zéro, sauf quand l'identifiant d'état actuel est utilisé par l'opération CLOSE ou OPEN_DOWNGRADE. Si il n'y a pas d'opération dans le COMPOUND qui a retourné une valeur d'identifiant d'état, le serveur DOIT retourner l'erreur NFS4ERR_BAD_STATEID. Comme illustré à la Figure 6, si la valeur d'un identifiant d'état courant est un identifiant d'état spécial et si l'identifiant d'état des arguments d'une opération a "autre" réglé à zéro et "seqid" réglé à un, alors le serveur DOIT retourner l'erreur NFS4ERR_BAD_STATEID.

* Quand "autre" est zéro et "seqid" est NFS4_UINT32_MAX, l'identifiant d'état représente une valeur d'identifiant d'état réservée définie comme invalide. Quand cet identifiant d'état est utilisé, le serveur DOIT retourner l'erreur NFS4ERR_BAD_STATEID.

Si une valeur d'identifiant d'état utilisée est toute de zéros ou toute de uns dans le champ "autre" mais ne correspond pas à un des cas ci-dessus, le serveur DOIT retourner l'erreur NFS4ERR_BAD_STATEID.

Les identifiants d'état spéciaux, à la différence des autres identifiants d'état, ne sont pas associés à des identifiants de client ou brides de fichier individuels et peuvent être utilisés avec tous les identifiants de client et brides de fichier valides. Dans le cas d'un identifiant d'état spécial qui désigne l'identifiant d'état en cours, la valeur courante de l'identifiant d'état substituée à l'identifiant d'état spécial est associée à un identifiant de client et bride de fichier particulier, et donc, si il est utilisé lorsque la bride de fichier en cours ne correspond pas à celle associée à l'identifiant d'état en cours, l'opération à laquelle l'identifiant d'état est passé va retourner NFS4ERR_BAD_STATEID.

8.2.4 Durée de vie et validation d'identifiant d'état

Les identifiants d'état doivent rester valides jusqu'à ce qu'un client redémarre ou un redémarrage de serveur ou jusqu'à ce que le client retourne tous les verrous associés au identifiant d'état au moyen d'une opération comme CLOSE ou DELEGRETURN. Si les verrous sont perdus du fait d'une révocation, tant que l'identifiant de client est valide, l'identifiant d'état reste une désignation valide de cet état révoqué jusqu'à ce que le client le libère en utilisant FREE_STATEID. Les identifiants d'état associés aux verrous de gamme d'octets sont une exception. Ils restent valides même si un LOCKU libère tous les verrous restants, tant que le fichier ouvert avec lequel ils sont associés reste ouvert, sauf si le client libère les identifiants d'état via l'opération FREE_STATEID.

On notera qu'il y a des situations dans lesquelles les verrous de client deviennent invalides, sans que le client demande qu'ils soient retournés. Cela inclut l'expiration de prêt et un certain nombre de formes de révocation de verrous pendant la période de prêt. Il est important de noter que dans ces situations, l'identifiant d'état reste valide et le client peut l'utiliser pour déterminer la disposition des verrous perdus associés.

Une valeur "autre" ne doit jamais être réutilisée pour un objet différent (c'est-à-dire, une bride de fichier, propriétaire, ou type de verrous différents) dans le contexte d'un seul identifiant de client. Un serveur peut conserver la valeur "autre" pour le même objet au delà du point où il peut autrement être libéré, mais si il le fait, il doit conserver la continuité du "seqid" avec les valeurs précédentes.

Une mécanisme qui peut être utilisé pour satisfaire l'exigence que le serveur reconnaisse les identifiants d'état invalides et périmés est que le serveur divise le champ "autre" de l'identifiant d'état en deux champs.

* un indice dans un tableau de structures d'état de verrouillage,
* un numéro de génération qui est incrémenté à chaque allocation d'entrée du tableau pour une utilisation particulière.

Et alors mémoriser dans chaque entrée du tableau,

* l'identifiant de client avec lequel l'identifiant d'état est associé,
* le numéro de génération actuel pour l'identifiant d'état valide (au plus un) partageant cette valeur d'indice,
* la bride de fichier du fichier sur lequel les verrous sont pris,
* une indication du type d'identifiant d'état (ouvert, verrou de gamme d'octets, délégation de fichier, délégation de répertoire, disposition),
* la dernière valeur de "seqid" retournée correspondant à la valeur de "autre" actuelle,
* une indication de l'état actuel des verrous associés à cet identifiant d'état, en particulier, si ils ont été révoqués et si ils le sont, pour quelle raison.

Avec ces informations, un identifiant d'état entrant peut être validé et l'erreur appropriée retournée quand nécessaire. Les identifiants d'état spéciaux et non spéciaux sont traités séparément. (Voir au paragraphe 8.2.3 la discussion des identifiants d'état spéciaux.)

Noter que les identifiants d'état sont implicitement qualifiés par l'identifiant de client courant, comme déduit de l'identifiant de client associé à la session en cours. Noter, cependant, que la sémantique de la session va empêcher les identifiants d'état associés à une précédente instance de client ou serveur d'être analysée par cette procédure.

Si le redémarrage de serveur a résulté en un identifiant de client ou un identifiant de session invalide, SEQUENCE va retourner une erreur et l'opération qui prend un identifiant d'état comme argument ne va jamais être traitée.

Si il y a eu un redémarrage de serveur là où il y a une session persistante et tout l'état prêté a été perdu, alors la session en question va, bien que valide, être marquée comme morte, et toute opération non satisfaite au moyen de la mémoire tampon de réponse va recevoir l'erreur NFS4ERR_DEADSESSION, et donc n'être pas traitée comme indiqué ci-dessous.

Quand un identifiant d'état est vérifié et que le champ "autre" est tout de zéros ou tout de uns, une vérification que les champs "autre" et "seqid" correspondent à une combinaison définie pour un identifiant d'état spécial est faite et le résultat déterminé comme suit :

* Si les champs "autre" et "seqid" ne correspondent pas à une combinaison définie associée à un identifiant d'état spécial, l'erreur NFS4ERR_BAD_STATEID est retournée.
* Si l'identifiant d'état spécial désigne l'identifiant d'état en cours et qu'il y a un identifiant d'état en cours, alors l'identifiant d'état en cours est substitué à l'identifiant d'état spécial et les vérifications appropriées aux identifiants d'état non spéciaux sont effectuées.
* Si la combinaison est valide en général mais n'est pas appropriée au contexte dans lequel l'identifiant d'état est utilisé (par exemple, un identifiant d'état tout de zéros est utilisé quand un identifiant d'état OPEN est requis dans une opération LOCK) l'erreur NFS4ERR_BAD_STATEID est aussi retournée.
* Autrement, la vérification est achevée et l'identifiant d'état spécial est accepté comme valide.

Quand un identifiant d'état est vérifié, et que le champ "autre" n'est ni tout de zéros ni tout de uns, la procédure suivante pourrait être utilisée pour valider un identifiant d'état entrant et retourner une erreur appropriée, quand nécessaire, en supposant que le champ "autre" va être divisé en un indice de tableau et une génération d'entrée.

* Si le champ Indice de tableau est en dehors de la gamme du tableau associé, retourner NFS4ERR_BAD_STATEID.

* Si l'entrée de tableau choisie est d'une génération différente de celle spécifiée dans l'identifiant d'état entrant, retourner NFS4ERR_BAD_STATEID.

* Si l'entrée de tableau choisie ne correspond pas à la bride de fichier en cours, retourner NFS4ERR_BAD_STATEID.

* Si l'identifiant de client dans l'entrée de tableau ne correspond pas à l'identifiant de client associé à la session en cours, retourner NFS4ERR_BAD_STATEID.

* Si l'identifiant d'état représente un état révoqué, alors retourner NFS4ERR_EXPIRED, NFS4ERR_ADMIN_REVOKED, ou NFS4ERR_DELEG_REVOKED, comme approprié.

* Si le type d'identifiant d'état n'est pas valide pour le contexte dans lequel apparaît l'identifiant d'état, retourner NFS4ERR_BAD_STATEID. Noter qu'un identifiant d'état peut être valide en général, comme il serait rapporté par l'opération TEST_STATEID, mais être invalide pour une opération particulière, comme, par exemple, quand un identifiant d'état qui ne représente pas des verrous de gamme d'octets est passé au cas no-from_open de LOCK ou à LOCKU, ou quand un identifiant d'état qui ne représente pas un open est passé à CLOSE ou OPEN_DOWNGRADE. Dans ces cas, le serveur DOIT retourner NFS4ERR_BAD_STATEID.

* Si le champ "seqid" n'est pas zéro et est supérieur à la valeur de séquence courante correspondant au champ "autre" en cours, retourner NFS4ERR_BAD_STATEID.

* Si le champ "seqid" n'est pas zéro et est inférieur à la valeur de séquence courante correspondant au champ "autre" en cours, retourner NFS4ERR_OLD_STATEID.

* Autrement, l'identifiant d'état est valide et l'entrée de tableau devrait contenir toutes les informations supplémentaires sur le type d'identifiant d'état et les informations associées à ce type de type d'identifiant d'état particulier, comme l'ensemble de verrous associés, par exemple, les informations de propriétaire d'ouverture et propriétaire de verrou, ainsi que les informations sur les verrous spécifiques, par exemple, les modes d'ouverture et les gammes d'octets.

8.2.5 Utilisation d'identifiant d'état pour les opérations d'entrée/sortie

Les clients qui effectuent des opérations d'entrée/sortie ont besoin de choisir un identifiant d'état approprié sur la base des verrous (incluant les ouvertures et les délégations) détenus par le client et les divers types de propriétaires d'état qui envoient les demandes d'entrée/sortie. Les opérations SETATTR qui changent la taille de fichier sont traitées comme des opérations d'entrée/sortie à cet égard.

Les règles suivantes, appliquées dans l'ordre de priorité décroissante, gouvernent le choix de l'identifiant d'état approprié. En suivant ces règles, le client va seulement considérer les verrous dont il a réellement reçu notification par une réponse ou rappel d'opération approprié. Noter que les règles sont légèrement différentes dans le cas d'entrée/sortie aux serveurs de données quand des dispositions de fichier sont utilisées (voir le paragraphe 13.9.1).

* Si le client détient une délégation pour le fichier en question, l'identifiant d'état de délégation DEVRAIT être utilisé.

* Autrement, si l'entité correspondant au propriétaire de verrou (par exemple, un processus) qui envoie l'entrée/sortie a un identifiant d'état de verrou de gamme d'octets pour le fichier ouvert associé, alors l'identifiant d'état de verrou de gamme d'octets pour ce propriétaire de verrou et le fichier ouvert DEVRAIT être utilisé.

* Si il n'y a pas d'identifiant d'état de verrou de gamme d'octets, alors l'identifiant d'état OPEN pour le fichier ouvert en question DEVRAIT être utilisé.

* Finalement, si aucune des conditions ci-dessus ne s'applique, alors un identifiant d'état spécial DEVRAIT être utilisé.

Ignorer ces règles peut résulter en des situations dans lesquelles le serveur n'a pas les informations nécessaires pour traiter correctement la demande. Par exemple, quand des verrous de gamme d'octets obligatoires sont en vigueur, si l'identifiant d'état n'indique pas le propriétaire de verrou approprié, via un identifiant d'état de verrou, une demande pourrait être inutilement rejetée.

Le serveur ne devrait cependant pas essayer d'appliquer ces règles d'ordre et devrait utiliser toute information disponible pour traiter correctement les demandes d'entrée/sortie. En particulier, quand un client a une délégation pour un fichier donné, il DEVRAIT en prendre note dans le traitement d'une demande, même si elle est envoyée avec un identifiant d'état spécial.

8.2.6 Utilisation d'identifiant d'état pour les opérations SETATTR

Parce que chaque opération est associée à un identifiant de session d'où le clientid peut être déterminé, les opérations n'ont pas besoin d'inclure un identifiant d'état pour que le serveur soit capable de déterminer si elles devraient causer le rappel d'une délégation ou sont à traiter comme dans la portée de la délégation.

Dans le cas des opérations SETATTR, un identifiant d'état est présent. Dans les cas autres que ceux qui établissent la taille de fichier, le client peut envoyer un identifiant d'état spécial ou, quand une délégation est détenue pour le fichier en question, un identifiant d'état de délégation. Alors que le serveur DEVRAIT valider l'identifiant d'état et peut utiliser l'identifiant d'état pour optimiser la détermination de si une délégation est détenue, il DEVRAIT noter la présence d'une délégation même quand un identifiant d'état spécial est envoyé, et DOIT accepter un identifiant d'état de délégation valide quand il est envoyé.

8.3 Renouvellement de prêt

Chaque paire de client/serveur, comme représentée par un identifiant de client, a un seul prêt. L'objet du prêt est de permettre au client d'indiquer au serveur, d'une façon peu coûteuse en frais généraux, qu'il est actif, et donc que le serveur doit conserver les verrous du client. Cet arrangement permet au serveur de supprimer les objets périmés relatifs au verrouillage qui sont détenus par un client qui s'est arrêté ou est autrement injoignable, une fois que le prêt pertinent arrive à expiration. Cela permet ensuite que d'autres clients obtiennent des verrous en conflit sans être retardés indéfiniment par des clients inactifs ou injoignables. Ce n'est pas un mécanisme pour la cohérence des antémémoires et les renouvellements de prêts ne peuvent pas être refusés si l'intervalle de prêt n'est pas expiré.

Comme chaque session est associée à un client spécifique (identifié par son identifiant de client) toute opération envoyée sur cette session est une indication que le client associé est joignable. Quand une demande est envoyée pour une session, la réussite de l'exécution d'une opération SEQUENCE (ou la restitution réussie du résultat de SEQUENCE de la mémoire tampon de réponse) sur un prêt non expiré va résulter en le renouvellement implicite du prêt, pour la période de renouvellement standard (égale à l'attribut lease_time).

Si le prêt de l'identifiant de client n'a pas encore expiré quand le serveur reçoit une opération SEQUENCE, alors le serveur DOIT renouveler le prêt. Si le prêt de l'identifiant de client a expiré quand le serveur reçoit une opération SEQUENCE, le serveur PEUT renouveler le prêt ; cela dépend de si un état a été révoqué par suite de l'échec du client à renouveler le prêt avant l'expiration.

En l'absence d'autre activité qui renouvellerait le prêt, un COMPOUND consistant en une seule opération SEQUENCE va suffire. Le client devrait aussi prendre en compte les délais relatifs à la communication et prendre des mesures pour s'assurer que les messages de renouvellement atteignent réellement le serveur en temps utile. Par exemple :

* Quand le multiplexage est utilisé, le client devrait envisager d'envoyer plusieurs demandes sur différentes connexions, afin d'assurer que le renouvellement se produit, même en cas de blocage sur le chemin utilisé par une de ces connexions.

* Les délais de transport de retransmission pourraient devenir si grands qu'ils s'approchent ou excèdent la durée de la période de prêt. Cela peut être particulièrement probable quand le serveur ne répond pas à cause d'un redémarrage ; voir le paragraphe 8.4.2.1. Si la mise en œuvre de client n'y est pas attentive, les délais de transport de retransmission peuvent résulter en ce que le client échoue à détecter un redémarrage de serveur avant la fin de la période de grâce. Le scénario est que le client utilise un transport avec un retard exponentiel, de sorte que la temporisation maximum de retransmission excède à la fois la période de grâce et l'attribut lease_time. Une partition de réseau cause l'intervalle de retransmission de la connexion du client à doubler, et même après la réparation de la partition, la prochaine retransmission de niveau transport est envoyée après que le serveur a redémarré et que sa période de grâce s'est terminée.

Le client DOIT soit récupérer des erreurs NFS4ERR_NO_GRACE qui s'ensuivent, soit DOIT s'assurer que, en dépit des intervalles de retransmission de niveau transport qui excèdent le lease_time, une opération SEQUENCE est envoyée qui renouvelle le prêt avant son expiration. Le client peut réaliser cela en associant une nouvelle connexion à la session, et en envoyant une opération SEQUENCE sur elle. Cependant, si la tentative d'établir une nouvelle connexion est retardée pour une raison quelconque (par exemple, le retard exponentiel des paquets d'établissement de la connexion) le client devra interrompre la tentative d'établissement de la connexion avant l'expiration du prêt, et tenter de se reconnecter.

Si le serveur renouvelle le prêt à réception d'une opération SEQUENCE, le serveur NE DOIT PAS permettre que le prêt expire alors que le reste des opérations de la demande de procédure COMPOUND est encore en cours d'exécution. Une fois que la dernière opération est finie, et que la réponse au COMPOUND a été envoyée, le serveur DOIT régler le prêt à expirer pas plus tôt que la somme de l'heure courante et de la valeur de l'attribut lease_time.

Un prêt d'identifiant de client peut expirer quand il y a eu au moins l'intervalle de prêt (lease_time) depuis la dernière

opération SEQUENCE de renouvellement de prêt envoyée sur toute session de l'identifiant de client il qu'il n'y a pas d'opération COMPOUND non active sur de telles sessions.

Parce que l'opération SEQUENCE est le mécanisme de base pour renouveler un prêt, et parce que elle doit être faite au moins une fois pour chaque période de prêt, c'est le mécanisme naturel par lequel le serveur va informer le client de changements dans l'état de prêt dont le client a besoin d'être informé. Le client devrait inspecter les fanions d'état (sr_status_flags) retournés par séquence et prendre les mesures appropriées (voir les détails au paragraphe 18.46.3).

* Les bits d'état SEQ4_STATUS_CB_PATH_DOWN et SEQ4_STATUS_CB_PATH_DOWN_SESSION indiquent des problèmes avec le canal de retour que le client peut devoir traiter afin de recevoir les demandes de rappel.

* Les bits d'état SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRING et SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRED indiquent des problèmes avec les contextes GSS ou les brides RPCSEC_GSS pour le canal de retour que le client pourrait devoir traiter afin de permettre l'envoi des demandes de rappel.

* Les bits d'état SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED, SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED, SEQ4_STATUS_ADMIN_STATE_REVOKED, et SEQ4_STATUS_RECALLABLE_STATE_REVOKED notifient au client les événements de révocation de verrou. Quand ces bits sont établis, le client devrait utiliser TEST_STATEID pour trouver quels identifiants d'état ont été révoqués et utiliser FREE_STATEID pour accuser réception de la perte de l'état associé.

* Le bit d'état SEQ4_STATUS_LEASE_MOVE indique que la responsabilité du renouvellement de prêt a été transférée à un ou plusieurs nouveaux serveurs.

* Le bit d'état SEQ4_STATUS_RESTART_RECLAIM_NEEDED indique que du fait du redémarrage de serveur, le client doit redemander l'état de verrouillage.

* Le bit d'état SEQ4_STATUS_BACKCHANNEL_FAULT indique que le serveur a rencontré une faute irrécupérable avec le canal de retour (par exemple, il a perdu trace d'un identifiant de séquence pour un créneau dans le canal de retour).

8.4 Récupération de panne

Une exigence critique dans la récupération de panne est que le client et le serveur sachent quand l'autre est défaillant. De plus, il est exigé qu'un client ait une vue cohérente des données à travers les redémarrages du serveur. Toutes les opérations READ et WRITE qui peuvent avoir été mises en file d'attente dans les mémoires-tampons du client ou du réseau doivent attendre que le client ait réussi à récupérer les verrous protégeant les opérations READ et WRITE. Toutes celles qui atteignent le serveur avant qu'il puisse sûrement déterminer que le client a récupéré assez d'état de verrouillage pour que de telles opérations puissent être traitées en toute sécurité doivent être rejetées. Cela va arriver parce que soit :

* L'état présenté n'est plus valide car il est associé à un identifiant de client maintenant invalide. Dans ce cas, le client va recevoir une erreur NFS4ERR_BADSESSION ou NFS4ERR_DEADSESSION, et toute tentative d'attacher une nouvelle session à cet identifiant de client invalide va résulter en une erreur NFS4ERR_STALE_CLIENTID.
* La récupération de verrous suivante peut rendre l'exécution de l'opération inappropriée (NFS4ERR_GRACE).

8.4.1 Défaillance et récupération de client

Dans le cas où un client échoue, le serveur peut libérer les verrous du client quand le prêt associé a expiré. Les verrous en conflit provenant d'un autre client peuvent seulement être accordés après cette expiration de prêt. Comme discuté au paragraphe 8.3, quand un client n'a pas défailli et rétablit son prêt avant que se produise l'expiration, les demandes de verrous en conflit ne seront pas accordées.

Pour minimiser les délais sur le client lors d'un redémarrage, les demandes de verrous sont associées à une instance du client par un vérificateur fourni par le client. Ce vérificateur fait partie du client_owner4 envoyé dans l'appel EXCHANGE_ID initial fait par le client. Le serveur retourne un identifiant de client par suite de l'opération EXCHANGE_ID. Le client confirme alors l'utilisation de l'identifiant de client en établissant une session associée à cet identifiant de client (voir au paragraphe 18.36.3 la description de la façon dont cela est fait). Tous les verrous, y compris d'ouverture, verrous de gamme d'octets, de délégations, et de dispositions obtenus par des sessions en utilisant cet identifiant de client, sont associés à cet identifiant de client.

Comme le vérificateur va être changé par le client à chaque initialisation, le serveur peut comparer un nouveau vérificateur au vérificateur associé aux verrous actuellement détenus et déterminer si ils ne correspondent pas. Cela signifie la nouvelle instanciation et la perte ultérieure de l'état de verrouillage du client (à la confirmation du nouvel identifiant de client). Par suite, le serveur est libre de libérer tous les verrous détenus qui sont associés au vieil identifiant de client déduit du vieux vérificateur. À ce point, les verrous en conflit provenant d'autres clients, gardés en attente tant que le prêt n'avait pas expiré, peuvent être accordés. De plus, tous les identifiant d'état associés au vieil identifiant de client peuvent aussi être libérés, car ils ne sont plus capables de référence.

Noter que le vérificateur doit avoir les mêmes propriétés d'unicité que le vérificateur pour l'opération COMMIT.

8.4.2 Défaillance et récupération de serveur

Si le serveur perd l'état de verrouillage (généralement par suite d'un redémarrage) il doit donner aux clients le temps de découvrir cela et rétablir l'état de verrouillage perdu. Le client doit être capable de rétablir l'état de verrouillage sans que le serveur refuse des demandes valides parce que le serveur a accordé un accès en conflit à un autre client. De même, si il y a une possibilité que des clients n'aient pas encore rétabli leur état de verrouillage pour un fichier et que cet état de verrouillage pourrait le rendre invalide pour effectuer des opérations READ ou WRITE. Par exemple, si des verrous obligatoires sont possibles, le serveur doit interdire les opérations READ et WRITE pour ce fichier.

Un client peut déterminer que la perte d'état de verrouillage s'est produite via plusieurs méthodes.

1. Quand une opération SEQUENCE (le plus courant) ou autre retourne NFS4ERR_BADSESSION, cela peut signifier que la session a été détruite mais que l'identifiant de client est encore valide. Le client envoie une demande CREATE_SESSION avec l'identifiant de client pour rétablir la session. Si CREATE_SESSION échoue avec NFS4ERR_STALE_CLIENTID, le client doit établir un nouvel identifiant de client (voir le paragraphe 8.1) et rétablir son état de verrou avec le nouvel identifiant de client, après la réussite de l'opération CREATE_SESSION (voir le paragraphe 8.4.2.1).

2. Quand une opération SEQUENCE (le plus courant) ou autre sur une session persistante retourne NFS4ERR_DEADSESSION, cela indique qu'une session n'est plus utilisable pour de nouvelles opérations, c'est-à-dire, non satisfaites à partir de la mémoire tampon de réponse. Une fois que toutes les opérations en instance sont déterminées être effectuées avant le ré-essai ou non effectuées, le client envoie une demande CREATE_SESSION avec l'identifiant de client pour rétablir la session. Si CREATTE_SESSION échoue avec NFS4ERR_STALE_CLIENTID, le client doit établir un nouvel identifiant de client (voir le paragraphe 8.1) et rétablir son état de verrou après que le CREATE_SESSION réussit, avec le nouvel identifiant de client (paragraphe 8.4.2.1).

3. Quand une opération, ni SEQUEFNCE ni précédée par SEQUENCE (par exemple, CREATE_SESSION, DESTROY_SESSION) retourne NS4ERR_STALE_CLIENTID, le client DOIT établir un nouvel identifiant de client (paragraphe 8.1) et rétablir son état de verrou (paragraphe 8.4.2.1).

8.4.2.1 Réclamation d'état

Quand les informations d'état et les verrous associés sont perdus par suite d'un redémarrage de serveur, le protocole doit fournir un moyen de causer le rétablissement de cet état. L'approche utilisée est de définir, pour la plupart des types d'état de verrouillage (les dispositions sont une exception) une demande dont la fonction est de permettre au client de rétablir sur le serveur un verrou obtenu d'abord d'une précédente instance. Généralement, ces demandes sont des variantes des demandes normalement utilisées pour créer des verrous de ce type et sont appelées des demandes "de réclamation de type", et le processus de rétablissement de tels verrous est appelé une "réclamation".

Parce que chaque client doit avoir une opportunité de réclamer tous les verrous qu'il a sans qu'il soit possible qu'un autre client se voir accorder un verrou en conflit, une "période de grâce" est dédiée au processus de réclamation. Durant cette période, les demandes de création des identifiants de client et de sessions sont traitées normalement, mais les demandes de verrouillage sont soumises à des restrictions particulières. Seules les demandes de verrouillage de type réclamation sont permises, sauf si le serveur peut déterminer fiablement (par un état maintenu de façon persistante à travers les instances de redémarrage) qu'accorder un tel verrou ne peut pas entrer en conflit avec une réclamation ultérieure. Quand une demande est faite pour obtenir un nouveau verrou (c'est-à-dire, pas une demande de type réclamation) durant la période de grâce et qu'une telle détermination ne peut pas être faite, le serveur doit retourner l'erreur NFS4ERR_GRACE.

Une fois une session établie en utilisant le nouvel identifiant de client, le client va utiliser des demandes de verrouillage de

type réclamation (par exemple, des opérations LOCK avec réclamation réglé à VRAI et opérations OPEN avec un type de réclamation CLAIM_PREVIOUS ; voir le paragraphe 9.11) pour rétablir son état de verrouillage. Une fois cela fait, ou si il n'y a pas de tel état de verrouillage à réclamer, le client envoie une opération RECLAIM_COMPLETE globale, c'est-à-dire, une avec l'argument rca_one_fs réglé à FAUX, pour indiquer qu'il a réclamé tout l'état de verrouillage qu'il veut réclamer. Une fois qu'un client a envoyé une telle opération RECLAIM_COMPLETE, il peut tenter des opérations de verrouillage autres que de réclamation, bien qu'il pourrait obtenir un résultat d'état NFS4ERR_GRACE sur chacune de ces opérations jusqu'à ce que la période de traitement spécial soit terminée. Voir au paragraphe 11.11.9 la discussion du traitement analogue de réclamation de verrou dans le cas de transition de systèmes de fichiers de serveur à serveur.

Durant la période de grâce, le serveur doit rejeter les opérations READ et WRITE et les demandes de verrouillage non de réclamation (c'est-à-dire, les autres opérations LOCK et OPEN) avec une erreur de NFS4ERR_GRACE, sauf si il peut garantir que cela peut être fait en toute sécurité, comme décrit ci-dessous.

La période de grâce peut durer jusqu'à ce que tous les clients qui sont connus comme pouvant avoir des verrous ont fait une opération RECLAIM_COMPLETE globale, indiquant qu'ils ont fini de réclamer les verrous qu'ils détenaient avant le redémarrage de serveur. Cela signifie que un client qui a fait un RECLAIM_COMPLETE doit être prêt à recevoir un NFS4ERR_GRACE quand il tente d'acquérir de nouveaux verrous. Afin que le serveur sache que tous les clients avec un état de verrou antérieur possible ont fait un RECLAIM_COMPLETE, le serveur doit tenir dans une mémorisation stable une liste des clients qui peuvent avoir de tels verrous. Le serveur peut aussi terminer la période de grâce avant que tous les clients aient fait un RECLAIM_COMPLETE global. Le serveur NE DEVRAIT PAS terminer la période de grâce avant une durée égale à la période de prêt afin de donner aux clients l'opportunité de découvrir le redémarrage de serveur, suite à l'envoi de demandes sur les sessions associées à une fréquence gouvernée par le temps de prêt. Noter que quand un client n'envoie pas de telles demandes (ou qu'elles sont envoyées par le client mais non reçues par le serveur) il est possible que la période de grâce expire avant que le client découvre que le redémarrage de serveur s'est produit.

Du temps supplémentaire pour permettre à un client d'établir un nouvel identifiant de client et de session et d'effectuer les réclamations de verrou peut être ajouté au temps de prêt. Noter que des règles analogues s'appliquent aux périodes de grâce spécifiques de système de fichiers discutées au paragraphe 11.11.9.

Si le serveur peut déterminer fiablement qu'accorder une demande non de réclamation ne va pas entrer en conflit avec une réclamation de verrous par d'autres clients, l'erreur NFS4ERR_GRACE n'a pas à être retournée même dans la période de grâce, bien que NFS4ERR_GRACE doive toujours être retourné aux clients qui tentent une demande non de réclamation de verrou avant de faire leur propre RECLAIM_COMPLETE global. Pour que le serveur soit capable de servir les opérations READ et WRITE durant la période de grâce, il doit là encore être capable de garantir qu'aucun conflit possible ne pourrait survenir entre une potentielle réclamation de verrouillage et l'opération READ ou WRITE. Si le serveur est incapable de donner cette garantie, l'erreur NFS4ERR_GRACE doit être retournée au client.

Pour qu'un serveur fournisse un traitement simple et valide durant la période de grâce, la méthode la plus facile est de simplement rejeter toutes les demandes de verrouillage non de réclamation et les opérations READ et WRITE en retournant l'erreur NFS4ERR_GRACE. Cependant, un serveur peut garder les informations sur les verrous accordés dans une mémorisation stable. Avec ces informations, le serveur pourrait déterminer si un verrouillage, une opération READ ou WRITE peut être traitée en toute sécurité.

Par exemple, si le serveur a conservé dans une mémorisation stable un sommaire des informations sur si il existe des verrous obligatoires, des verrous de gamme d'octets obligatoires, ou des réservations partagées spécifiant des modes de refus, de nombreuses demandes pourraient être permises durant la période de grâce. Si il est connu qu'aucun partage de réservations n'existe, la demande OPEN qui ne spécifie pas de modes de refus peut être accordée en toute sécurité. Si, de plus, il est connu qu'il n'existe pas de verrous de gamme d'octets obligatoires, soit par des informations conservées dans une mémorisation stable ou simplement parce que le serveur ne prend pas en charge de tels verrous, les opérations READ et WRITE peuvent être traitées en toute sécurité durant la période de grâce. Un autre cas important est lorsque il est connu que il existe des verrous de gamme d'octets non obligatoires, soit parce que le serveur ne fournit pas leur prise en charge, soit parce que leur absence est connue d'après de données enregistrées de façon persistante. Dans ce cas, les opérations READ et WRITE qui spécifient des identifiants d'état déduits d'opérations de type réclamation peuvent être traitées validement pendant la période de grâce à cause du fait que la réclamation valide assure que aucun verrou accordé ensuite ne peut empêcher l'entrée/sortie.

Pour répéter, pour qu'un serveur permette de traiter des demandes non réclamation de verrou et d'entrée/sortie durant la période de grâce, il DOIT déterminer qu'aucun verrou réclamé ultérieurement ne va être rejeté et qu'aucun verrou réclamé ultérieurement n'aura empêché de traiter une opération d'entrée/sortie durant la période de grâce.

Les clients devraient être prêts au retour d'erreurs NFS4ERR_GRACE pour des demandes non de réclamation de verrou et

d'entrée/sortie. Dans ce cas, le client devrait employer un mécanisme de réessai pour la demande. Un délai (de l'ordre de plusieurs secondes) entre les réessais devrait être utilisé pour éviter de surcharger le serveur. Une discussion de la question générale est incluse dans [SYNCHRO]. Le client doit tenir compte des serveurs qui peuvent effectuer des demandes d'entrée/sortie et de non réclamation de verrouillage dans la période de grâce ainsi que de ceux qui ne le peuvent pas.

Une demande de verrouillage de type réclamation en dehors de la période de grâce du serveur peut seulement réussir si le serveur peut garantir qu'aucune demande de verrou en conflit ou d'entrée/sortie n'a été accordée depuis le redémarrage.

Un serveur peut, au redémarrage, établir une nouvelle valeur pour la période de prêt. Donc, les clients devrait, une fois qu'un nouvel identifiant de client est établi, retourner aller chercher l'attribut lease_time et l'utiliser comme base du renouvellement de prêt pour le prêt associé à ce serveur. Cependant, le serveur doit établir, pour cet événement de redémarrage, une période de grâce au moins aussi longue que la période de prêt pour la précédente instance de serveur. Cela permet que l'état de client obtenu durant la précédente instance de serveur soit rétabli de façon fiable.

La possibilité existe que, à cause d'événements de configuration de serveur, le client communique avec un serveur différent de celui sur lequel les verrous ont été obtenus, comme montré par les combinaisons de eir_server_scope et eir_server_owner. Cela conduit à la question de si et quand le client devrait tenter de réclamer les verrous précédemment obtenus sur ce qui est rapporté comme étant un serveur différent. Les règles pour résoudre cette question sont les suivantes :

* Si la portée de serveur est différente, le client ne devrait pas tenter de réclamer les verrous. Dans cette situation, aucune réclamation de verrou n'est possible. Toute tentative de ré-obtenir les verrous avec des opérations non de réclamation est problématique car il n'est pas garanti que les brides de fichier existantes vont être reconnues par le nouveau serveur, ou que si elles sont reconnues, elles notent les mêmes objets. Il vaut mieux traiter les verrous comme ayant été révoqués par l'événement de reconfiguration.

* Si la portée de serveur est la même, le client devrait tenter de réclamer les verrous, même si la valeur de eir_server_owner est différente. Dans cette situation, il est de la responsabilité du serveur de retourner NFS4ERR_NO_GRACE si il ne peut pas fournir un soutien correct des opérations de réclamation de verrous, y compris la prévention de conditions limites.

Le champ eir_server_owner n'est pas utilisé pour cette détermination. Sa fonction est de spécifier les possibilités de multiplexage pour le client (voir le paragraphe 2.10.5) et non de contrôler les réclamations de verrou.

8.4.2.1.1 Considérations sur la sécurité de la réclamation d'état

Durant la période de grâce, un client peut réclamer l'état qu'il croit ou affirme qu'il avait avant le redémarrage du serveur. Sauf si le serveur a maintenu un enregistrement complet de tout l'état qu'avait le client, le serveur a peu de choix à part de faire confiance au client. (Bien sûr, si le serveur a conservé un enregistrement complet, il n'aurait alors pas à forcer le client à réclamer l'état après le redémarrage de serveur.) Alors que le serveur doit faire confiance au client pour dire la vérité, les conséquences négatives pour la sécurité sont limitées à permettre des attaques de déni de service dans des situations où AUTH_SYS est pris en charge. La règle fondamentale pour le serveur quand il traite les demandes de réclamation est qu'il NE DOIT PAS accorder la réclamation si une demande équivalente non de réclamation ne serait pas accordée durant l'état permanent à cause de problèmes de contrôle d'accès ou de conflit d'accès. Par exemple, une demande OPEN durant une réclamation va être refusée avec NFS4ERR_ACCESS si le principal qui fait la demande n'a pas accès à l'ouverture du fichier en accord avec l'ACL discrétionnaire (paragraphe 6.2.2) sur le fichier.

Néanmoins, il est possible qu'un client opérant par erreur ou malveillance pourrait, durant une réclamation, empêcher un autre client de réclamer l'accès à l'état. Par exemple, un attaquant pourrait envoyer une opération de réclamation OPEN avec un mode de refus qui empêche un autre client de réclamer l'état OPEN qu'il avait avant le redémarrage du serveur. L'attaquant pourrait effectuer le même déni de service durant l'état permanent avant le redémarrage de serveur, pour autant que l'attaquant avait les permissions. Étant donné que les vecteurs d'attaque sont équivalents, la période de grâce n'offre aucune opportunité supplémentaire de déni de service, et tous les problèmes sur ce vecteur d'attaque, que ce soit durant la période de grâce ou l'état permanent, sont traités de la même façon : utiliser RPCSEC_GSS pour l'authentification et limiter l'accès au fichier aux seuls principaux auxquels le propriétaire du fichier fait confiance.

Noter que si avant de redémarrer le serveur avait les identifiants de client avec la capacité EXCHGID4_FLAG_BIND_PRINC_STATEID (paragraphe 18.35) établie, alors le serveur DEVRAIT enregistrer dans une mémorisation stable le propriétaire de client et le principal qui avait établi l'identifiant de client via EXCHANGE_ID. Si le serveur ne l'a pas fait, alors il y a un risque qu'un client soit incapable de réclamer l'état si il n'a pas un accréditif pour un principal qui était à l'origine autorisé à établir l'état.

8.4.3 Partitions et récupérations de réseau

Si la durée d'une partition de réseau est supérieure à la période de prêt fournie par le serveur, le serveur n'aura pas reçu de renouvellement de prêt du client. Si cela se produit, le serveur peut libérer tous les verrous détenus pour le client ou il peut permettre que l'état de verrou reste pour une durée considérable, sous réserve de la contrainte que si une demande pour un verrou en conflit est faite, les verrous associés à un prêt expiré n'empêchent pas un tel verrou en conflit d'être accordé mais il DOIT être révoqué comme nécessaire pour éviter d'interférer avec de telles demandes en conflit.

Si le serveur choisit de retarder la libération de l'état de verrouillage jusqu'à ce qu'il y ait un conflit, il peut soit libérer tous les verrous du client une fois qu'il y a un conflit, soit seulement révoquer l'ensemble minimum de verrous nécessaire pour permettre les demandes en conflit. Quand il adopte l'approche de plus fine granularité, il doit révoquer tous les verrous associés à un identifiant d'état donné, même si le conflit est seulement avec un sous ensemble de verrous.

Quand le serveur choisit de libérer tout l'état de verrouillage d'un client, soit immédiatement à l'expiration du prêt, soit par suite de la première tentative d'obtenir un verrou en conflit, le serveur peut rapporter la perte de l'état de verrouillage d'un certain nombre de façons.

Le serveur peut choisir d'invalider la session et l'identifiant de client associé. Dans ce cas, une fois que le client peut communiquer avec le serveur, il va recevoir une erreur NFS4ERR_BADSESSION. En tentant de créer une nouvelle session, il va obtenir un NFS4ERR_STALE_CLIENTID. En créant le nouvel identifiant de client et la nouvelle session, le client va tenter de réclamer les verrous. Normalement, le serveur ne va pas permettre au client de réclamer les verrous, parce que le serveur ne va pas être dans sa période de grâce de récupération.

Une autre possibilité est que le serveur maintienne l'identifiant de session et de client mais pour que tous les identifiants d'état détenus par le client deviennent invalides ou périmés. Une fois que le client peut joindre le serveur après une telle partition de réseau, l'état retourné par l'opération SEQUENCE va indiquer une perte de l'état de verrouillage ; c'est-à-dire, le fanion SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED va être établi dans les fanions sr_status_flags. De plus, toutes les entrées/sorties soumises par le client avec les identifiants d'état maintenant invalides vont échouer, le serveur retournant l'erreur NFS4ERR_EXPIRED. Une fois que le client apprend la perte de l'état de verrouillage, il va convenablement le notifier aux applications qui détenaient les verrous invalidés. Le client devrait alors effectuer une action pour libérer les identifiants d'état invalidés, soit en établissant un nouvel identifiant de client en utilisant un nouveau vérificateur, soit en faisant une opération FREE_STATEID pour libérer chacun des identifiants d'état invalidés.

Quand le serveur adopte une approche de granularité plus fine de la révocation de verrous lorsque le prêt d'un client a expiré, seul un sous ensemble d'identifiants d'état va normalement devenir invalide durant une partition de réseau. Quand le client peut communiquer avec le serveur après qu'une telle partition de réseau est réparée, l'état retourné par l'opération SEQUENCE va indiquer une perte partielle de l'état de verrouillage (SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED). De plus, les opérations, y compris les entrées/sorties soumises par le client, avec les identifiants d'état maintenant invalides vont échouer avec le serveur qui retourne l'erreur NFS4ERR_EXPIRED. Une fois que le client a appris la perte de l'état de verrouillage, il va utiliser l'opération TEST_STATEID sur tous ses identifiants d'état pour déterminer quels verrous ont été perdus et alors en notifier convenablement les applications qui détiennent les verrous invalidés. Le client peut alors libérer l'état de verrouillage invalidé et accuser réception de la révocation des verrous associés en faisant une opération FREE_STATEID sur chacun des identifiants d'état invalidés.

Quand une partition de réseau est combinée avec un redémarrage de serveur, il y a des conditions limites qui imposent des exigences au serveur afin d'éviter la corruption silencieuse de données à la suite du redémarrage de serveur. Deux de ces conditions limites sont connues, et sont discutées ci-dessous.

La première condition limite survient par suite de scénarios comme le suivant :

1. Le client A acquiert un verrou.
2. Le client A et le serveur subissent une partition de réseau mutuelle, telle que le client A est incapable de renouveler son prêt.
3. Le prêt du client A expire, et le serveur libère le verrou.
4. Le client B acquiert un verrou qui serait rentré en conflit avec celui du client A.
5. Le client B libère son verrou.
6. Le serveur redémarre.
7. La partition de réseau entre le client A et le serveur est réparée.
8. Le client A se connecte à une nouvelle instance de serveur et découvre le redémarrage du serveur.

9. Le client A réclame son verrou dans la période de grâce du serveur.

Donc, à l'étape finale, le serveur a par erreur accordé la réclameation de verrou du client A. Si le client B a modifié l'objet que le verrou protégeait, le client A va rencontrer une corruption d'objet.

La seconde condition limite connue survient dans des situations comme la suivante :
1. Le client A acquiert un ou plusieurs verrous.
2. Le serveur redémarre.
3. Le client A et le serveur subissent une partition de réseau mutuelle, telle que le client A est incapable de réclamer tous ses verrous dans la période de grâce.
4. La période de grâce de réclamation du serveur se termine. Le client A n'a soit pas de verrous, soit un ensemble incomplet de verrous connus du serveur.
5. Le client B acquiert un verrou qui aurait été en conflit avec un verrou du client A qui n'a pas été réclamé.
6. Le client B libère le verrou.
7. Le serveur redémarre une seconde fois.
8. La partition de réseau entre le client A et le serveur est réparée.
9. Le client A se connecte à la nouvelle instance de serveur et découvre le redémarrage du serveur.
10. Le client A réclame son verrou dans la période de grâce du serveur.

Comme avec la première condition limite, l'étape finale du scénario de la seconde condition limite fait que le serveur accorde à tort la réclamation de verrou du client A.

Résoudre ces deux conditions limites exige que le serveur suppose toujours après son redémarre qu'une condition limite va se produire, et donc retourne NFS4ERR_NO_GRACE pour toutes les tentatives de réclamation, ou que le serveur enregistre des informations dans une mémorisation stable. La quantité d'informations que le serveur enregistre dans une mémorisation stable est en proportion inverse de la sévérité que le serveur entend avoir chaque fois que des conditions limites surviennent. Le serveur qui est complètement tolérant à toutes les conditions limites va enregistrer dans une mémorisation stable chaque verrou acquis, supprimant l'enregistrement de verrou de sa mémorisation stable seulement quand le verrou est libéré. Pour les deux conditions limites discutées ci-dessus, le plus sévère qu'un serveur puisse être, et prendre encore en charge une période de grâce pour les réclamations, exige que le serveur enregistre dans sa mémorisation stable des informations minimales. Par exemple, une mise en œuvre de serveur pourrait, pour chaque client, sauvegarder dans sa mémorisation stable un enregistrement contenant :

* Le champ co_ownerid provenant du client_owner4 présenté dans l'opération EXCHANGE_ID.

* Un booléen qui indique si le prêt du client a expiré ou si il y a eu une intervention administrative (voir le paragraphe 8.5) pour révoquer un verrou de gamme d'octets, un partage de réservation, ou une délégation et qu'il n'y a pas eu d'accusé de réception, via FREE_STATEID, d'une telle révocation.

* Un booléen qui indique si le client peut avoir des verrous qu'il estime être réclamables dans des situations où la période de grâce était terminée, rendant suspecte la vue du serveur de la possibilité de réclamer le verrou. Le serveur va l'établir pour tout enregistrement de client dans sa mémorisation stable lorsque le client n'a pas fait un RECLAIM_COMPLETE convenable (global ou spécifique du système de fichiers selon la cible de la demande de verrou) avant qu'il accorde un nouveau (c'est-à-dire, non réclamé) verrou à un client.

En supposant la conservation d'enregistrement ci-dessus, pour la première condition limite, après le redémarrage du serveur, l'enregistrement que le prêt du client A a expiré signifie qu'un autre client pourrait avoir acquis un verrou de gamme d'octets, un partage de réservation, ou une délégation, en conflit. Donc, le serveur doit rejeter une réclamation du client A avec l'erreur NFS4ERR_NO_GRACE.

Pour la seconde condition limite, après le redémarrage du serveur pour la seconde fois, l'indication que le client n'avait pas terminé ses réclamations au moment de la fin de la période de grâce signifie que le serveur doit rejeter une réclamation du client A avec l'erreur NFS4ERR_NO_GRACE.

Quand l'une ou l'autre des conditions limites se produit, la tentative du client de réclamer des verrous va résulter en l'erreur NFS4ERR_NO_GRACE. Quand cette réponse est reçue, ou après que le client redémarre sans état de verrouillage, le client va envoyer un RECLAIM_COMPLETE global. Quand le RECLAIM_COMPLETE est reçu, le serveur et le client sont de nouveau en accord sur les verrous réclamables et les deux booléens dans la mémorisation persistante peuvent être réinitialisés, pour n'être établis à nouveau que quand un événement suivant cause la mise en question des opérations de réclamation de verrou.

Sans considération du niveau et de l'approche de la tenue d'enregistrements, le serveur DOIT mettre en œuvre une des stratégies suivantes (qui s'appliquent aux réclamations de partage de réservations, de verrous de gamme d'octets, et de délégations) :

1. Rejeter toutes les réclamations avec NFS4ERR_NO_GRACE. C'est extrêmement impitoyable, mais nécessaire si le serveur n'enregistre pas l'état de verrouillage dans une mémorisation stable.

2. Enregistrer suffisamment d'état dans une mémorisation stable pour que toute condition limite connue impliquant un redémarrage de serveur, y compris les deux notées dans ce paragraphe, soit détectée. Il est acceptable de reconnaître à tort une condition limite et ne pas permettre une réclamation, quand, avec une connaissance suffisante, elle aurait été permise. L'erreur que le serveur retournerait dans ce cas est NFS4ERR_NO_GRACE. Noter qu'on ne sait pas si il y a d'autres conditions limites.

Dans le cas où, après un redémarrage de serveur, le serveur détermine qu'il y a un dommage ou corruption irrécupérable des informations dans la mémorisation stable, alors pour tous les clients et/ou verrous qui peuvent être affectés, le serveur DOIT retourner NFS4ERR_NO_GRACE.

Une directive sur le traitement par le client de l'erreur NFS4ERR_NO_GRACE sortirait du domaine d'application de la présente spécification, car les stratégies pour un tel traitement dépendent beaucoup de l'environnement de fonctionnement du client. Cependant, une approche potentielle est décrite ci-dessous.

Quand le client reçoit NFS4ERR_NO_GRACE, il pourrait examiner l'attribut de changement des objets pour lesquels le client essaye de réclamer l'état, et utiliser cela pour déterminer si il rétablit l'état via des opérations OPEN ou LOCK normales. C'est acceptable pourvu que l'environnement de fonctionnement du client le permette. En d'autres termes, il est conseillé à la mise en œuvre de client de documenter le comportement pour ses utilisateurs. Le client pourrait aussi informer l'application que son verrou de gamme d'octets ou ses réservations partagées (qu'elles aient ou non été déléguées) ont été perdus, comme via un signal UNIX, une interface d'utilisateur graphique (GUI, *Graphical User Interface*) une fenêtre déroulante, etc. Voir au paragraphe 10.5 la discussion de ce que le client devrait faire des délégations non réclamées sur l'état de client. Pour la discussion sur la révocation de verrous, voir le paragraphe 8.5.

8.5 Révocation de verrous par le serveur

À tout moment, le serveur peut révoquer les verrous détenus par un client, et le client doit être prêt pour cela. Quand le client détecte que ses verrous ont été ou peuvent avoir été révoqués, il est responsable de la validation des informations d'état entre lui et le serveur. Valider l'état de verrouillage pour le client signifie qu'il doit vérifier ou réclamer l'état pour chaque verrou actuellement détenu.

La première occasion de révocation de verrou est au redémarrage du serveur. Noter que cela inclut des situations dans lesquelles les sessions sont persistantes et l'état de verrouillage est perdu. Dans cette classe d'instances, le client va recevoir une erreur (NFS4ERR_STALE_CLIENTID) sur une opération qui prend l'identifiant de client, généralement au titre d'une récupération en réponse à un problème avec la session en cours) et le client va procéder à la récupération normale de défaillance comme décrit au paragraphe 8.4.2.1.

La seconde occasion de révocation de verrou est l'incapacité de renouveler le prêt avant expiration, comme discuté au paragraphe 8.4.3. Bien que ceci soit considéré un événement rare ou inhabituel, le client doit être prêt à récupérer. Le serveur est responsable de la détermination des conséquences précises de l'expiration de prêt, d'informer le client de la portée de la décision de révocation du verrou. Le client utilise alors les informations d'état fournies par le serveur dans le résultat de SEQUENCE (champ sr_status_flags, voir le paragraphe 18.46.3) pour synchroniser son état de verrouillage avec celui du serveur, afin de récupérer.

La troisième occasion de révocation de verrou peut survenir par suite d'une révocation de verrous dans la période de prêt, soit à cause d'une intervention administrative, soit parce que un verrou rappelable (délégation ou disposition) n'a pas été retourné dans la période de prêt après avoir été rappelé. Bien que ces événements soient considérés être rares, ils sont possibles, et le client doit être prêt à les traiter. Quand un de ces événements se produit, le client découvre la situation par l'état retourné par l'opération SEQUENCE. Toute utilisation des identifiants d'état associés aux verrous révoqués durant la période de prêt va recevoir l'erreur NFS4ERR_ADMIN_REVOKED ou NFS4ERR_DELEG_REVOKED, comme approprié.

Dans toutes les situations dans lesquelles un sous ensemble de l'état de verrouillage peut avoir été révoqué, ce qui inclus tous les cas où l'état de verrouillage est révoqué dans la période de prêt, il appartient au client de déterminer quels verrous

ont été révoqués et ceux qui ne le sont pas. Il fait cela en utilisant l'opération TEST_STATEID sur l'ensemble approprié d'identifiants d'état. Une fois que l'ensemble des verrous révoqués a été déterminé, les applications peuvent en être notifiées, les identifiants d'état invalidés peuvent être libérés, et la révocation de verrou acquittée avec FREE_STATEID.

8.6 Prêts courts et longs

Quand on détermine la durée pour le prêt du serveur, le compromis usuel de prêt s'applique. Un prêt court est bon pour la récupération rapide du serveur au prix de plus d'opérations pour effectuer le renouvellement de prêt (quand il n'y a pas d'autres opérations durant la période pour affecter les renouvellement de prêt). Un long prêt est certainement plus doux pour les serveurs qui essayent de traiter de très grands nombres de clients. Le nombre de demandes supplémentaires pour effectuer le renouvellement de verrou chute en proportion inverse de la durée de prêt. L'inconvénient des longs prêts inclut la possibilité de récupération plus lente après certaines défaillances. Après une défaillance de serveur, une plus longue période de grâce peut être nécessaire quand certains clients ne réclament pas rapidement leurs verrous et font un RECLAIM_COMPLETE global. En cas de défaillance du client, la plus longue période pour l'expiration d'un prêt va forcer une plus longue attente des demandes en conflit.

Un long prêt est pratique si le serveur peut mémoriser l'état de prêt dans une mémorisation stable. À la récupération, le serveur peut reconstruire l'état de prêt à partir de sa mémorisation stable et continuer de fonctionner avec ses clients.

8.7 Horloges, délai de propagation, et calcul de l'expiration du prêt

Pour éviter d'avoir besoin de synchroniser les horloges, les temps de prêts sont accordés par le serveur comme des différences de temps. Cependant, il y a l'exigence que les horloges du client et du serveur ne dérivent pas excessivement sur la durée du prêt. Il y a aussi la question du délai de propagation à travers le réseau, qui pourrait facilement être de plusieurs centaines de millisecondes, ainsi que la possibilité que des demandes soient perdues et doivent être retransmises.

Pour prendre en compte le délai de propagation, le client devrait le soustraire des temps de prêt (par exemple, si le client estime le délai de propagation unidirectionnel à 200 millisecondes, alors il peut supposer que le prêt est déjà vieux de 200 millisecondes quand il l'obtient). De plus, il va falloir 200 autres millisecondes pour obtenir le retour d'une réponse au serveur. Donc le client doit envoyer une demande de renouvellement de prêt ou renvoyer des données au serveur au moins 400 millisecondes avant l'expiration du prêt. Si le délai de propagation varie pendant la vie du prêt (par exemple, si le client est sur un hôte mobile) le client va devoir continuellement soustraire l'augmentation du délai de propagation du temps de prêt.

La configuration de période de prêt du serveur devrait tenir compte de la distance sur le réseau des clients qui vont accéder aux ressources du serveur. On s'attend à ce que la période de prêt prenne en compte les délais de propagation du réseau et les autres facteurs de délai du réseau pour la population de clients. Comme le protocole ne permet pas une méthode automatique pour déterminer une période de prêt appropriée, l'administrateur du serveur peut devoir régler la période de prêt.

8.8 Infrastructure de verrouillage obsolète de NFSv4.0

Un certain nombre d'opérations et de champs parmi les opérations existantes n'ont plus de fonction dans NFSv4.1. D'une façon ou d'une autre, ces changements sont tous dus à la mise en œuvre de sessions qui fournissent le contexte de client et la sémantique de exactement une seule fois comme caractéristique de base du protocole, à part le verrouillage lui-même.

Les opérations NFSv4.0 suivantes NE DOIVENT PAS être mises en œuvre dans NFSv4.1. Le serveur DOIT retourner NFS4ERR_NOTSUPP si ces opérations sont trouvées dans un COMPOUND NFSv4.1.

* SETCLIENTID car sa fonction a été remplacée par EXCHANGE_ID.
* SETCLIENTID_CONFIRM car la confirmation d'identifiant de client se fait maintenant au moyen de CREATE_SESSION.
* OPEN_CONFIRM parce que les identifiants de séquence fondés sur le propriétaire d'état ont été remplacés par l'identifiant de séquence dans l'opération SEQUENCE.
* RELEASE_LOCKOWNER parce que les propriétaires de verrous avec des verrous non associés n'ont aucun état relatif à la séquence et peuvent donc être supprimés à volonté par le serveur.
* RENEW parce que chaque opération SEQUENCE pour une session cause le renouvellement du prêt, rendant une opération séparée superflue.

Il y a aussi un certain nombre de champs, présents dans des opérations existantes, relatives au verrouillage qui n'ont aucune

utilisation dans la version mineure 1. Ils étaient utilisés dans la version mineure 0 pour effectuer des fonctions maintenant fournies d'une façon différente.

* Les identifiants de séquence utilisés pour les demandes de séquence pour un propriétaire d'état donné et pour fournir la protection contre le ré-essai, maintenant fournie via les sessions.

* Les identifiants de client utilisés pour identifier le client associé à une demande. L'identification du client est maintenant disponible en utilisant l'identifiant de client associé à la session en cours, sans avoir besoin d'un champ d'identifiant de client explicite.

Ces vestiges de champs dans les opérations existantes n'ont pas de fonction dans NFSv4.1 et sont ignorés par le serveur. Noter que les identifiants de client dans les opérations nouvelles de NFSv4.1 (comme CREATE_SESSION et DESTROY_CLIENTID) ne sont pas ignorées.

9. Verrouillage de fichier et réservations partagées

Pour prendre en charge les réservations partagées Win32, il est nécessaire de fournir des opérations qui ouvrent ou créent atomiquement des fichiers. Avoir une opération séparée partagée/non partagée ne permettrait pas une mise en œuvre correcte de l'API Win32 de OpenFile. Afin de mettre correctement en œuvre la sémantique de partage, les mécanismes du précédent protocole NFS utilisés quand un fichier est ouvert ou créé (LOOKUP, CREATE, ACCESS) doivent être remplacés. Le protocole NFSv4.1 définit une opération OPEN qui est capable de chercher, créer, et verrouiller atomiquement un fichier sur le serveur.

9.1 Verrouillages d'ouvertures et de gamme d'octets

Il est supposé que la manipulation d'un verrou de gamme d'octets est rare comparée aux opérations READ et WRITE. Il est aussi supposé que les redémarrages du serveur et les partitions de réseau sont relativement rares. Donc, il est important que les opérations READ et WRITE aient un mécanisme léger pour indiquer si elles possèdent un verrou. Une opération LOCK contient les informations lourdes requises pour établir un verrou de gamme d'octets et définir de façon univoque le propriétaire du verrou.

9.1.1 Définition de propriétaire d'état

Quand il ouvre un fichier ou demande un verrou de gamme d'octets, le client doit spécifier un identifiant qui représente le propriétaire du verrou demandé. Cet identifiant est sous la forme d'un propriétaire d'état, représenté dans le protocole par un state_owner4, un dispositif opaque de longueur variable qui, quand il est enchaîné avec l'identifiant de client en cours, définit de façon univoque le propriétaire d'un verrou géré par le client. Ce peut être un identifiant de trame, un identifiant de processus, ou une autre valeur unique.

Les propriétaires d'ouverture ou les propriétaires de verrous de gamme d'octets sont des entités séparées et qui restent séparées même si les mêmes dispositifs opaques sont utilisés pour désigner les propriétaires de chacun. Le protocole distingue les propriétaires d'ouverture (représentés par des structures open_owner4) et les propriétaires de verrous (représentés par des structures lock_owner4).

Chaque ouverture est associée à un propriétaire d'ouverture spécifique tandis que chaque verrou de gamme d'octets est associé à un propriétaire de verrou et un propriétaire d'ouverture, ce dernier étant le propriétaire d'ouverture associé au fichier ouvert sous lequel l'opération LOCK a été faite. Les délégations et dispositions, par ailleurs, ne sont pas associées à un propriétaire spécifique mais sont associées au client comme un tout (identifié par un identifiant de client).

9.1.2 Utilisation de Stateid et de Locking

Toutes les opérations READ, WRITE, et SETATTR contiennent un identifiant d'état. Pour les besoins de ce paragraphe, les opérations SETATTR qui changent l'attribut de taille d'un fichier sont traitées comme si elles écrivaient la zone entre la nouvelle et l'ancienne taille (c'est-à-dire, la gamme d'octets tronquée ou ajoutée au fichier au moyen du SETATTR) même lorsque SETATTR n'est pas explicitement mentionné dans le texte. L'identifiant d'état passé à une de ces opérations doit être un qui représente une ouverture, un ensemble de verrous de gamme d'octets, ou une délégation, ou il peut être un identifiant d'état spécial représentant un accès anonyme ou l'identifiant d'état d'outrepassement spécial.

Si le propriétaire d'état effectue une opération READ ou WRITE dans une situation dans laquelle il a établi un verrou de gamme d'octets ou un partage de réservation sur le serveur (tout OPEN constitue un partage de réservation) l'identifiant d'état (précédemment retourné par le serveur) doit être utilisé pour indiquer quels verrous, y compris les deux verrous de gamme d'octets et de partage de réservations, sont détenus par le propriétaire d'état. Si aucun état n'est établi par le client, un verrou de gamme d'octets ou de partage de réservation, un identifiant d'état spécial pour état anonyme (zéro comme valeur pour "autre" et "seqid") est utilisé. (Voir au paragraphe 8.2.3 une description des identifiants d'état spéciaux en général.) Sans considération de si un identifiant d'état pour l'état anonyme ou un identifiant d'état retourné par le serveur est utilisé, si il y a un conflit de partage de réservation ou un verrou de gamme d'octets obligatoire détenu sur le fichier, le serveur DOIT refuser de servir l'opération READ ou WRITE.

Les partages de réservations sont établis par les opérations OPEN et par nature sont obligatoires en ce que quand le OPEN refuse les opérations READ ou WRITE, ce refus résulte en ce que ces opérations sont rejetées avec l'erreur NFS4ERR_LOCKED. Les verrous de gamme d'octets peuvent être mis en œuvre par le serveur comme obligatoires ou indicatifs, ou le choix de comportement obligatoire ou indicatif peut être déterminé par le serveur sur la base du fichier accédé (par exemple, certains serveurs fondés sur UNIX prennent en charge un "bit de verrou obligatoire" sur l'attribut mode tel que si il est établi, les verrous de gamme d'octets sont exigés sur le fichier avant que des entrées/sorties soient possibles). Quand les verrous de gamme d'octets sont indicatifs, ils empêchent seulement d'accorder des demandes de verrou en conflit et n'ont pas d'effet sur les READ ou WRITE. Les verrous de gamme d'octets obligatoires empêchent cependant les opérations d'entrée/sortie en conflit. Quand elles sont tentées, elles sont rejetées avec NFS4ERR_LOCKED. Quand le client obtient NFS4ERR_LOCKED sur un fichier pour lequel il sait qu'il a le partage de réservation approprié, il va devoir envoyer une opération LOCK sur la gamme d'octets du fichier qui inclut la gamme d'octets des entrées/sorties à effectuer, avec un champ Type de verrou approprié aux arguments de l'opération LOCK (c'est-à-dire, READ*_LT pour une opération READ, WRITE*_LT pour une opération WRITE).

Noter que pour les environnements UNIX qui prennent en charge le verrouillage obligatoire de gamme d'octets, la distinction entre verrouillage indicatif et obligatoire est subtile. En fait, les verrous de gamme d'octets indicatifs et obligatoires sont exactement les mêmes pour ce qui concerne les API et les exigences de mise en œuvre. Si l'attribut de verrou obligatoire est établi sur le fichier, le serveur vérifie si le propriétaire de verrou a un verrou de gamme d'octets partagé (READ_LT) ou exclusif (WRITE_LT) approprié sur la gamme d'octets qu'il souhaite lire ou sur laquelle il souhaite écrire. Si il n'y a pas de verrou approprié, le serveur vérifie si il y a un conflit de verrous (ce qui peut être fait en tentant d'acquérir le verrou en conflit au nom du propriétaire de verrou, et si cela réussit, libérer le verrou après que l'opération READ ou WRITE est faite) et si il y en a, le serveur retourne NFS4ERR_LOCKED.

Pour les environnements Windows, les verrous de gamme d'octets sont toujours obligatoires, donc le serveur vérifie toujours les verrous de gamme d'octets durant les demandes d'entrée/sortie.

Donc, l'opération LOCK n'a pas besoin de distinguer entre verrous de gamme d'octets indicatifs et obligatoires. C'est le traitement par le serveur des opérations READ et WRITE qui introduit la distinction.

Chaque identifiant d'état qui est validement passé à READ, WRITE, ou SETATTR, à l'exception des valeurs d'identifiant d'état spécial, définit un mode d'accès pour le fichier (c'est-à-dire, OPEN4_SHARE_ACCESS_READ, OPEN4_SHARE_ACCESS_WRITE, ou OPEN4_SHARE_ACCESS_BOTH).

* Pour les identifiants d'état associés aux ouvertures, c'est le mode défini par le OPEN original qui a causé l'allocation du statid OPEN et comme modifié par les OPEN et OPEN_DOWNGRADE suivants pour la même paire de propriétaire d'ouverture/fichier.

* Pour les identifiants d'état retournés par les opérations LOCK de gamme d'octets, le mode approprié est le mode d'accès pour l'identifiant d'état OPEN associé à l'ensemble de verrous représenté par l'identifiant d'état.

* Pour les identifiants d'état de délégation, le mode d'accès se fonde sur le type de délégation.

Quand une opération READ, WRITE, ou SETATTR (qui spécifie l'attribut de taille) est faite, l'opération est soumise à une confrontation au mode d'accès pour vérifier que l'opération est appropriée à l'identifiant d'état auquel l'opération est associée.

Dans le cas des opérations de type WRITE (c'est-à-dire, WRITE et SETATTR qui établissent la taille) le serveur DOIT vérifier que le mode d'accès permet l'écriture et DOIT retourner une erreur NFS4ERR_OPENMODE si il ne le fait pas. Dans le cas de READ, le serveur peut effectuer la vérification correspondante sur le mode d'accès, ou il peut choisir de permettre READ sur les OPEN pour OPEN4_SHARE_ACCESS_WRITE, pour s'accommoder des clients dont la mise en

œuvre de WRITE ne peut pas éviter de lire (par exemple, du fait de contraintes de mémoire tampon). Cependant, même si les READ sont permis dans ces circonstances, le serveur DOIT quand même vérifier les verrous en conflit avec le READ (par exemple, un autre OPEN a spécifié OPEN4_SHARE_DENY_READ ou OPEN4_SHARE_DENY_BOTH). Noter que un serveur qui applique la vérification de mode d'accès sur les READ n'a pas besoin de vérifier explicitement un conflit de partage de réservation car l'existence de OPEN pour OPEN4_SHARE_ACCESS_READ garantit qu'aucun partage de réservation en conflit ne peut exister.

L'identifiant d'état spécial d'outrepassement de READ (tous les bits de "autre" et "seqid" réglés à un) indique le désir d'outrepasser les vérifications de verrouillage. Le serveur PEUT permettre que des opérations READ outrepassent les vérifications de verrouillage chez le serveur, quand cet identifiant d'état spécial est utilisé. Cependant, les opérations WRITE avec cette valeur d'identifiant d'état spécial NE DOIVENT PAS outrepasser les vérifications de verrouillage et sont traitées exactement de la même façon que si un identifiant d'état spécial pour l'état anonyme était utilisé.

Un verrou ne peut pas être accordé lorsque une opération READ ou WRITE utilisant un des identifiants d'état spéciaux est effectuée et que la portée du verrou à accorder entrerait en conflit avec l'opération READ ou WRITE. Cela peut arriver quand :

* Un verrou de gamme d'octets obligatoire est demandé avec une gamme d'octets qui est en conflit avec la gamme d'octets de l'opération READ ou WRITE. Pour les besoins de ce paragraphe, un conflit survient quand un verrou partagé est demandé et qu'une opération WRITE est effectuée, ou quand un verrou exclusif est demandé et qu'une opération READ ou WRITE est effectuée.

* Une réservation partagée est demandée qui refuse la lecture et/ou l'écriture et que l'opération correspondante est en train d'être effectuée.

* Une délégation est à accorder et le type de délégation empêcherait l'opération d'entrée/sortie, c'est-à-dire, READ et WRITE serait en conflit avec une délégation OPEN_DELEGATE_WRITE et WRITE serait en conflit avec une délégation OPEN_DELEGATE_READ.

Quand un client détient une délégation, il a besoin de s'assurer que l'identifiant d'état envoyé porte l'association de l'opération à la délégation, pour éviter que la délégation soit inutilement rappelée. Quand l'identifiant d'état de délégation, un identifiant d'état open associé à cette délégation, ou un identifiant d'état représentant des verrous de gamme d'octets déduits d'un tel open est utilisé, le serveur sait que le READ, WRITE, ou SETATTR n'est pas en conflit avec la délégation mais est envoyé sous l'égide de la délégation. Bien qu'il soit possible au serveur de déterminer à partir de l'identifiant de client (via l'identifiant de session) que le client a en fait une délégation, le serveur n'est pas obligé de vérifier cela, donc l'utilisation d'un identifiant d'état spécial peut résulter en un rappel évitable de la délégation.

9.2 Gammes de verrous

Le protocole permet à un propriétaire de verrou de demander un verrou avec une gamme d'octets et ensuite mettre à niveau, dégrader, ou déverrouiller une sous gamme du verrou initial, ou une gamme d'octets qui se chevauche -- en totalité ou partiellement -- avec ce verrou initial ou avec une combinaison d'un ensemble de verrous existants pour le même propriétaire de verrou. Il est supposé que cela va être un type de demande peu courant. En tous cas, les serveurs ou systèmes de fichiers de serveur peuvent n'être pas capables de prendre en charge la sémantique de sous gammme de verrous. Au cas où un serveur reçoit une demande de verrouillage qui représente une sous gamme de l'état de verrouillage en cours pour le propriétaire de verrou, il est permis au serveur de retourner l'erreur NFS4ERR_LOCK_RANGE pour signifier qu'il ne prend pas en charge les opérations de sous gamme de verrous. Donc, le client devrait être prêt à recevoir cette erreur et, si c'est approprié, rapporter l'erreur à l'application demandeuse.

Il est déconseillé au client de combiner plusieurs gammes indépendantes de verrouillage qui se trouveraient être adjacentes dans une seule demande car le serveur peut ne pas prendre en charge les demandes de sous gammes pour des raisons relatives à la récupération de l'état de verrouillage de gamme d'octets en cas de défaillance du serveur. Comme discuté au paragraphe 8.4.2, le serveur peut employer durant la récupération certaines optimisations qui ne fonctionnent effectivement que quand le comportement du client durant la récupération de verrou est similaire au comportement de verrouillage du client avant la défaillance du serveur.

9.3 Mise à niveau et dégradation de verrous

Si un client a un verrou WRITE_LT sur une gamme d'octets, il peut demander une dégradation atomique du verrou en un

verrou READ_LT via l'opération LOCK, en réglant le type à READ_LT. Si le serveur prend en charge la dégradation atomique, la demande va réussir. Sinon, il va retourner NFS4ERR_LOCK_NOTSUPP. Le client devrait être prêt à recevoir cette erreur et, si c'est approprié, rapporter l'erreur à l'application demandeuse.

Si un client a un verrou READ_LT sur une gamme d'octets, il peut demander une mise à niveau atomique du verrou à un verrou WRITE_LT via l'opération LOCK en réglant le type à WRITE_LT ou WRITEW_LT. Si le serveur ne prend pas en charge la mise à niveau atomique, il va retourner NFS4ERR_LOCK_NOTSUPP. Si la mise à niveau peut être réalisée sans qu'il existe de conflit, la demande va réussir. Autrement, le serveur va retourner NFS4ERR_DENIED ou NFS4ERR_DEADLOCK. L'erreur NFS4ERR_DEADLOCK est retournée si le client a envoyé l'opération LOCK avec le type réglé à WRITEW_LT et si le serveur a détecté une impasse. Le client devrait être prêt à recevoir de telles erreurs et, si c'est approprié, à rapporter l'erreur à l'application demandeuse.

9.4 Valeurs de seqid d'identifiant d'état et verrouillages de gammes d'octets

Quand une opération LOCK ou LOCKU est effectuée, l'identifiant d'état retourné a la même valeur "autre" que l'identifiant d'état de l'argument, et une valeur de "seqid" qui est incrémentée (par rapport au statid de l'argument) pour refléter l'occurrence de l'opération LOCK ou LOCKU. Le serveur DOIT incrémenter la valeur du champ "seqid" chaque fois qu'il y a un changement de l'état de verrouillage de tout décalage d'octet comme décrit par tout verrou couvert par l'identifiant d'état. Un changement d'état de verrouillage inclut un changement de verrouillé à déverrouillé ou l'inverse ou un changement d'être verrouillé pour READ_LT en être verrouillé pour WRITE_LT ou l'inverse.

Quand il n'y a pas un tel changement, comme, par exemple, quand une gamme déjà verrouillée pour WRITE_LT est verrouillée à nouveau pour WRITE_LT, le serveur PEUT incrémenter la valeur de "seqid".

9.5 Questions de propriétaires d'ouvertures multiples

Quand le même fichier est ouvert par plusieurs propriétaires d'ouverture, un client va avoir plusieurs identifiants d'état OPEN pour ce fichier, chacun associé à un différent propriétaire d'ouverture. Dans ce cas, il peut y avoir plusieurs demandes LOCK et LOCKU pour le même propriétaire de verrou envoyées en utilisant les différents identifiants d'état OPEN, et donc une situation peut se produire dans laquelle il y a plusieurs identifiants d'état, chacun représentant des verrous de gamme d'octets sur le même fichier et détenus par le même propriétaire de verrou mais chacun associé à un propriétaire d'ouverture différent.

Dans une telle situation, l'état de verrouillage de chaque octet (c'est-à-dire, si il est verrouillé, le type du verrou READ_LT ou WRITE_LT, et le propriétaire de verrou détenant le verrou) DOIT refléter la dernière opération LOCK ou LOCKU faite pour le propriétaire de verrou en question, indépendamment de l'identifiant d'état par lequel la demande a été envoyée.

Quand un octet est verrouillé par le propriétaire du verrou en question, le propriétaire d'ouverture auquel ce verrou de gamme d'octets est alloué DEVRAIT être celui du propriétaire d'ouverture associé à l'identifiant d'état à travers lequel le dernier LOCK de cet octet a été fait. Quand il y a un changement du propriétaire d'ouverture associé aux verrous pour l'identifiant d'état à travers lequel un LOCK ou LOCKU a été fait, le champ "seqid" de l'identifiant d'état DOIT être incrémenté, même si le verrouillage, en termes de propriétaire du verrou n'a pas changé. Quand il y a un changement dans l'ensemble d'octets verrouillés associé à un identifiant d'état différent pour le même propriétaire de verrou, c'est-à-dire, associé à un propriétaire d'ouverture différent, la valeur de "seqid" pour ce identifiant d'état NE DOIT PAS être incrémentée.

9.6 Verrous bloquants

Certains clients exigent la prise en charge de verrous bloquants. Bien que NFSv4.1 fournisse un rappel quand un verrou précédemment indisponible devient disponible, cette caractéristique est FACULTATIVE et les clients peuvent ne pas dépendre de sa présence. Les clients doivent être prêts à interroger continuellement sur le verrou. Cela pose un problème d'équité. Deux types de verrou, READW_LT et WRITEW_LT, sont utilisés pour indiquer au serveur que le client demande un verrou bloquant. Quand le rappel n'est pas utilisé, le serveur devrait tenir une liste ordonnée de verrous bloquants en instance. Quand le verrou en conflit est libéré, le serveur peut attendre une durée égale à lease_time pour le premier client en attente pour redemander le verrou. Après l'expiration de la période de prêt, le verrou est alloué à la prochaine demande de client en attente. Il est exigé du client qu'il interroge à un intervalle suffisamment réduit pour qu'il soit probable qu'il acquière le verrou en temps utile. Le serveur n'est pas obligé de tenir une liste des verrous bloqués en instance car elle est utilisée pour augmenter l'équité et non pour corriger le fonctionnement. À cause de la nature non ordonnée de la récupération de pannes, mémoriser l'état de verrouillage dans une mémorisation stable serait nécessaire pour garantir un

accord ordonné des verrous bloquants.

Les serveurs peuvent aussi noter les types de verrou et le délai de retour du refus de la demande pour permettre du temps supplémentaire pour qu'un verrou en conflit soit libéré, permettant un retour réussi. De cette façon, les clients peuvent éviter la charge de fréquentes interrogations inutiles pour les verrous bloquants. Le serveur devrait veiller à la longueur du délai au cas où le client retransmettrait la demande.

Si un serveur reçoit une opération LOCK bloquante, la refuse, et reçoit ensuite une demande non bloquante pour le même verrou, qui est aussi refusé, alors il devrait supprimer le verrou en question de sa liste des verrous bloquants en instance. Les clients devraient utiliser une telle demande non bloquante pour indiquer au serveur que c'est la dernière fois qu'ils entendent interroger pour le verrou, comme cela peut se produire quand le processus qui demande le verrou est interrompu. C'est une politesse pour le serveur, pour l'empêcher d'attendre inutilement une période de prêt avant d'accorder d'autres opérations LOCK. Cependant, les clients ne sont pas obligés de faire cela, et les serveurs ne doivent pas compter qu'ils vont le faire. Aussi, les clients doivent être prêts à la possibilité que cette demande de verrouillage finale soit acceptée.

Quand un serveur indique, via le fanion OPEN4_RESULT_MAY_NOTIFY_LOCK, que des rappels CB_NOTIFY_LOCK pourraient être faits pour le fichier ouvert en cours, le client devrait le noter, mais, comme c'est une indication, ne peut pas compter qu'un CB_NOTIFY_LOCK sera toujours fait. Un client peut raisonnablement réduire la fréquence de ses interrogations sur un verrou refusé, car la plus grande latence qui pourrait se produire va probablement être éliminée par un rappel rapide, mais il doit quand même interroger. Quand il reçoit un CB_NOTIFY_LOCK, il devrait rapidement essayer d'obtenir le verrou, mais il devrait être conscient que d'autres clients peuvent être en train d'interroger et que le serveur n'a pas d'obligation de réserver le verrou pour ce client particulier.

9.7 Partage de réservations

Un partage de réservation est un mécanisme pour contrôler l'accès à un fichier. C'est un mécanisme séparé et indépendant du mécanisme de verrouillage de gamme d'octets. Quand un client ouvre un fichier, il envoie une opération OPEN au serveur, spécifiant le type de d'accès requis (READ, WRITE, ou BOTH) et le type d'accès pour refuser les autres (OPEN4_SHARE_DENY_NONE, OPEN4_SHARE_DENY_READ, OPEN4_SHARE_DENY_WRITE, ou OPEN4_SHARE_DENY_BOTH). Si le OPEN échoue, le client va faire échouer la demande d'ouverture de l'application.

Pseudo-code de définition de la sémantique :

```
si (request.access == 0) {
  retourner (NFS4ERR_INVAL)
} autrement {
  si ((request.access & file_state.deny)) || (request.deny & file_state.access)) {
    retourner (NFS4ERR_SHARE_DENIED)
}
retourner (NFS4ERR_OK) ;
```

Quand on fait cette vérification de partage de réservation sur OPEN, l'état de fichier en cours utilisé dans l'algorithme inclut des bits qui reflètent toutes les ouvertures en cours, y compris celles pour le propriétaire d'ouverture qui fait la nouvelle demande OPEN.

Les constantes utilisées pour les opérations OPEN et OPEN_DOWNGRADE pour les champs d'accès et de refus sont les suivantes :

```
const OPEN4_SHARE_ACCESS_READ   = 0x00000001 ;
const OPEN4_SHARE_ACCESS_WRITE = 0x00000002 ;
const OPEN4_SHARE_ACCESS_BOTH   = 0x00000003 ;
const OPEN4_SHARE_DENY_NONE      = 0x00000000 ;
const OPEN4_SHARE_DENY_READ      = 0x00000001 ;
const OPEN4_SHARE_DENY_WRITE     = 0x00000002 ;
const OPEN4_SHARE_DENY_BOTH      = 0x00000003 ;
```

9.8 Opérations OPEN/CLOSE

Pour fournir une sémantique de partage correcte, un client DOIT utiliser l'opération OPEN pour obtenir la bride de fichier

initiale et indiquer l'accès désiré et quels accès, si il en est, refuser. Même si le client entend utiliser un identifiant d'état spécial pour l'état anonyme ou l'outrepassement de READ, il doit quand même obtenir la bride de fichier pour le fichier régulier avec l'opération OPEN afin que la sémantique de partage appropriée puisse être appliquée. Les clients qui n'ont pas de mode de refus incorporé dans leurs interfaces de programmation pour ouvrir un fichier devraient demander un mode de refus de OPEN4_SHARE_DENY_NONE.

L'opération OPEN avec le fanion CREATE englobe aussi l'opération CREATE pour les fichiers réguliers comme utilisé dans les précédentes versions du protocole NFS. Cela permet qu'une création avec un partage soit faite atomiquement.

L'opération CLOSE supprime tout partage de réservation détenu par le propriétaire d'ouverture sur ce fichier. Si des verrous de gamme d'octets sont détenus, le client DEVRAIT libérer tous les verrous avant d'envoyer une opération CLOSE. Le serveur PEUT libérer tous les verrous en instance sur CLOSE, mais certains serveurs peuvent ne pas prendre en charge le CLOSE d'un fichier qui détient encore des verrous de gamme d'octets. Le serveur DOIT retourner un échec, NFS4ERR_LOCKS_HELD, si des verrous existent après le CLOSE.

L'opération LOOKUP va retourner une bride de fichier sans établir aucun état de verrouillage sur le serveur. Sans un identifiant d'état valide, le serveur va supposer que le client a le moindre accès. Par exemple, si un client ouvrait un fichier avec OPEN4_SHARE_DENY_BOTH et si un autre client accède au fichier via une bride de fichier obtenue par LOOKUP, le second client pourrait seulement lire le fichier en utilisant l'identifiant d'état spécial "outrepassement de lecture" *(read bypass)*. Le second client ne pourrait pas écrire du tout sur le fichier parce que il n'aurait pas un identifiant d'état valide provenant de OPEN et l'identifiant d'état spécial anonyme n'aurait pas de permission d'accès.

9.9 Mise à niveau et dégradation de OPEN

Quand un OPEN est fait pour un fichier et que le propriétaire d'ouverture pour lequel le OPEN est fait a déjà le fichier ouvert, le résultat est de mettre à niveau l'état de fichier ouvert conservé sur le serveur pour inclure les bits d'accès et de refus spécifiés par le nouveau OPEN ainsi que ceux pour le OPEN existant. Le résultat est que il y a un fichier ouvert, en ce qui concerne le protocole, et il inclut l'union des bits d'accès et de refus pour toutes les demandes OPEN achevées. Le OPEN est représenté par un seul identifiant d'état dont la valeur "autre" correspond à celle du open original, et dont la valeur de "seqid" est incrémentée pour refléter l'occurrence de la mise à niveau. L'incrément est exigé dans les cas où la "mise à niveau" ne résulte en aucun changement au mode open (par exemple, un OPEN est fait pour lire quand le fichier ouvert existant est ouvert pour OPEN4_SHARE_ACCESS_BOTH). Un seul CLOSE va être fait pour rétablir les effets des deux OPEN. Le client peut utiliser l'identifiant d'état retourné par le OPEN qui effectue la mise à niveau ou avec un identifiant d'état partageant le même champ "autre" et un seqid de zéro, bien qu'il faille faire attention aux mises à niveau qui se produisent pendant que le CLOSE est en cours. Noter que le client, quand il envoie le OPEN, ne peut pas savoir que le même fichier est en fait en cours d'ouverture. Ce qui précède s'applique seulement si les deux OPEN résultent en ce que l'objet ouvert soit désigné par la même bride de fichier.

Quand le serveur choisit d'exporter plusieurs brides de fichier correspondant au même objet de fichier et retourne des brides de fichier différentes sur deux OPEN différents du même objet de fichier, le serveur NE DOIT PAS faire un "OU" des bits d'accès et de refus et fusionner les deux fichiers ouverts. À la place, le serveur doit maintenir séparés les OPEN avec des identifiant d'état séparés et exigera des CLOSE séparés pour les libérer.

Quand plusieurs fichiers ouverts sur le client sont fusionnés en un seul objet de fichier OPEN sur le serveur, la fermeture d'un des fichiers ouverts (sur le client) peut nécessiter de changer l'état d'accès et de refus du fichier ouvert sur le serveur. C'est parce que l'union des bits d'accès et de refus des ouvertures restantes peut être plus petite (c'est-à-dire, un sous ensemble) que précédemment. L'opération OPEN_DOWNGRADE est utilisée pour faire le changement nécessaire et le client devrait l'utiliser pour mettre à jour le serveur afin que les demandes de partage de réservation des autres clients soient traitées de façon appropriée. L'identifiant d'état retourné a le même champ "autre" que celui passé au serveur. La valeur "seqid" dans l'identifiant d'état retourné DOIT être incrémentée, même dans des situations où il n'y a pas de changement aux bits d'accès et de refus pour le fichier.

9.10 OPEN parallèles

À la différence du cas de NFSv4.0, dans lequel les opérations OPEN pour le même propriétaire d'ouverture sont par nature en série à cause du seqid fondé sur le propriétaire, plusieurs OPEN pour le même propriétaire d'ouverture peuvent être faits en parallèle. Quand les clients font cela, ils peuvent rencontrer des situations dans lesquelles, à cause de l'existence de liaisons physiques, deux opérations OPEN peuvent se trouver ouvrir le même fichier, avec un OPEN ultérieur effectué qui est une mise à niveau du premier, ce fait étant seulement visible au client une fois les opérations achevées.

Dans cette situation, les clients peuvent déterminer l'ordre dans lequel les OPEN ont été effectués en examinant les identifiants d'état retournés par les OPEN. Les identifiants d'état qui partagent une valeur commune du champ "autre" peuvent être reconnus comme ayant ouvert le même fichier, avec l'ordre des opérations déterminable à partir de l'ordre des champs "seqid", modulo un possible retour à zéro du champ de 32 bits.

Quand il existe une possibilité que le client envoie en parallèle plusieurs OPEN pour le même propriétaire d'ouverture, il se peut que survienne une mise à niveau de l'ouverture sans que le client sache à l'avance que cela pourrait arriver. À cause de cette possibilité, les CLOSE et OPEN_DOWNGRADE devraient généralement être envoyés avec un seqid non à zéro dans l'identifiant d'état, pour éviter la possibilité que le changement d'état associé à une mise à niveau d'ouverture soit perdu par inadvertance.

9.11 Réclamation de verrous Open et de gamme d'octets

Des formes spéciales d'opérations LOCK et OPEN sont fournies quand il est nécessaire de rétablir des verrous de gamme d'octets ou d'ouverture après une défaillance du serveur.

* Pour réclamer des ouvertures existantes, une opération OPEN est effectuée en utilisant un CLAIM_PREVIOUS. Parce que le client, dans ce type de situation, aura déjà ouvert le fichier et aura la bride de fichier du fichier cible, cette opération exige que la bride de fichier courante soit celle du fichier cible, plutôt que d'un répertoire, et qu'aucun nom de fichier ne soit spécifié.

* Pour réclamer des verrous de gamme d'octets, une opération LOCK avec le paramètre reclaim réglé à vrai est utilisée.

Les réclamations d'ouvertures associées à des délégations sont discutées au paragraphe 10.2.1.

10. Mise en antémémoire côté client

La mise en antémémoire de données côté client, des attributs de fichier, et des noms de fichiers est essentielle pour fournir de bonnes performances avec le protocole NFS. Assurer la cohérence d'une antémémoire distribuée est un problème difficile, et les précédentes versions du protocole NFS ne l'ont pas tenté. À la place, plusieurs techniques de mise en œuvre de client NFS ont été utilisées pour réduire les problèmes que le manque de cohérence pose aux utilisateurs. Ces techniques n'ont pas été clairement définies par les spécifications antérieures du protocole, et ce qui est un comportement valide ou invalide du client n'est souvent pas clair.

Le protocole NFSv4.1 utilise de nombreuses techniques similaires à celles qui ont été utilisées dans les précédentes versions du protocole. Le protocole NFSv4.1 ne fournit pas la cohérence des antémémoires distribuées. Cependant, il définit un ensemble plus limité de garanties de mise en antémémoire pour permettre que les verrous et le partage de réservation soient utilisés sans interférence destructive de la part de la mise en antémémoire côté client.

De plus, le protocole NFSv4.1 introduit un mécanisme de délégation, qui permet que de nombreuses décisions normalement prises par le serveur soient faites localement par les clients. Ce mécanisme fournit une prise en charge efficace de cas courants où le partage est peu fréquent ou lorsque le partage est en lecture seule.

10.1 Défis de performances pour la mise en antémémoire côté client

Les techniques de mise en antémémoire utilisées dans les précédentes versions du protocole NFS ont réussi à assurer de bonnes performances. Cependant, plusieurs défis d'adaptabilité peuvent survenir quand ces techniques sont utilisées avec de très grands nombres de clients. Ceci est particulièrement vrai quand les clients sont répartis géographiquement, ce qui augmente classiquement la latence pour les demandes de revalidation d'antémémoire.

Les précédentes versions du protocole NFS répétaient leurs demandes de validation d'antémémoire de données au moment de l'ouverture du fichier. Ce comportement peut avoir de sérieux inconvénients pour les performances. Un cas courant est celui où un fichier est accédé par seulement un client. Donc, le partage est peu fréquent.

Dans ce cas, les références répétées au serveur pour trouver qu'aucun conflit n'existe sont coûteuses. Une meilleure option à l'égard des performances est de permettre à un client qui ouvre de façon répétée un fichier de le faire sans référence au serveur. Ceci est fait jusqu'à ce que des opérations potentiellement en conflit venant d'un autre client se produisent.

Une situation similaire se produit en connexion avec un verrouillage de gamme d'octets. L'envoi d'opérations LOCK et LOCKU ainsi que des opérations READ et WRITE nécessaires pour rendre cohérente la mise en antémémoire de données avec la sémantique de verrouillage (voir le paragraphe 10.3.2) peut limiter sévèrement les performances. Quand le verrouillage est utilisé pour fournir la protection contre des conflits peu fréquents, une lourde pénalité est subie. Cette pénalité peut décourager l'utilisation du verrouillage de gamme d'octets par les applications.

Le protocole NFSv4.1 fournit des stratégies plus agressives de mise en antémémoire avec les objectifs de conception suivants :

* Compatibilité avec une large gamme de sémantiques de serveurs.
* Fournir les mêmes avantages de mise en antémémoire que les précédentes versions du protocole NFS quand il est incapable de prendre en charge le modèle plus agressif.
* Les exigences de mise en antémémoire agressive sont organisées de telle sorte qu'une large portion des avantages peut être obtenue même quand toutes les exigences ne peuvent être satisfaites.

Les exigences appropriées pour le serveur sont discutées plus loin dans les paragraphes qui couvrent les formes spécifiques de mise en antémémoire (voir le paragraphe 10.4).

10.2 Délégation et rappels

La délégation rappelable des responsabilités de serveur pour un fichier à un client améliore les performances en évitant des demandes répétées au serveur en l'absence de conflit inter clients. Avec l'utilisation d'un "rappel" RPC du serveur au client, un serveur rappelle les responsabilités déléguées quand un autre client s'engage dans un partage d'un fichier délégué.

Une délégation est passée du serveur au client, spécifiant l'objet et le type de la délégation. Il y a différents types de délégations, mais chaque type contient un identifiant d'état utilisé pour représenter la délégation quand on effectue des opérations qui dépendent de la délégation. Cet identifiant d'état est similaire à ceux associés aux verrous et aux partages de réservation mais en diffère en ce que l'identifiant d'état pour une délégation est associé à un identifiant de client et peut être utilisé au nom de tous les propriétaires d'ouverture pour le client en question. Une délégation est faite au client comme un tout et aucun processus spécifique ou trame de contrôle en son sein.

Le canal de retour est établi par CREATE_SESSION et BIND_CONN_TO_SESSION, et il est exigé du client qu'il le maintienne. Parce que le canal de retour peut être mort, même temporairement, un fonctionnement correct du protocole n'en dépend pas. L'essai préliminaire de la fonction de canal de retour au moyen d'une procédure CB_COMPOUND avec une seule opération, CB_SEQUENCE, peut être utilisé pour vérifier la continuité du canal de retour. Un serveur évite de déléguer des responsabilités jusqu'à ce qu'il ait déterminé que le canal de retour existe. Parce que accorder une délégation est toujours conditionnel sur l'absence d'accès en conflit, les clients NE DOIVENT PAS supposer qu'une délégation va être accordée et ils DOIVENT toujours être prêts à des OPEN, WANT_DELEGATION, et GET_DIR_DELEGATION à traiter sans qu'aucune délégation ne soit accordée.

À la différence des verrous, une opération par un second client sur un fichier délégué va causer le rappel par le serveur d'une délégation par un rappel. Pour les opérations individuelles, on va décrire, dans une mise en œuvre, quand de telles opérations doivent effectuer un rappel. On devrait cependant noter un certain nombre de points.

* Le serveur est libre de rappeler une délégation chaque fois qu'il sent que s'est souhaitable et peut le faire même si aucune opération exigeant un rappel n'est faite.

* Les opérations faites en dehors du protocole NFSv4.1, dues, par exemple, à l'accès par d'autres protocoles, ou par un accès local, doivent aussi résulter en un rappel de délégation quand elles font des changements analogues aux données du système de fichiers. Ce qui est crucial est si le changement invaliderait les garanties fournies par la délégation. Quand c'est possible, la délégation doit être rappelée et DOIT être retournée ou révoquée avant de permettre la poursuite de l'opération.

* La sémantique du système de fichiers est cruciale pour définir quand le rappel de délégation est exigé. Si un changement particulier au sein d'une mise en œuvre spécifique cause un changement à un attribut de fichier, alors le rappel de délégation est exigé, si cette opération a été spécifiquement mentionnée comme exigeant un rappel de délégation. Là encore, ce qui est critique est si les garanties fournies dans la délégation ont été invalidées.

En dépit de ces avertissements, les paragraphes de mise en œuvre pour un certain nombre d'opérations décrivent des situations dans lesquelles le rappel de délégation serait exigé dans certaines circonstances courantes :

* pour GETATTR, voir le paragraphe 18.7.4,
* pour OPEN, voir le paragraphe 18.16.4,
* pour READ, voir le paragraphe 18.22.4,
* pour REMOVE, voir le paragraphe 18.25.4,
* pour RENAME, voir le paragraphe 18.26.4,
* pour SETATTR, voir le paragraphe 18.30.4,
* pour WRITE, voir le paragraphe 18.32.4.

Sur un rappel, le client qui détient la délégation doit purger tout état modifié (comme des données modifiées) au serveur et retourner la délégation. La demande en conflit ne va pas être traitée jusqu'à ce que le rappel soit terminé. Le rappel est considéré comme terminé quand le client retourne la délégation ou quand le serveur épuise son temps d'attente de retour de la délégation et révoque la délégation par suite de son expiration. Dans l'intervalle, le serveur va retarder la réponse aux demandes en conflit ou leur répondre avec NFS4ERR_DELAY. Suite à la résolution du rappel, le serveur a les informations nécessaire pour accorder ou refuser la demande du second client.

Au moment où le client reçoit un rappel de délégation, il peut y avoir un état substantiel qui ait besoin d'être purgé sur le serveur. Donc, le serveur devrait permettre un temps suffisant pour que la délégation soit retournée car cela peut impliquer de nombreux RPC au serveur. Si le serveur est capable de déterminer que le client est rapide à purger l'état sur le serveur par suite du rappel, le serveur peut étendre la durée usuelle permise pour un rappel. Cependant, le temps alloué à l'achèvement du rappel ne devrait pas être illimité.

Un exemple de cela est quand la responsabilité de traiter les ouvertures sur un fichier donné est déléguée à un client (voir le paragraphe 10.4). Le serveur ne va pas savoir quelles ouvertures sont effectuées sur le client. Sans cette connaissance, le serveur va être incapable de déterminer si les états d'accès et de refus pour le fichier permettent une ouverture particulière tant que la délégation pour le fichier n'a pas été retournée.

La défaillance d'un client ou une partition de réseau peut résulter en l'échec de réponse à un retour de rappel. Dans ce cas, le serveur va révoquer la délégation, ce qui à son tout va rendre inutile toute modification d'état encore chez le client.

10.2.1 Récupération de délégation

La récupération de délégation doit traiter trois situations :
* le redémarrage de client,
* le redémarrage de serveur,
* la partition de réseau (complète ou seulement du canal de retour).

En cas de redémarrage du client, l'échec du renouvellement du prêt va résulter en la révocation des verrous de gamme d'octets et du partage de réservation. Les délégations, cependant, peuvent être traitées un peu différemment.

Il va y avoir des situations dans lesquelles les délégations vont devoir être rétablies après le redémarrage d'un client. La raison en est que le client peut avoir des données de fichier mémorisées localement et que ces données étaient associées aux délégations détenues précédemment. Le client va devoir rétablir l'état de fichier approprié sur le serveur.

Pour permettre ce type de récupération de client, le serveur PEUT étendre la période de récupération de délégation au delà de l'expiration normale de période de prêt. Cela implique que les demandes provenant d'autres clients qui sont en conflit avec ces délégations vont devoir attendre. Parce que le processus normal de rappel peut exiger un temps significatif pour que le client purge l'état changé sur le serveur, les autres clients doivent être prêts pour les délais qui surviennent à cause de délégations en conflit. Ce plus long intervalle va augmenter la fenêtre pour que les clients redémarrent et consultent la mémorisation afin que les délégations puissent être réclamées. Pour les délégations OPEN, elles sont réclamées en utilisant OPEN avec un type de CLAIM_DELEGATE_PREV ou CLAIM_DELEG_PREV_FH (voir les paragraphes 10.5 et 18.16 pour la discussion de délégation OPEN et les détails de OPEN, respectivement).

Un serveur PEUT prendre en charge les réclamations de types CLAIM_DELEGATE_PREV et CLAIM_DELEG_PREV_FH, et si il le fait, il NE DOIT PAS supprimer les délégations sur un CREATE_SESSION qui confirme un identifiant de client créé par un EXCHANGE_ID. À la place, le serveur DOIT, pour une durée non inférieure à celle de la valeur de l'attribut lease_time, maintenir les délégations du client pour donner au client le temps d'envoyer les demandes CLAIM_DELEGATE_PREV et/ou CLAIM_DELEG_PREV_FH. Le serveur qui prend en charge CLAIM_DELEGATE_PREV et/ou CLAIM_DELEG_PREV_FH DOIT prendre en charge l'opération DELEGPURGE.

Quand le serveur redémarre, les délégations sont réclamées (en utilisant l'opération OPEN avec CLAIM_PREVIOUS)

d'une façon similaire à celle des verrous de gamme d'octets et du partage de réservation. Cependant, il y a une légère différence de signification. Dans le cas normal, si le serveur décide qu'une délégation ne devrait pas être accordée, il effectue l'action demandée (par exemple, OPEN) sans accorder de délégation. Pour une réclamation, le serveur accorde la délégation mais une désignation spéciale est appliquée afin que le client traite la délégation comme ayant été accordée mais rappelée par le serveur. À cause de cela, le client a le devoir d'écrire tout l'état modifié au serveur et ensuite de retourner la délégation. Ce processus de traitement de la réclamation de délégation regroupe trois principes du protocole NFSv4.1 :

* Sur une réclamation, un client qui fait rapport de ressources qui lui ont été allouées par une instance de serveur antérieure doit se voir accorder ces ressources.
* Le serveur a une autorité incontestable pour déterminer si des délégations sont à accorder et, une fois qu'elles sont accordées, si elles doivent être poursuivies.
* On ne devrait pas dépendre de l'utilisation de rappels avant que le client ait prouvé sa capacité de les recevoir.

Quand un client a besoin de réclamer une délégation et qu'il n'y a pas d'ouverture associée, le client peut utiliser la variante CLAIM_PREVIOUS de l'opération WANT_DELEGATION. Cependant, comme le serveur n'est pas obligé de prendre cette opération en charge, une solution de remplacement est de réclamer via un OPEN factice et la délégation utilisant un OPEN de type CLAIM_PREVIOUS. L'ouverture de fichier factice peut être réalisée en utilisant un CLOSE pour rétablir l'état original à réclamer, une délégation sans ouverture associée.

Quand un client a plus d'une seule ouverture associée à une délégation, l'état pour ces ouvertures supplémentaires peut être établi en utilisant des opérations OPEN de type CLAIM_DELEGATE_CUR. Quand elles sont utilisées pour établir des ouvertures associées à des délégations réclamées, le serveur DOIT les permettre quand elles sont faites dans la période de grâce.

Quand une partition de réseau se produit, les délégations sont sujettes à libération par le serveur quand la période de renouvellement de prêt expire. C'est similaire au comportement pour les verrous et le partage de réservation. Pour les délégations, le serveur peut cependant étendre la période dans laquelle les demandes en conflit sont écartées. Finalement, l'occurrence d'une demande en conflit provenant d'un autre client va causer la révocation de la délégation. Une perte du canal de retour (par exemple, par un changement ultérieur de configuration du réseau) va avoir le même effet. Une demande de rappel va échouer et la révocation de la délégation va s'ensuivre.

Un client découvre normalement la révocation d'une délégation quand il utilise un identifiant d'état associé à une délégation et reçoit une des erreurs NFS4ERR_EXPIRED, NFS4ERR_ADMIN_REVOKED, ou NFS4ERR_DELEG_REVOKED. Il peut aussi découvrir la révocation de délégation après un redémarrage de client quand il tente de réclamer une délégation et reçoit cette même erreur. Noter que dans le cas d'une délégation OPEN_DELEGATE_WRITE révoquée, il y a des problèmes parce que des données peuvent avoir été modifiées par le client dont la délégation est révoquée et aussi séparément par d'autres clients. Voir au paragraphe 10.5.1 la discussion de ces questions. Noter aussi que quand des délégations sont révoquées, les informations sur la délégation révoquée vont être écrites par le serveur dans une mémorisation stable (comme décrit au paragraphe 8.4.3). C'est fait pour traiter le cas d'un redémarrage du serveur après la révocation d'une délégation mais avant que le client qui détient la délégation révoquée soit notifié de la révocation.

10.3 Mise en antémémoire de données

Quand les applications partagent l'accès à un ensemble de fichiers, elles doivent être mises en œuvre de façon à prendre en compte la possibilité de conflits d'accès avec une autre application. Ceci est vrai si les applications en question s'exécutent sur différents clients ou résident sur le même client.

Les partages de réservations et les verrous de gamme d'octets sont les facilités que le protocole NFSv4.1 fournit pour permettre aux applications de coordonner l'accès en utilisant des facilitiés d'exclusion mutuelles. La mise en antémémoire des données du protocole NFSv4.1 doit être mise en œuvre de façon telle qu'elle n'invalide pas les hypothèses dont dépendent l'utilisation de ces facilités.

10.3.1 Mise en antémémoire et OPEN

Afin d'éviter d'invalider les hypothèses de partage sur lesquelles s'appuient les applications, les clients NFSv4.1 ne devraient pas fournir les données en antémémoire aux applications ou les modifier au nom d'une application quand il ne serait pas valide d'obtenir ou modifier ces mêmes données via une opération READ ou WRITE.

De plus, en l'absence d'une délégation OPEN (voir le paragraphe 10.4) deux règles supplémentaires s'appliquent. Noter que ces règles sont observées en pratique par de nombreux clients NFSv3.

* D'abord, les données en antémémoire présentes chez un client doivent être revalidées après un OPEN. Revalider signifie que le client va chercher l'attribut de changement chez le serveur, le comparer à l'attribut de changement en antémémoire, et si ils diffèrent, déclarer que les données en antémémoire (ainsi que les attributs en antémémoire) sont invalides. C'est pour s'assurer que les données pour le fichier objet de l'OPEN sont encore reflétées correctement dans l'antémémoire du client. Cette validation doit être faite au moins quand l'opération OPEN du client inclut un refus de OPEN4_SHARE_DENY_WRITE ou OPEN4_SHARE_DENY_BOTH, terminant donc une période dans laquelle d'autres clients peuvent avoir eu l'opportunité d'ouvrir le fichier avec un accès OPEN4_SHARE_ACCESS_WRITE / OPEN4_SHARE_ACCESS_BOTH. Les clients peuvent choisir de faire la revalidation plus souvent (c'est-à-dire, à des OPEN qui spécifient un mode de refus de OPEN4_SHARE_DENY_NONE) pour faire un parallèle avec la pratique du protocole NFSv3 au bénéfice des utilisateurs qui supposent ce degré de revalidation d'antémémoire.

  Comme l'attribut de changement est mis à jour pour les modifications de données et métadonnées, certaines mises en œuvre de client peuvent être tentées d'utiliser l'attribut time_modify et non l'attribut de changement pour valider les données en antémémoire, afin que des changements de métadonnées parasites n'invalident pas des données propres. Cette approche ne doit pas être mise en œuvre. Il est garanti que l'attribut de changement change pour chaque mise à jour du fichier, tandis que le changement de time_modify n'est garanti qu'à la granularité de l'attribut time_delta. L'utilisation par la logique de validation d'antémémoire de données du client de time_modify et non de changement encoure le risque que le client marque à tort comme valides des données périmées. Donc, toute approche de validation d'antémémoire par le client DOIT inclure l'utilisation de l'attribut de changement.

* Ensuite, les données modifiées doivent être purgées sur le serveur avant de clore un fichier ouvert pour OPEN4_SHARE_ACCESS_WRITE. Ceci est complémentaire de la première règle. Si les données ne sont pas purgées à CLOSE, la revalidation faite après que le client ouvre un fichier est incapable de réaliser cet objectif. L'autre aspect de la purge des données avant la clôture est que les données doivent être engagées sur une mémorisation stable, sur le serveur, avant que l'opération CLOSE soit demandée par le client. Dans le cas d'un redémarrage de serveur et d'un fichier clos, il se peut qu'il ne soit pas possible de retransmettre les données à écrire dans le fichier, d'où cette exigence.

10.3.2 Mise en antémémoire de données et verrouillage de fichier

Pour les applications qui choisissent d'utiliser le verrouillage de gamme d'octets plutôt que le partage de réservation pour exclure l'accès non conforme au fichier, il y a un ensemble analogue de contraintes qui s'appliquent à la mise en antémémoire de données côté client. Ces règles ne sont effectives que si le verrouillage de gamme d'octets est utilisé d'une façon qui corresponde à une façon équivalente aux opérations READ et WRITE réelles exécutées. Ceci s'oppose au verrouillage de gamme d'octets qui est fondé sur une pure convention. Par exemple, il est possible de manipuler un fichier de deux méga octets en divisant le fichier en deux gammes de un méga octets et en protégeant l'accès aux deux gammes d'octets par des verrous de gamme d'octets sur les octets zéro et un. Un verrou WRITE_LT sur l'octet zéro du fichier représenterait la bonne façon d'effectuer les opérations READ et WRITE sur la première gamme d'octets. Un verrou WRITE_LT sur l'octet un du fichier représenterait la bonne façon d'effectuer les opérations READ et WRITE sur la seconde gamme d'octets. Tant que toutes les applications qui manipulent le fichier respectent cette convention, elles vont fonctionner sur un système de fichiers local. Cependant, cela peut ne pas fonctionner avec le protocole NFSv4.1 à moins que les clients s'abstiennent de mettre les données en antémémoire.

Les règles pour la mise en antémémoire des données dans l'environnement de verrouillage de gamme d'octets sont :

* D'abord, quand un client obtient un verrou de gamme d'octets pour une gamme d'octets particulière, l'antémémoire de données correspondant à cette gamme d'octets (si une antémémoire de données existe) doit être revalidée. Si l'attribut de changement indique que le fichier peut avoir été mis à jour depuis que les données en antémémoire ont été obtenues, le client doit purger ou invalider les données en antémémoire pour la gamme d'octets nouvellement verrouillée. Un client pourrait choisir d'invalider toutes les données en antémémoire non modifiées qu'il a pour le fichier, mais la seule exigence pour un fonctionnement correct est d'invalider toutes les données dans la gamme d'octets nouvellement verrouillée.

* Ensuite, avant de libérer un verrou WRITE_LT pour une gamme d'octets, toutes les données modifiées pour cette gamme d'octets doivent être purgées sur le serveur. Les données modifiées doivent aussi être écrites dans une mémorisation stable.

Noter que la purge des données sur le serveur et l'invalidation des données en antémémoire doit refléter les gammes d'octets réelles verrouillées ou déverrouillées. Les arrondir au dessus ou en dessous pour refléter les limites de bloc de l'antémémoire du client va causer des problèmes si ce n'est pas fait avec soin. Par exemple, écrire sur un bloc modifié

quand seulement la moitié de ce bloc est dans une zone déverrouillées peut causer une modification invalide de la gamme d'octets en dehors de la zone déverrouillée. Cela peut à son tour faire partie d'une gamme d'octets verrouillée par un autre client. Les clients peuvent éviter cette situation en effectuant de façon synchronisée des portions d'opérations WRITE qui chevauchent cette portion (initiale ou finale) qui n'est pas un bloc complet. De même, invalider une zone verrouillée qui n'est pas un nombre entier de blocs complets de mémoire tampon va exiger que le client lise un ou deux blocs partiels provenant du serveur si la procédure de revalidation montre que les données que le client possède peuvent n'être pas valides.

Les données qui sont écrites sur le serveur comme prérequis du déverrouillage d'une gamme d'octets doivent être écrites, chez le serveur, dans une mémorisation stable. Le client peut réaliser cela soit avec des écritures synchronisées, soit en faisant suivre les écritures asynchrones d'une opération COMMIT. Ceci est exigé parce que la retransmission des données modifiées après un redémarrage de serveur pourraient être en conflit avec un verrou détenu par un autre client.

Une mise en œuvre de client peut choisir de s'accommoder des applications qui utilisent le verrouillage de gamme d'octets de façons non standard (par exemple, en utilisant un verrou de gamme d'octets comme un sémaphore global) en purgeant chez le serveur plus de données sur un LOCKU que ce qui est couvert par la gamme verrouillée. Cela peut inclure des données modifiées dans des fichiers autres que celui pour lequel les déverrouillages sont faits. Dans ce cas, le client ne doit pas interférer avec des applications dont les READ et WRITE sont faits seulement dans les limites des verrous de gamme d'octets que détient l'application. Par exemple, une application verrouille un seul octet d'un fichier et procède à l'écriture de ce seul octet. Un client qui a choisi de traiter un LOCKU en purgeant toutes les données modifiées sur le serveur pourrait validement écrire ce seul octet en réponse à une opération LOCKU sans relation. Cependant, il ne serait pas valide d'écrire le bloc entier dans lequel ce seul octet écrit était situé car il inclut une zone qui n'est pas verrouillée et pourrait être verrouillée par un autre client. Les mises en œuvre de client peuvent éviter ce problème en divisant les fichiers avec des données modifiées en ceux pour lesquels toutes les modifications sont faites aux zones couvertes par un verrou de gamme d'octets approprié et ceux pour lesquels il y a des modifications non couvertes par un verrou de gamme d'octets. Toutes les écritures faites pour la première classe de fichiers doivent ne pas inclure de zones non verrouillées et donc non modifiées chez le client.

10.3.3 Mise en antémémoire de données et verrouillage de fichier obligatoire

La mise en antémémoire de données côté client doit respecter le verrouillage obligatoire de gamme d'octets quand il est en vigueur. La présence de verrouillage obligatoire de gamme d'octets pour un fichier donné est indiquée quand le client reçoit en retour un NFS4ERR_LOCKED provenant d'une opération READ ou WRITE sur un fichier pour lequel il a un partage de réservation approprié. Quand le verrouillage obligatoire est en vigueur pour un fichier, le client doit vérifier qu'il y a un verrou de gamme d'octets approprié pour les données lues ou écrites. Si un verrou de gamme d'octets existe pour la gamme lue ou écrite, le client peut satisfaire la demande en utilisant l'antémémoire validée du client. Si un verrou de gamme d'octets approprié n'est pas détenu pour la gamme d'octets lus ou écrits, la demande de lecture ou écriture ne doit pas être satisfaite par l'antémémoire du client et la demande doit être envoyée au serveur pour traitement. Quand une demande de lecture ou écriture se recouvre partiellement avec une gamme d'octets verrouillée, la demande devrait être subdivisée en plusieurs parties avec chaque gamme d'octets (verrouillée ou non) traitée de façon appropriée.

10.3.4 Mise en antémémoire de données et identité de fichier

Quand les clients mettent des données en antémémoire, les données du fichier doivent être organisées conformément à l'objet de système de fichiers auquel les données appartiennent. Pour les clients NFSv3, la pratique normale a été de supposer pour les besoins de la mise en antémémoire que des brides de fichier distinctes représentent des objets de système de fichiers distincts. Le client a alors le choix d'organiser et maintenir l'antémémoire de données sur cette base.

Dans le protocole NFSv4.1, il y a maintenant la possibilité d'avoir des variantes significatives du modèle de "une bride de fichier par objet" parce que une bride de fichier peut être construite sur la base du nom de chemin de l'objet. Donc, les clients ont besoin d'une méthode fiable pour déterminer si deux brides de fichier désignent le même objet de système de fichiers. Si les clients supposent simplement que toutes les brides de fichier distinctes notent des objets distincts et procèdent à la mise en antémémoire des données sur cette base, des incohérences de mise en antémémoire se produiraient entre les objets distincts côté client qui se transposent en le même objet côté serveur.

En fournissant une méthode pour différencier les brides de fichier, le protocole NFSv4.1 diminue une régression potentielle par rapport au protocole NFSv3. Sans cette méthode, des incohérences de mise en antémémoire au sein du même client pourrait survenir, et cela n'était pas présent dans les précédentes versions du protocole NFS. Noter qu'il est possible d'avoir de telles incohérences avec des applications qui s'exécutent sur plusieurs clients, mais ce n'est pas le problème traité ici.

Pour les besoins de la mise en antémémoire des données, les étapes suivantes permettent à un client NFSv4.1 de déterminer si deux brides de fichier distinctes notent le même objet côté serveur :

* Si le GETATTR dirigé sur deux brides de fichier retourne des valeurs différentes de l'attribut fsid, alors les brides de fichier représentent des objets distincts.

* Si le GETATTR pour tout fichier avec un fsid qui correspond au fsid des deux brides de fichier en question retourne un attribut unique_handles avec une valeur de VRAI, alors les deux objets sont distincts.

* Si le GETATTR dirigé sur les deux brides de fichier ne retourne pas l'attribut fileid pour les deux brides, alors on ne peut pas déterminer si les deux objets sont les mêmes. Donc, les opérations qui dépendent de cette connaissance (par exemple, la mise en antémémoire des données côté client) ne peuvent pas être faites de façon fiable. Noter que si GETATTR ne retourne pas l'attribut fileid pour les deux brides de fichier, il ne va le retourner pour aucune des brides de fichier, car le fsid pour les deux brides de fichier est le même.

* Si le GETATTR dirigé sur les deux brides de fichier retourne des valeurs différentes pour l'attribut fileid, alors ce sont des objets distincts.

* Autrement, c'est le même objet.

10.4 Délégation ouverte

Lors de l'ouverture d'un fichier, le serveur peut déléguer le traitement ultérieur des ouvertures et clôtures de ce fichier au client qui ouvre. Une telle délégation est rappelable car les circonstances qui ont permis la délégation sont sujettes à changement. En particulier, si le serveur reçoit d'un autre client un OPEN en conflit, le serveur doit rappeler la délégation avant de décider si le OPEN provenant de l'autre client peut être accordé. Faire une délégation est à la discrétion du serveur, et les clients ne devraient pas supposer qu'un OPEN particulier va ou non résulter en une délégation de OPEN. Voici un ensemble typique de conditions que les serveurs pourraient utiliser pour décider si un OPEN devrait être délégué :

* Le client doit être capable de répondre aux demandes de rappel du serveur. Si un canal de retour a été établi, le serveur va envoyer une demande CB_COMPOUND, contenant une seule opération, CB_SEQUENCE, pour un essai de disponibilité du canal de retour.
* Le client doit avoir répondu de façon appropriée aux rappels précédents.
* Il ne doit pas y avoir de OPEN en cours en conflit avec la délégation demandée.
* Il ne devrait pas y avoir de délégation courante en conflit avec la délégation demandée.
* La probabilité de futures demandes d'ouverture en conflit devrait être faible sur la base de l'historique récent du fichier.
* L'existence de toute sémantique spécifique du serveur de OPEN/CLOSE qui rendrait le traitement demandé incompatible avec le traitement prescrit que le client délégué va appliquer (voir ci-dessous).

Il y a deux types de délégations OPEN : OPEN_DELEGATE_READ et OPEN_DELEGATE_WRITE. Une délégation OPEN_DELEGATE_READ permet à un client de traiter, de lui-même, les demandes d'ouverture d'un fichier pour une lecture qui ne dénie pas l'accès OPEN4_SHARE_ACCESS_READ aux autres. Plusieurs délégations OPEN_DELEGATE_READ peuvent être en instance simultanément et n'être pas en conflit. Une délégation OPEN_DELEGATE_WRITE permet au client de traiter, de lui-même, toutes les ouvertures. Une seule délégation OPEN_DELEGATE_WRITE peut exister pour un fichier donné à un instant donné, et elle n'est cohérente avec aucune délégation OPEN_DELEGATE_READ.

Quand un client a une délégation OPEN_DELEGATE_READ, il est assuré que ni le contenu, les attributs (à l'exception de time_access) ni les noms de toutes liaisons au fichier ne vont changer sans qu'il le sache, tant qu'il détient la délégation. Quand un client a une délégation OPEN_DELEGATE_WRITE, il peut modifier en local les données du fichier car aucun autre client ne va accéder aux données du fichier. Le client qui détient une délégation OPEN_DELEGATE_WRITE peut seulement affecter localement les attributs de fichier qui sont intimement connectés aux données du fichier : taille, changement, heure d'accès, heure des métadonnées, et heure de modification. Tous les autres attributs doivent être reflétés sur le serveur.

Quand un client a une délégation OPEN, il n'a pas besoin d'envoyer des OPEN ou CLOSE au serveur. À la place, le client peut mettre à jour en interne l'état approprié. Pour une délégation OPEN_DELEGATE_READ, les ouvertures qui ne peuvent pas être traitées localement (les ouvertures qui sont pour OPEN4_SHARE_ACCESS_WRITE / OPEN4_SHARE_ACCESS_BOTH ou qui refusent l'accès OPEN4_SHARE_ACCESS_READ) doivent être envoyées au serveur.

Quand une délégation OPEN est faite, la réponse au OPEN contient une structure de délégation OPEN qui spécifie :
* le type de délégation OPEN (OPEN_DELEGATE_READ ou OPEN_DELEGATE_WRITE)
* les informations de limitation d'espace pour contrôler la purge de données sur une fermeture (délégation OPEN_DELEGATE_WRITE seulement; voir le paragraphe 10.4.1)
* un nfsace4 spécifiant les permissions de lecture et d'écriture
* un identifiant d'état pour représenter la délégation.

L'identifiant d'état de délégation est séparé et distinct de l'identifiant d'état pour le OPEN propre. L'identifiant d'état standard, à la différence de l'identifiant d'état de délégation, est associé à un propriétaire de verrou particulier et va continuer d'être valide après que la délégation est rappelée et que le fichier reste ouvert.

Quand une demande interne au client est faite pour ouvrir un fichier et qu'une délégation OPEN est en vigueur, elle va être acceptée ou rejetée seulement sur la base des conditions suivantes. Toute exigence pour d'autres vérifications à faire par le délégué devrait résulter en le refus de la délégation OPEN afin que les vérifications puissent être faites par le serveur lui-même.

* Les bits d'accès et refus pour la demande et le fichier comme décrit au paragraphe 9.7.

* Les permissions de lecture et écriture comme déterminé ci-dessous.

Le nfsace4 passé avec la délégation peut être utilisé pour éviter des appels ACCESS fréquents. La vérification de permission devrait être comme suit :

* Si le nfsace4 indique que l'ouverture peut être faite, alors elle devrait être accordée sans référence au serveur.

* Si le nfsace4 indique que l'ouverture ne peut pas être faite, alors une demande ACCESS doit être envoyée au serveur pour obtenir la réponse définitive.

Le serveur peut retourner un nfsace4 plus restrictif que l'ACL réelle du fichier. Cela inclut un nfsace4 qui spécifie le refus de tout accès. Noter que certaines pratiques courantes comme la transposition de la "racine" traditionnelle d'utilisateur en l'utilisateur "personne" (voir le paragraphe 5.9) peuvent rendre incorrect de retourner l'ACL réelle du fichier dans la réponse de délégation.

L'utilisation d'une délégation avec diverses autres formes de mise en antémémoire crée la possibilité qu'aucune authentification et autorisation de serveur ne soient jamais effectuées pour un certain utilisateur car toutes les demandes de l'utilisateur pouvaient être satisfaites localement. Lorsque le client dépend du serveur pour l'authentification et l'autorisation, le client devrait être sûr que l'authentification et l'autorisation se produisent pour chaque utilisateur par l'utilisation de l'opération ACCESS. Ce devrait être le cas même si une opération ACCESS ne serait pas exigée autrement. Comme mentionné auparavant, le serveur peut appliquer une authentification fréquente en retournant un nfsace4 refusant tout accès avec chaque délégation OPEN.

10.4.1 Délégation ouverte et mise en antémémoire de données

Une délégation OPEN permet d'éliminer beaucoup des frais généraux de message associés à l'ouverture et à la fermeture des fichiers. Une ouverture quand une délégation OPEN est en vigueur n'exige pas qu'un message de validation soit envoyé au serveur. La résistance continue de la délégation "OPEN_DELEGATE_READ" fournit la garantie qu'aucun OPEN pour OPEN4_SHARE_ACCESS_WRITE/OPEN4_SHARE_ACCESS_BOTH, et donc aucune écriture, ne s'est produit. De même, quand on ferme un fichier ouvert pour OPEN4_SHARE_ACCESS_WRITE/OPEN4_SHARE_ACCESS_BOTH et si une délégation OPEN_DELEGATE_WRITE est en vigueur, les données écrites n'ont pas à l'être sur le serveur avant que la délégation OPEN soit rappelée. La résistance continue de la délégation OPEN fournit la garantie qu'aucune ouverture, et donc aucun READ ou WRITE, n'a été fait par un autre client.

Pour les besoins de délégation OPEN, les READ et WRITE faits sans un OPEN sont traités comme les équivalents fonctionnels d'un type correspondant de OPEN. Bien qu'un client NE DEVRAIT PAS utiliser des identifiant d'état spéciaux quand une ouverture existe, le traitement de délégation sur le serveur peut utiliser l'identifiant de client associé à la session en cours pour déterminer si l'opération a été faite par le détenteur de la délégation (dans ce cas, aucun rappel n'est nécessaire) ou par un autre client (dans ce cas, la délégation doit être rappelée et les entrées/sorties non traitées jusqu'à ce que la délégation soit retournée ou révoquée).

Avec les délégations, un client est capable d'éviter d'écrire des données sur le serveur quand la clôture d'un fichier est traitée. L'invocation du système de clôture de fichier est le moment habituel où le client est notifié d'un manque de mémorisation stable pour les données du fichier modifié générées par l'application. À la clôture, les données du fichier sont écrites au serveur et, par la comptabilité normale, le serveur est capable de déterminer si l'espace disponible du système de fichiers pour les données a été dépassé (c'est-à-dire, le serveur retourne NFS4ERR_NOSPC ou NFS4ERR_DQUOT). Cette comptabilité inclut des quotas. L'introduction de délégations exige qu'une méthode de remplacement soit en place pour que le même type de communication se produise entre client et serveur.

Dans la réponse de délégation, le serveur fournit la limite de la taille du fichier ou le nombre de blocs modifiés et la taille de bloc associée. Le serveur doit s'assurer que le client va être capable d'écrire les données modifiées sur le serveur d'une taille égale à celle fournie dans la délégation originale. Le serveur doit donner cette assurance pour toutes les délégations en instance. Donc, le serveur doit être attentif dans sa gestion de l'espace disponible pour les données nouvelles ou modifiées, en prenant en compte l'espace disponible du système de fichiers et tous les quotas applicables. Le serveur peut rappeler des délégations par suite de la gestion de l'espace disponible du système de fichiers. Le client devrait s'en tenir aux limites d'espace d'état du serveur pour les délégations. Si le client excède les limites déclarées pour la délégation, le comportement du serveur est indéfini.

Sur la base des conditions, quotas, ou espace disponible du système de fichiers du serveur, le serveur peut accorder des délégations OPEN_DELEGATE_WRITE avec des limitations d'espace très restrictives. Les limitations peuvent être définies d'une façon qui force toujours la purge des données modifiées à la fermeture du serveur.

À l'égard de l'authentification, la purge des données modifiées sur le serveur après un CLOSE peut être problématique. Par exemple, l'utilisateur de l'application peut avoir déconnecté le client, et des accréditifs d'authentification non expirés peuvent ne pas être présents. Dans ce cas, le client peut avoir besoin de faire particulièrement attention à s'assurer que des accréditifs locaux non expirés vont bien être disponibles. Cela peut être accompli en surveillant l'heure d'expiration des accréditifs et en les purgeant bien avant leur expiration ou en faisant des copies privées des accréditifs pour s'assurer de leur disponibilité quand nécessaire.

10.4.2 Délégation ouverte et verrouillages de fichiers

Quand un client détient une délégation OPEN_DELEGATE_WRITE, les opérations de verrouillage sont effectuées en local. Cela inclut celles requises pour le verrouillage obligatoire de gamme d'octets. Cela peut être fait parce que la délégation implique qu'il peut y avoir des verrous non conflictuels. De même, toutes les revalidations qui seraient normalement associées à l'obtention des verrous et la purge des données associées à la libération des verrous n'ont pas à être faites.

Quand un client détient une délégation OPEN_DELEGATE_READ, les opérations de verrouillage ne sont pas effectuées en local. Toutes les opérations de verrouillage, y compris celles qui demandent des verrous non exclusifs, sont envoyées au serveur pour résolution.

10.4.3 Traitement de CB_GETATTR

Le serveur doit employer un traitement spécial pour un GETATTR lorsque la cible est un fichier qui a une délégation OPEN_DELEGATE_WRITE en vigueur. La raison en est que le client qui détient la délégation OPEN_DELEGATE_WRITE peut avoir modifié les données, et le serveur a besoin de refléter ce changement au second client qui a soumis le GETATTR. Donc, le client qui détient la délégation OPEN_DELEGATE_WRITE doit être interrogé. Le serveur va utiliser l'opération CB_GETATTR. Les seuls attributs que le serveur peut fiablement interroger via CB_GETATTR sont size et change.

Comme CB_GETATTR est utilisé pour satisfaire la demande GETATTR d'un autre client, le serveur a seulement besoin de savoir si le client qui détient la délégation a une version modifiée du fichier. Si la copie du client du fichier délégué n'est pas modifiée (données ou taille) le serveur peut satisfaire la demande GETATTR du second client à partir des attributs mémorisés localement au serveur. Si le fichier est modifié, le serveur a seulement besoin de connaître cet état modifié. Si le serveur détermine que le fichier est actuellement modifié, il va répondre au GETATTR du second client comme si le fichier avait été modifié localement au serveur.

Comme la forme de l'attribut de changement est déterminée par le serveur et est opaque au client, le client et le serveur doivent s'accorder sur une méthode pour communiquer l'état modifié du fichier. Pour l'attribut de taille, le client va rapporter sa vue actuelle de la taille du fichier. Pour l'attribut de changement, le traitement est plus implicite.

Pour le client, les étapes suivantes vont être suivies quand il reçoit une délégation OPEN_DELEGATE_WRITE :

* La valeur de l'attribut de changement va être obtenue du serveur et mise en antémémoire. Appelons c cette valeur.
* Le client va créer une valeur supérieure à c qui va être utilisée pour communiquer que ces données modifiées sont détenues chez le client. Représentons cette valeur par d.
* Quand le client est interrogé via CB_GETATTR sur l'attribut de changement, il vérifie pour voir si il détient des données modifiées. Si le fichier est modifié, la valeur d est retournée comme valeur de l'attribut de changement. Si ce fichier n'est pas modifié actuellement, le client retourne la valeur c comme attribut de changement.

Pour la simplicité de mise en œuvre, le client PEUT pour chaque CB_GETATTR retourner la même valeur d. C'est vrai même si, entre des opérations CB_GETATTR successives, le client modifie encore les données du fichier ou les métadonnées dans son antémémoire. Le client peut retourner la même valeur parce que la seule exigence est que le client soit capable d'indiquer au serveur que le client détient des données modifiées. Donc, la valeur de d peut toujours être c + 1.

Bien que l'attribut de changement soit opaque pour le client en ce sens qu'il n'a pas d'idée des unités de temps, si il en est, avec lesquelles le serveur compte le changement, il n'est pas opaque en ce que le client doit le traiter comme un entier non signé, et que le serveur doit être capable de voir le résultat des changements du client à cet entier. Donc, le serveur DOIT coder l'attribut de changement dans l'ordre du réseau quand il l'envoie au client. Le client DOIT le décoder de l'ordre du réseau dans son ordre natif quand il le reçoit, et le client DOIT le coder dans l'ordre du réseau quand il l'envoie au serveur. Pour cette raison, le changement est défini comme un entier non signé plutôt que comme un dispositif opaque d'octets.

Pour le serveur, les étapes suivantes vont être effectuées quand il fournit une délégation OPEN_DELEGATE_WRITE :

* Lorsque il fournit une délégation OPEN_DELEGATE_WRITE, le serveur va mettre en antémémoire une copie de l'attribut de changement dans la structure de données qu'il utilise pour enregistrer la délégation. Représentons cette valeur par sc.

* Quand un second client envoie une opération GETATTR sur le même fichier au serveur, le serveur obtient du premier client l'attribut de changement. Disons que cette valeur est cc.

* Si la valeur cc est égale à sc, le fichier n'est pas modifié et le serveur retourne les valeurs courantes pour change, time_metadata, et time_modify (par exemple) au second client.

* Si la valeur cc n'est pas égale à sc, le fichier est actuellement modifié chez le premier client et très probablement va être modifié ultérieurement chez le serveur. Le serveur utilise alors son heure courante pour construire des valeurs d'attribut pour time_metadata et time_modify. Une nouvelle valeur de sc, qu'on va appeler nsc, est calculée par le serveur, telle que nsc ≥ sc + 1. Le serveur retourne alors les valeurs construites de time_metadata, time_modify, et nsc au demandeur. Le serveur remplace sc dans l'enregistrement de délégation par nsc. Pour prévenir la possibilité que time_modify, time_metadata, et change paraissent reculer (ce qui se produirait si le client qui détient la délégation échoue à écrire ses données modifiées sur le serveur avant que la délégation soit révoquée ou retournée) le serveur DEVRAIT mettre à jour l'enregistrement des métadonnées du fichier avec les valeur d'attribut construites. Pour des performances raisonnables, l'engagement des valeurs d'attribut construites dans une mémorisation stable est FACULTATIF.

Comme exposé précédemment dans cette section, le client PEUT retourner la même valeur cc sur les appels CB_GETATTR suivants, même si le fichier a été déjà modifié dans l'antémémoire du client entre les appels CB_GETATTR successifs. Donc, le serveur doit supposer que le fichier a déjà été encore modifié, et DOIT veiller à s'assurer que le nouveau nsc qu'il construit et retourne est supérieur au précédent nsc retourné. Un exemple de mise en œuvre d'enregistrement de délégation qui satisferait cela est d'inclure un champ booléen (appelons le "modifié") réglé à FAUX quand la délégation est accordée, et une valeur sc réglée à l'heure de l'octroi de la valeur de l'attribut de changement. Le champ modifié serait réglé à VRAI la première fois que cc != sc, et resterait VRAI jusqu'à ce que la délégation soit retournée ou révoquée. Le traitement de construction de nsc, time_modify, et time_metadata utiliserait le pseudo code suivant :

```
si (!modifié) {
   faire CB_GETATTR pour changement et taille ;
   si (cc != sc)
      modifié = VRAI;
} autrement {
   faire CB_GETATTR pour taille ;
}
```

```
si (modifié) {
   sc = sc + 1;
   time_modify = time_metadata = heure_actuelle ;
   mettre à jour sc, time_modify, time_metadata dans les métadonnées du fichier ;
}
```

Cela retournerait au client (qui a envoyé GETATTR) les attributs qu'il demande, mais s'assure que taille vient de ce que CB_GETATTR a retourné. Le serveur ne mettrait pas à jour les métadonnées du fichier avec la taille modifiée du client.

Dans le cas où la taille de l'attribut du fichier est différente de la valeur actuelle du serveur, le serveur traite cela comme une modification sans considération de la valeur de l'attribut de changement restitué via CB_GETATTR et répond au second client comme dans la dernière étape.

Cette méthodologie résout les questions de différences d'horloge entre client et serveur et autres scénarios où l'utilisation de CB_GETATTR échoue.

On devrait noter que le serveur n'a aucune obligation d'utiliser CB_GETATTR, et donc le serveur PEUT simplement rappeler la délégation pour éviter son utilisation.

10.4.4 Rappel d'une délégation ouverte

Les événements suivants nécessitent le rappel d'une délégation OPEN :

* demande OPEN potentiellement conflictuelle (ou une opération READ ou WRITE faite avec un identifiant d'état spécial)
* SETATTR envoyé par un autre client
* demande REMOVE pour le fichier
* demande RENAME pour le fichier comme source ou cible du RENAME

Si un RENAME d'un répertoire dans le chemin conduisant au fichier résulte en le rappel d'une délégation OPEN dépend de la sémantique du système de fichiers du serveur. Si ce système de fichiers refuse de tels RENAME quand un fichier est ouvert, le rappel doit être effectué pour déterminer si le fichier en question est, en fait, ouvert.

En plus des situations ci-dessus, le serveur peut choisir de rappeler les délégations OPEN à tout moment si des contraintes de ressources invitent à le faire. Les clients devraient toujours être prêts à la possibilité de rappel.

Quand un client reçoit un rappel pour une délégation OPEN, il doit mettre à jour l'état sur le serveur avant de retourner la délégation. Cette même mise à jour doit être faite chaque fois qu'un client choisit volontairement de retourner une délégation. Les éléments d'état suivants doivent être traités :

* Si le fichier associé à la délégation n'est plus ouvert et si aucune opération CLOSE précédente n'a été envoyée au serveur, une opération CLOSE doit être envoyée au serveur.

* Si un fichier a d'autres références open au client, alors les opérations OPEN doivent être envoyées au serveur. Les identifiants d'état appropriés vont être fournis par le serveur pour être utilisés ensuite par le client car l'identifiant d'état de délégation ne va plus être valide. Ces demandes OPEN sont faites avec le type de réclamation de CLAIM_DELEGATE_CUR. Cela va permettre la présentation de l'identifiant d'état de délégation afin que le client puisse établir les droits appropriés d'effectuer le OPEN. (Voir le paragraphe 18.16, qui décrit l'opération OPEN, pour les détails.)

* Si des verrous de gamme d'octets sont accordés, les opérations LOCK correspondantes doivent être effectuées. Cela s'applique seulement au cas de la délégation OPEN_DELEGATE_WRITE.

* Pour une délégation OPEN_DELEGATE_WRITE, si au moment du rappel le fichier n'est pas ouvert pour OPEN4_SHARE_ACCESS_WRITE/OPEN4_SHARE_ACCESS_BOTH, toutes les données modifiées pour le fichier doivent être purgées sur le serveur. Si la délégation n'avait pas existé, le client aurait fait cette purge de données avant l'opération CLOSE.

* Pour une délégation OPEN_DELEGATE_WRITE, quand un fichier est encore ouvert au moment du rappel, toutes les données modifiées pour le fichier doivent être purgées sur le serveur.

* Avec la délégation OPEN_DELEGATE_WRITE en place, il est possible que le fichier ait été tronqué durant la délégation. Par exemple, la troncature pourrait s'être produite par suite d'un OPEN UNCHECKED avec une valeur d'attribut de taille de zéro. Donc, si une troncature du fichier s'est produite et si cette opération n'a pas été propagée au serveur, la troncature doit se produire avant l'écriture de données modifiées sur le serveur.

Dans le cas d'une délégation OPEN_DELEGATE_WRITE, le verrouillage de gamme d'octets impose des exigences supplémentaires. Pour maintenir précisément l'invariant associé, il est exigé de purger toutes les données modifiées dans toute gamme d'octets pour laquelle un verrou WRITE_LT a été libéré pendant que la délégation OPEN_DELEGATE_WRITE était en vigueur. Cependant, parce que la délégation OPEN_DELEGATE_WRITE implique qu'il n'y a pas d'autre verrouillage par d'autres clients, une mise en œuvre plus simple est de purger toutes les données modifiées pour le fichier (comme décrit juste précédemment) si un verrou WRITE_LT a été libéré pendant que la délégation OPEN_DELEGATE_WRITE était en vigueur.

Une mise en œuvre n'a pas besoin d'attendre le rappel de la délégation (ou la décision de retourner volontairement une délégation) pour effectuer une des actions ci-dessus, si des considérations de mise en œuvre (par exemple, des contraintes de disponibilité de ressources) le rendent désirable. Généralement cependant, le fait que l'état OPEN réel du fichier puisse continuer de changer fait qu'il ne vaut pas la peine d'envoyer des informations sur les ouvertures et clôture au serveur, sauf au titre du retour de délégation. Une exception est quand le client n'a plus d'ouvertures internes du fichier. Dans ce cas, l'envoi d'un CLOSE est utile parce que cela réduit l'utilisation de ressources chez le client et le serveur. Sans considération des choix du client sur la programmation de ces actions, toutes doivent être effectuées avant que la délégation soit retournée, y compris (quand applicable) la clôture qui correspond à l'OPEN qui a résulté en la délégation. Ces actions peuvent être effectuées soit dans les demandes précédentes, soit dans les opérations précédentes dans la même demande COMPOUND.

10.4.5 Clients qui échouent à honorer les rappels de délégation

Un client peut échouer à répondre à un rappel pour diverses raisons, comme une défaillance du canal de retour du serveur au client. Le client peut être ignorant de la défaillance du canal de retour. Cette ignorance pourrait résulter en ce que le client découvre longtemps après la défaillance que sa délégation a été révoquée, et qu'un autre client a modifié les données pour lesquelles le client avait une délégation. Ceci est particulièrement un problème pour le client qui détient une délégation OPEN_DELEGATE_WRITE.

Les bits d'état retournés par les opérations SEQUENCE aident à fournir une autre façon d'informer le client des problèmes concernant l'état du canal de retour et des délégations rappelées. Quand le canal de retour n'est pas disponible, le serveur retourne le bit d'état SEQ4_STATUS_CB_PATH_DOWN sur les opérations SEQUENCE. Le client peut réagir en tentant de rétablir le canal de retour et en retournant des objets rappelables si un canal de retour ne peut pas réussir à être rétabli.

Que le canal de retour fonctionne ou non, il se peut que la délégation rappelée ne soit pas retournée. Noter que le prêt du client pourrait quand même être renouvelé, même si la délégation rappelée n'est pas retournée. Dans cette situation, les serveurs DEVRAIENT révoquer les délégations qui ne sont pas retournées dans une période égale à la période de prêt. Cette période devrait donner au client le temps de noter l'état du canal de retour défaillant et de rétablir le canal de retour.

Quand les délégations sont révoquées, le serveur va retourner avec le bit d'état SEQ4_STATUS_RECALLABLE_STATE_REVOKED établi sur les opérations SEQUENCE suivantes. Le client devrait noter cela et utiliser alors TEST_STATEID pour trouver quelles délégations ont été révoquées.

10.4.6 Révocation de délégation

Au moment où une délégation est révoquée, si il y a des ouvertures associées sur le client, ces ouvertures peuvent ou non être révoquées. Si aucun verrou de gamme d'octets ou ouverture n'est accordée qui soit incohérente avec l'ouverture existante, l'identifiant d'état pour l'ouverture peut rester valide et être déconnecté de la délégation révoquée, tout comme se serait le cas si la délégation avait été retournée.

Par exemple, si un OPEN pour OPEN4_SHARE_ACCESS_BOTH avec un refus de OPEN4_SHARE_DENY_NONE est associé à la délégation, accorder un autre OPEN à un client différent va révoquer la délégation mais n'a pas besoin de révoquer le OPEN, car les deux OPEN sont cohérents l'un avec l'autre. Par ailleurs, si un OPEN refusant l'accès en écriture est accordé, alors le OPEN existant doit être révoqué.

Quand des ouvertures et/ou des verrous sont révoqués, les applications qui détiennent ces ouvertures ou verrous doivent en

être notifiées. Cette notification se produit généralement en retournant des erreurs pour les opérations READ/WRITE ou quand une clôture est tentée pour le fichier ouvert.

Si aucune ouverture n'existe pour le fichier au moment où la délégation est révoquée, alors la notification de la révocation est inutile. Cependant, si il y a des données modifiées présentes chez le client pour le fichier, l'utilisateur de l'application devrait en être notifié. Malheureusement, il peut être impossible de le notifier à l'utilisateur car des applications actives peuvent n'être pas présentes chez le client. Voir au paragraphe 10.5.1 des détails supplémentaires.

10.4.7 Délégations via WANT_DELEGATION

En plus de fournir des délégations au titre de la réponse aux opérations OPEN, les serveurs PEUVENT fournir des délégations séparées des ouvertures, via l'opération FACULTATIVE WANT_DELEGATION. Cela permet aux délégations d'être obtenues à l'avance d'un OPEN qui pourrait en bénéficier, pour les objets qui ne sont pas une cible valide de OPEN, ou de traiter les cas où une délégation a été rappelée et où le client veut faire une tentative pour la rétablir si l'absence d'utilisation par d'autres clients le permet.

L'opération WANT_DELEGATION peut être effectuée sur tout type d'objet de fichier autre qu'un répertoire.

Quand une délégation est obtenue en utilisant WANT_DELEGATION, tous les fichiers ouverts pour la même bride de fichier détenue par ce client sont à traiter comme subordonnés à la délégation, tout comme si ils avaient été créés en utilisant un OPEN de type CLAIM_DELEGATE_CUR. Ils sont autrement inchangés quant au seqid, aux modes d'accès et de refus, et aux relations avec les verrous de gamme d'octets. De même, parce que les verrous de gamme d'octets existants sont subordonnés à une ouverture, ces verrous de gamme d'octets deviennent aussi indirectement subordonnés à cette nouvelle délégation.

L'opération WANT_DELEGATION fournit la livraison de délégations via des rappels, quand les délégations ne sont pas immédiatement disponibles. Quand une délégation demandée est disponible, elle est livrée au client via une opération CB_PUSH_DELEG. Quand cela arrive, les fichiers ouverts pour la même bride de fichier deviennent subordonnés à la nouvelle délégation au moment où la délégation est livrée, tout comme si elle avait été créée en utilisant un OPEN de type CLAIM_DELEGATE_CUR. De même, cela arrive pour les verrous de gamme d'octets existants subordonnés à une ouverture.

10.5 Mise en antémémoire de données et révocation

Quand des verrous et délégations sont révoqués, les hypothèses dont dépendent le succès de la mise en antémémoire ne sont plus garanties. Pour tout verrou ou partage de réservation qui a été révoqué, le propriétaire d'état correspondant doit être notifié. Cette notification inclut les applications avec un fichier ouvert qui a une délégation correspondante qui a été révoquée. Les données en antémémoire associées à la révocation doivent être supprimées chez le client. Dans le cas de données modifiées existantes dans l'antémémoire du client, ces données doivent être supprimées chez le client sans être écrites chez le serveur. Comme mentionné, les hypothèses faites par le client ne sont plus valides au moment où un verrou ou délégation a été révoqué. Par exemple, un autre client peut avoir reçu un verrou de gamme d'octets en conflit après la révocation du verrou de gamme d'octets chez le premier client. Donc, les données dans la gamme du verrou peuvent avoir été modifiées par l'autre client. Évidemment, le premier client est incapable de garantir à l'application ce qui est arrivé au fichier dans le cas de révocation.

La notification à un propriétaire d'état va dans de nombreux cas consister simplement à retourner une erreur sur le prochain READ/WRITE et tous les suivants sur le fichier ouvert ou sa fermeture. Lorsque les méthodes disponibles à un client rendent de telles notifications impossibles parce que des erreurs sur certaines opérations ne peuvent pas être retournées, une action plus énergique comme des signaux ou une terminaison de processus peut être appropriée. La justification est qu'un invariant dont une application dépend peut être violé. Selon la façon dont les erreurs sont normalement traitées pour l'environnement de fonctionnement du client, d'autres niveaux de notification incluant l'enregistrement, des messages à la console, et des invocations de GUI peuvent être appropriées.

10.5.1 Récupération de révocation pour délégation de Write Open

La récupération de révocation pour une délégation OPEN_DELEGATE_WRITE pose le problème particulier des données modifiées dans l'antémémoire du client alors que le fichier n'est pas ouvert. Dans cette situation, tout client qui ne purge pas les données modifiées chez le serveur à chaque clôture doit s'assurer que l'utilisateur reçoit une notification appropriée de la défaillance suite à la révocation. Comme de telles situations peuvent exiger une action humaine pour corriger les

problèmes, des schémas de notification dans lesquels l'utilisateur ou administrateur approprié est notifié peuvent être nécessaires. L'enregistrement et les messages à la console sont des exemples typiques.

Si il y a des données modifiées chez le client, elles ne doivent normalement pas être purgées chez le serveur. Un client peut tenter de fournir une copie des données du fichier comme modifié durant la délégation sous un nom différent dans l'espace de noms du système de fichiers pour faciliter la récupération. Noter que quand le client peut déterminer que le fichier n'a pas été modifié par un autre client, ou quand le client a une copie complète du fichier en question en antémémoire, une telle vue sauvegardée de la copie du fichier du client peut être d'une valeur particulière pour la récupération. Dans un autre cas, la récupération utilisant une copie du fichier fondée sur des données partiellement en antémémoire chez le client et partiellement sur la copie du serveur telle que modifiée par les autres clients va être tout sauf directe, de sorte que les clients peuvent éviter de sauvegarder les contenus de fichiers dans de telles situations ou de marquer spécialement les résultats pour avertir les utilisateurs de possibles problèmes.

Sauvegarder de telles données modifiées dans des situations de révocation de délégation peut être limité aux fichiers d'une certaine taille ou pourrait être utilisé seulement quand de l'espace disque suffisant est disponible dans le système de fichiers cible. Une telle sauvegarde peut aussi être restreinte aux situations où le client a des ressources suffisantes de mémoire tampon pour garder la copie en antémémoire disponible jusqu'à ce qu'elle soit mémorisée de façon appropriée dans le système de fichiers cible.

10.6 Mise en antémémoire d'attribut

Cette section traite de la mise en antémémoire des attributs d'un fichier chez un client quand ce client ne détient pas une délégation sur le fichier.

Les attributs discutés dans cette section n'incluent pas les attributs nommés. Les attributs nommés individuels sont analogues aux fichiers, et la mise en antémémoire des données pour eux doit être traitée tout comme la mise en antémémoire des données est faite pour les fichiers ordinaires. De même, les résultats de LOOKUP provenant d'un répertoire OPENATTR (ainsi que le contenu du répertoire) sont à mettre en antémémoire sur les mêmes bases que tout autre nom de chemin.

Les clients peuvent mettre en antémémoire des attributs de fichier obtenus du serveur et les utiliser pour éviter des demandes GETATTR suivantes. Une telle mise en antémémoire est écrite directement en ce que cette modification aux attributs de fichier est toujours faite au moyen de demandes au serveur et ne devrait pas être faite localement et ne devrait pas être mise en antémémoire. Une exception à cela est la modification des attributs qui sont intimement connectés avec la mise en antémémoire de données. Donc, étendre un fichier en écrivant des données dans l'antémémoire de données locale est reflété immédiatement dans la taille vue par le client sans que ce changement soit immédiatement reflété sur le serveur. Normalement, de tels changements ne sont pas propagés directement au serveur, mais quand les données modifiées sont purgées chez le serveur, des changements analogues d'attributs sont faits sur le serveur. Quand une délégation OPEN est en vigueur, les attributs modifiés peuvent être retournés au serveur en réaction à une invocation CB_RECALL.

Le résultat de la mise en antémémoire locale des attributs est que les antémémoires d'attribut maintenus sur les clients individuels ne vont pas être cohérents. Les changements faits dans un ordre sur le serveur peuvent être vus dans un ordre différent sur un client et dans un troisième ordre sur un autre client.

Les interfaces de programmation d'application normales de système de fichiers ne fournissent pas de moyens de modifier ou interroger atomiquement les attributs pour plusieurs fichiers en même temps. Les règles suivantes fournissent un environnement où les incohérences potentielles mentionnées ci-dessus peuvent être raisonnablement gérées. Ces règles sont déduites de la pratique des précédents protocoles NFS.

* Tous les attributs pour un fichier donné (sauf les attributs par fsid) sont mis en antémémoire comme une unité chez le client de sorte qu'aucune non mise en série ne peut apparaître dans le contexte d'un seul fichier.

* Une limite supérieure est maintenue sur la durée pendant laquelle une entrée d'antémémoire de client peut être conservée sans être rafraîchie par le serveur.

* Quand des opérations sont effectuées qui changent des attributs chez le serveur, l'ensemble d'attributs mis à jour est demandé au titre du RPC contenant. Cela inclut des opérations de répertoire qui mettent à jour indirectement les attributs. Ceci est réalisé en faisant suivre l'opération modificatrice d'une opération GETATTR et ensuite d'utiliser le résultat du GETATTR pour mettre à jour les attributs en antémémoire du client.

Noter que si l'ensemble complet des attributs à mettre en antémémoire est demandé par READDIR, le résultat peut être mis en antémémoire par le client sur les mêmes bases que les attributs obtenus via GETATTR.

Un client peut valider sa version des attributs en antémémoire pour un fichier en allant chercher les deux attributs change et time_access et en supposant que si l'attribut de changement a la même valeur que quand les attributs étaient en antémémoire, alors aucun attribut autre que time_access n'a changé. La raison pour laquelle time_access est aussi cherché est parce que de nombreux serveurs opèrent dans des environnements où l'opération que les mises à jour changent ne met pas à jour time_access. Par exemple, la sémantique de fichier POSIX ne met pas à jour l'heure d'accès quand un fichier est modifié par l'invocation du système d'écriture [IEEE-1003.1]. Donc, le client qui veut une valeur courante de time_access devrait aller la chercher avec le changement durant le traitement de validation de l'attribut en antémémoire et mettre à jour son heure d'accès en antémémoire.

Le client peut tenir une antémémoire des attributs modifiés pour les attributs intimement liés avec les données des fichiers réguliers modifiés (size, time_modify, et change). À part ces trois attributs, le client NE DOIT PAS tenir d'antémémoire des attributs modifiés. À la place, les changements d'attributs sont immédiatement envoyés au serveur.

Dans certains environnements de fonctionnement, l'équivalent de time_access est supposé être implicitement mis à jour par chaque lecture du contenu de l'objet de fichier. Si un client NFS met en antémémoire le contenu d'un objet de fichier, si c'est un fichier régulier, un répertoire, ou une liaison symbolique, le client NE DEVRAIT PAS mettre à jour l'attribut time_access (via SETATTR ou un petit READ ou demande READDIR) sur le serveur avec chaque lecture qui est satisfaite à partir de l'antémémoire. La raison en est que cela peut détruire les avantages de performances de la mise en antémémoire du contenu, en particulier parce qu'un SETATTR explicite de time_access peut altérer l'attribut de changement sur le serveur. Si l'attribut de changement change, les clients qui mettent en antémémoire le contenu vont penser que le contenu a changé, et vont relire des données non modifiées provenant du serveur. Le client n'est pas non plus encouragé à tenir une version modifiée de time_access dans son antémémoire, car le client va finalement devoir écrire l'heure d'accès sur le serveur avec de mauvais effets de performances ou ne jamais mettre à jour l'heure d'accès sur le serveur, résultant ainsi en une situation où une application qui met en antémémoire l'heure d'accès entre une clôture et une ouverture du même fichier observe une oscillation de l'heure d'accès entre le passé et le présent. L'attribut time_access signifie toujours l'heure du dernier accès à un fichier par une lecture qui a été satisfaite par le serveur. De cette façon, les clients vont tendre à voir seulement les changements de time_access qui vont plus loin dans le temps.

10.7 Mise en antémémoire de données et métadonnées et fichiers transposés de la mémoire

Certains environnements de fonctionnement incluent la capacité qu'une application transpose le contenu d'un fichier en l'espace d'adresses de l'application. Chaque fois que l'application accède à une localisation de mémoire qui correspond à un bloc qui n'a pas été chargé dans l'espace d'adresses, une faute de page se produit et le fichier est lu (ou si le bloc n'existe pas dans le fichier, le bloc est alloué et ensuite instancié dans l'espace d'adresses de l'application).

Tant que chaque accès transposé de la mémoire au fichier exige une faute de page, les attributs pertinents du fichier qui sont utilisés pour détecter l'accès et la modification (time_access, time_metadata, time_modify, et change) vont être mis à jour. Cependant, dans de nombreux environnements de fonctionnement, quand des fautes de page ne sont pas exigées, ces attributs ne vont pas être mis à jour sur les lectures ou mises à jour du fichier via un accès à la mémoire (sans considération de si le fichier est local ou est accédé à distance). Un client ou serveur PEUT échouer à mettre à jour les attributs d'un fichier qui est accédé via une entrée/sortie transposée de la mémoire. Cela a plusieurs implications :

* Si il y a une application sur le serveur qui a un fichier transposé de la mémoire auquel accède aussi un client, le client peut n'être pas capable d'obtenir une valeur cohérente de l'attribut de changement pour déterminer si son antémémoire est périmée ou non. Un serveur qui sait que le fichier est transposé de la mémoire pourrait toujours retourner des valeurs pessimistes de mise à jour pour le changement de façon à forcer l'application à obtenir toujours les données et métadonnées les plus à jour pour le fichier. Cependant, du fait des implications négatives de ceci sur les performances, un tel comportement est FACULTATIF.

* Si le fichier n'est pas modifié sur le serveur, et est juste lu par une application via l'interface de transposition de la mémoire, le client ne va pas voir un attribut time_access mis à jour. Cependant, dans de nombreux environnements de fonctionnement, il ne verra pas non plus de traitement fonctionnant sur le serveur. Donc, les clients NFS n'ont aucun désavantage par rapport au traitement local.

* Si il y a un autre client qui transpose le fichier depuis la mémoire, et si ce client détient une délégation OPEN_DELEGATE_WRITE, le même ensemble de problèmes que discutés dans les deux alinéas précédents s'applique. Donc, quand un serveur fait un CB_GETATTR sur un fichier que le client a modifié dans son antémémoire,

la réponse provenant de CB_GETATTR ne va pas nécessairement être précise. Comme exposé précédemment, l'obligation du client est de rapporter que le fichier a été modifié depuis que la délégation a été accordée, et non si il a été modifié à nouveau entre des invocations successives de CB_GETATTR, et le serveur DOIT supposer que tout fichier que le client a modifié dans son antémémoire a été modifié à nouveau entre les invocations successives de CB_GETATTR. Selon la nature du système de gestion de la mémoire du client, cette faible obligation peut n'être pas possible. Un client PEUT retourner des informations périmées dans CB_GETATTR chaque fois que le fichier est transposé de la mémoire.

* Le mélange de transposition de mémoire et de verrouillage de gamme d'octets sur le même fichier est problématique. Considérons le scénario suivant, où la taille de page chez chaque client est de 8192 octets.
  - la mémoire du client A transpose la première page (8192 octets) du fichier X ;
  - la mémoire du client B transpose la première page (8192 octets) du fichier X ;
  - le WRITE_LT du client A verrouille les 4096 premiers octets ;
  - le WRITE_LT du client B verrouille les seconds 4096 octets ;
  - le client A, via une instruction STORE, modifie une partie de sa gamme d'octets verrouillée ;
  - simultanément au client A, le client B exécute un STORE sur une partie de sa gamme d'octets verrouillée.

Ici, le défi est pour chaque client de se resynchroniser pour obtenir une vue correcte de la première page. Dans de nombreux environnements de fonctionnement, les systèmes de gestion de mémoire virtuelle sur chaque client savent seulement qu'une page est modifiée, pas qu'un sous ensemble de la page correspondant aux gammes d'octets verrouillées respectives a été modifié. Donc il n'est pas possible à chaque client de faire la bonne chose, qui est d'écrire au serveur seulement la portion de la page qui est verrouillée. Par exemple, si le client A écrit simplement la page, et qu'ensuite le client B écrit la page, les données du client A sont perdues.

De plus, si le verrouillage obligatoire est activé sur le fichier, on a alors un problème différent. Quand les clients A et B exécutent les instructions STORE, les fautes résultantes de page exigent un verrou de gamme d'octets sur la page entière. Chaque client essaye alors d'étendre sa gamme verrouillée de la page entière, ce qui aboutit à une impasse. La communication de l'erreur NFS4ERR_DEADLOCK à une instruction STORE est difficile au mieux.

Si un client verrouille le fichier entier transposé de la mémoire, il n'y a aucun problème avec le verrouillage indicatif ou obligatoire de gamme d'octets, au moins jusqu'à ce que le client déverrouille une gamme d'octets au milieu du fichier.

Étant donnés les problèmes ci-dessus, ce qui suit est permis :

* Les clients et serveurs PEUVENT refuser de transposer de la mémoire un fichier pour lequel ils savent qu'il y a des verrous de gamme d'octets.
* Les clients et serveurs PEUVENT refuser un verrou de gamme d'octets sur un fichier dont ils savent qu'il est transposé de la mémoire.
* Un client PEUT refuser de transposer de la mémoire un fichier dont il sait qu'il exige le verrouillage obligatoire pour l'entrée/sortie. Si le verrouillage obligatoire est activé après que le fichier est ouvert et transposé, le client PEUT refuser que l'application accède à son fichier transposé.

10.8 Mise en antémémoire de nom et de répertoire sans délégation de répertoire

La facilité de délégation de répertoire NFSv4.1 (décrite au paragraphe 10.9) est de mise en œuvre FACULTATIVE pour les serveurs. Même lorsque elle est mise en œuvre, elle peut n'être pas toujours fonctionnelle à cause de contraintes de disponibilité de ressource ou autres. Donc, il est important de comprendre comment la mise en antémémoire de noms et répertoire est faite en l'absence de délégations de répertoire. Ces sujets sont discutés dans les deux sous paragraphes suivants.

10.8.1 Mise en antémémoire de nom

Le résultat des opérations LOOKUP et READDIR peut être mis en antémémoire pour éviter les frais d'opérations LOOKUP suivantes. Tout comme dans le cas de la mise en antémémoire d'attribut, des incohérences peuvent survenir entre les diverses antémémoires du client. Pour atténuer les effets de ces incohérences et étant donné le contexte des API de système de fichiers normales, une limite supérieure est établie pour la durée pendant laquelle une entrée d'antémémoire de nom de client peut être conservée sans vérifier que l'entrée n'a pas été rendue invalide par une opération de changement de répertoire effectuée par un autre client.

Quand un client ne fait pas de changement à un répertoire pour lequel il existe des entrées d'antémémoire de noms, le client

doit périodiquement aller chercher les attributs pour ce répertoire pour s'assurer qu'il n'est pas modifié. Après avoir déterminé qu'aucune modification n'est survenue, l'heure d'expiration pour les entrées d'antémémoire de noms associées peut être mise à jour à l'heure actuelle plus la limite de péremption de l'antémémoire de noms.

Quand un client fait des changements à un répertoire, il doit déterminer si il y a eu des changements faits au répertoire par d'autres clients. Il fait cela en utilisant l'attribut de changement tel que rapporté avant et après l'opération de répertoire dans la valeur change_info4 associée retournée pour l'opération. Le serveur est capable de communiquer avec le client si les données de change_info4 sont fournies de façon atomique par rapport à l'opération de répertoire. Si les valeurs de changement sont fournies de façon atomique, le client a une base pour déterminer, avec les précautions appropriées, si d'autres clients ont modifié le répertoire en question.

La façon la plus simple de permettre au client de faire cette détermination est que le client mette en série tous les changements faits à un répertoire spécifique. Quand ceci est fait, et que le serveur fournit les valeurs atomiques avant et après de l'attribut de changement, le client peut simplement comparer la valeur après de l'attribut de changement provenant d'une opération sur un répertoire avec la valeur avant sur l'opération suivante qui modifie ce répertoire. Quand elles sont égales, le client est assuré qu'aucun autre client n'a modifié le répertoire en question.

Quand une telle mise en série n'est pas utilisée, et qu'il peut y avoir plusieurs opérations simultanées en instance qui modifient un seul répertoire envoyé d'un seul client, faire cette sorte de détermination peut être plus compliqué. Si deux opérations s'achèvent dans un ordre différent de celui qui a été réellement effectué, cela pourrait donner une apparence de cohérence avec la modification faite par un autre client. Lorsque cela paraît arriver, le client doit attendre l'achèvement de toutes ces modifications qui ont commencé précédemment, pour voir si les numéros de changement en instance avant et après peuvent être triés en une chaîne telle que la valeur avant d'un numéro de changement corresponde à la valeur après d'une précédente, dans une chaîne cohérente, ce client étant le seul qui modifie le répertoire.

Dans ces deux cas, le client est capable de déterminer si le répertoire est modifié par un autre client. Si la comparaison indique que le répertoire a été mis à jour par un autre client, l'antémémoire de nom associée au répertoire modifié est purgée chez le client. Si la comparaison n'indique aucune modification, l'antémémoire de nom peut être mise à jour chez le client pour refléter l'opération de répertoire et la temporisation associée peut être étendue. La valeur de changement post opération doit être sauvegardée comme base des futures comparaisons de change_info4.

Comme montré par le scénario ci-dessus, la mise en antémémoire de nom exige que le client revalide les données d'antémémoire de nom en inspectant l'attribut de changement d'un répertoire au moment où l'élément d'antémémoire de nom a été mis en antémémoire. Cela exige que le serveur mette à jour l'attribut de changement pour les répertoires quand le contenu du répertoire correspondant est modifié. Pour qu'un client utilise les informations de change_info4 de façon appropriée et correcte, le serveur doit rapporter atomiquement les valeurs d'attribut de changement pré et post opération. Quand le serveur est incapable de rapporter les valeurs atomiques avant et après par rapport à l'opération de répertoire, le serveur doit indiquer ce fait dans la valeur de retour de change_info4. Quand l'information n'est pas rapportée de façon atomique, le client ne devrait pas supposer que d'autres clients n'ont pas changé le répertoire.

10.8.2 Mise en antémémoire de répertoire

Les résultats des opérations READDIR peuvent être utilisés pour éviter des opérations READDIR suivantes. Tout comme dans le cas de la mise en antémémoire d'attribut et de nom, des incohérences peuvent survenir parmi les diverses antémémoires de client. Pour atténuer les effets de ces incohérences, et étant donné le contexte des API de système de fichiers normales, les règles suivantes devraient être respectées :

* Les informations de READDIR en antémémoire pour un répertoire qui ne sont pas obtenues dans une seule opération READDIR doivent toujours être un instantané cohérent du contenu du répertoire. Ceci est déterminé en utilisant un GETATTR avant le premier READDIR et après le dernier READDIR qui contribuent à l'antémémoire.

* Une limite supérieure de durée est maintenue pour indiquer pendant combien de temps une entrée d'antémémoire de répertoire est considérée comme valide avant que le client doive revalider les informations en antémémoire.

La technique de revalidation est parallèle à celle discutée dans le cas de la mise en antémémoire de noms. Quand le client ne change pas le répertoire en question, vérifier l'attribut de changement du répertoire avec GETATTR est adéquat. La durée de vie de l'entrée d'antémémoire peut être étendue à ces points de vérification. Quand un client modifie le répertoire, il doit utiliser les données de change_info4 pour déterminer si d'autres clients modifient le répertoire. Si il est déterminé qu'aucun autre client ne produit de modifications, le client peut mettre à jour son antémémoire de répertoire pour refléter ses propres changements.

Comme montré précédemment, la mise en antémémoire de répertoire exige que le client revalide les données d'antémémoire de répertoire en inspectant l'attribut de changement d'un répertoire au moment où le répertoire a été mis en antémémoire. Cela exige que le serveur mette à jour l'attribut de changement pour les répertoires quand le contenu du répertoire correspondant est modifié. Pour qu'un client utilise les informations de change_info4 de façon appropriée et correcte, le serveur doit rapporter de façon atomique les valeurs de l'attribut de changement pré et post opération. Quand le serveur est incapable de rapporter de façon atomique des valeurs avant et après par rapport à l'opération de répertoire, le serveur doit indiquer ce fait dans la valeur de change_info4 retournée. Quand l'information n'est pas rapportée de façon atomique, le client ne devrait pas supposer que d'autres clients n'ont pas changé le répertoire.

10.9 Délégations de répertoire

10.9.1 Introduction aux délégations de répertoire

La mise en antémémoire de répertoire pour le protocole NFSv4.1, comme décrit précédemment, est similaire à la mise en antémémoire de fichier dans les versions précédentes. Les clients mettent normalement en antémémoire les informations de répertoire pour une durée déterminée par le client. À la fin d'une période prédéfinie, le client va interroger le serveur pour voir si le répertoire a été mis à jour. En mettant en antémémoire des attributs, les clients réduisent le nombre d'appels GETATTR faits au serveur pour valider les attributs. De plus, les fichiers et répertoires fréquemment accédés, comme le répertoire de travail courant, ont leurs attributs en antémémoire chez le client afin que certaines opérations NFS puissent être effectuées sans avoir à faire un appel RPC. En mettant en antémémoire le nom et les informations inode sur la plupart des entrées examinées récemment dans une antémémoire de recherche de nom de répertoire (DNLC, *Directory Name Lookup Cache*) les clients n'ont pas besoin d'envoyer d'appel LOOKUP au serveur chaque fois qu'ils accèdent à ces fichiers.

Cette approche de mise en antémémoire fonctionne raisonnablement bien pour réduire le trafic réseau dans de nombreux environnements. Cependant, cela ne vise pas les environnements où il y a de nombreuses interrogations pour des fichiers qui n'existent pas. Dans ces cas de "manque", le client envoie des demandes au serveur afin de fournir une sémantique d'application raisonnable et détecter rapidement la création de nouvelles entrées de répertoire. Des exemples d'activité élevée de manques sont la compilation dans des environnements de développement de logiciel. Le comportement actuel de NFS limite son adaptabilité potentielle et l'effectivité du partage de large zone dans ces types d'environnements. D'autres architectures réparties de système de fichiers à états pleins tels que AFS et DFS ont prouvé que l'ajout d'état autour du contenu de répertoire peut réduire considérablement le trafic réseau dans les environnements de manques élevés.

La délégation de contenu de répertoire est une caractéristique FACULTATIVE de NFSv4.1. Les délégations de répertoire fournissent des avantages de réduction de trafic similaires à celles des délégations de fichier. En permettant aux clients de mettre en antémémoire le contenu d'un répertoire (en lecture seule) tout en étant notifié des changements, le client peut éviter de faire de fréquentes demandes d'interrogation sur les contenus de répertoires qui changent lentement, réduisant le trafic du réseau et améliorant les performances du client. Cela peut aussi simplifier la tâche de déterminer si d'autres clients font des changements au répertoire quand le client fait lui-même de nombreux changements au répertoire et que les changements ne sont pas à la suite.

Les délégations de répertoire permettent de réaliser une cohérence améliorée de l'espace de noms d'antémémoire par des délégations et des rappels synchrones, en l'absence de notifications. De plus, si la cohérence fondée sur le temps est suffisante, des notifications asynchrones peuvent fournir des avantages de performances pour le client, et éventuellement le serveur, dans des conditions de fonctionnement courantes comme sur des répertoires changeant lentement et/ou très grands.

10.9.2 Conception de la délégation de répertoire

NFSv4.1 introduit l'opération GET_DIR_DELEGATION (paragraphe 18.39) pour permettre au client de demander une délégation de répertoire. La délégation couvre les attributs de répertoire et toutes les entrées dans le répertoire. Si l'un change, la délégation va être rappelée en synchrone. L'opération causant le rappel va devoir attendre que le rappel soit achevé. Un changement d'attribut d'entrée de répertoire ne va pas causer le rappel de la délégation.

En plus de demander des délégations, un client peut aussi demander des notifications pour certains événements. Ces événements incluent des changements des attributs et/ou du contenu du répertoire. Si un client demande la notification d'un certain événement, le serveur va le notifier au client quand cet événement se produit. Il ne va pas en résulter le rappel de la délégation pour ce client. Les notifications sont asynchrones et fournissent un moyen d'éviter des rappels dans des situations où un répertoire change assez pour que le modèle de pur rappel ne puisse pas être efficace quand on essaye de

permettre au client d'obtenir un avantage substantiel. En l'absence de notifications, une fois que la délégation est rappelée le client doit rafraîchir son antémémoire de répertoire ; cela pourrait n'être pas très efficace pour les très grands répertoires.

La délégation est en lecture seule et le client ne peut pas faire de changements au répertoire autrement qu'en effectuant des opérations NFSv4.1 qui modifient le répertoire ou les attributs de fichier associés afin que le serveur ait connaissance de ces changements. Afin de garder l'espace de noms du client synchronisé avec celui du serveur, le serveur va notifier au client qui détient la délégation (en supposant qu'il a demandé les notifications) les changements faits par suite de ces opérations de modification de répertoire du client. C'est pour éviter que le client ait besoin d'envoyer des opérations GETATTR ou READDIR suivantes au serveur. Si un seul client détient la délégation et si ce client fait des changements au répertoire (c'est-à-dire, les changements sont faits via des opérations envoyées sur une session associée à l'identifiant de client détenant la délégation) la délégation ne va pas être rappelée. Plusieurs clients peuvent détenir une délégation sur le même répertoire, mais si un tel client modifie le répertoire, le serveur DOIT rappeler la délégation des autres clients, sauf si ces clients ont pris des dispositions pour être notifiés de cette sorte de modification.

Les délégations peuvent être rappelées par le serveur à tout moment. Normalement, le serveur va rappeler la délégation quand le répertoire change d'une façon qui n'est pas couverte par la notification, ou quand le répertoire change et que les notifications n'ont pas été demandées. Si un autre client supprime le répertoire pour lequel une délégation a été accordée, le serveur va rappeler la délégation.

10.9.3 Attributs pour la prise en charge des notifications de répertoire

Voir au paragraphe 5.11 la description des attributs associés à des notifications de répertoire.

10.9.4 Rappel de délégation de répertoire

Le serveur va rappeler la délégation de répertoire en envoyant un rappel au client. Il va utiliser la même procédure de rappel qu'utilisée pour rappeler les délégations de fichier. Le serveur va rappeler la délégation quand le répertoire change d'une façon qui n'est pas couverte par la notification. Cependant, le serveur n'a pas besoin de rappeler la délégation si des attributs d'une entrée dans le répertoire changent.

Si le serveur remarque que la distribution d'une délégation pour un répertoire cause l'envoi de trop de notifications, il peut décider de ne pas distribuer les délégations pour ce répertoire et/ou rappeler celles déjà accordées. Si un client essaye de supprimer le répertoire pour lequel une délégation a été accordée, le serveur va rappeler toutes les délégations associées.

Les paragraphes de mise en œuvre pour un certain nombre d'opérations décrivent les situations dans lesquelles la notification ou le rappel de délégation va être requis dans des circonstances communes. À cet égard, un ensemble d'avertissements similaires à ceux mentionnés au paragraphe 10.2 s'applique.

* Pour CREATE, voir le paragraphe 18.4.4.
* Pour LINK, voir le paragraphe 18.9.4.
* Pour OPEN, voir le paragraphe 18.16.4.
* Pour REMOVE, voir le paragraphe 18.25.4.
* Pour RENAME, voir le paragraphe 18.26.4.
* Pour SETATTR, voir le paragraphe 18.30.4.

10.9.5 Récupération de délégation de répertoire

La récupération du client ou le redémarrage de serveur pour l'état sur les fichiers réguliers a deux principaux objectifs : éviter d'avoir besoin de casser les garanties d'application à l'égard des fichiers verrouillés et de la livraison de mises à jour d'antémémoire chez le client. Aucun de ces objectifs ne s'applique aux répertoires protégés par les délégations et notifications OPEN_DELEGATE_READ. Donc, aucune disposition n'est prise pour réclamer les délégations de répertoire en cas de redémarrage de client ou de serveur. Le client peut simplement établir une délégation de répertoire de la même façon que comme il l'a fait initialement.

11. Espace de noms multi-serveurs

NFSv4.1 prend en charge des attributs qui permettent qu'un espace de noms s'étende au delà des frontières d'un seul serveur. Il est souhaitable que clients et serveurs prennent en charge la construction de tels espaces de noms multi-serveurs.

L'utilisation de tels espaces de noms multi-serveurs est FACULTATIVE ; cependant, et pour de nombreux objets, des espaces de noms d'un seul serveur sont parfaitement acceptables. L'utilisation d'espaces de noms multi-serveurs peut fournir de nombreux avantages en séparant la position logique d'un système de fichiers dans un espace de noms des considérations (possiblement changeantes) logistiques et administratives qui causent la localisation d'un système de fichiers particulier sur un serveur particulier via un seul chemin d'accès réseau qui doit être connu à l'avance ou déterminé en utilisant le DNS.

11.1 Terminologie

Dans toute cette section (c'est-à-dire, toute la Section 11) la phrase "identifiant de client" se réfère toujours à l'identifiant abrégé de 64 bits alloué par le serveur (un clientid4) et jamais à la structure que le client utilise pour s'identifier au serveur (appelé un nfs_client_id4 ou client_owner dans NFSv4.0 et NFSv4.1, respectivement). L'identifiant opaque dans ces structures est appelé une "chaîne d'identifiant de client".

11.1.1 Terminologie relative au multiplexage

Il est particulièrement important de préciser la distinction entre détection de multiplexage et découverte de multiplexage. Les définitions qu'on présente sont applicables à toutes les versions mineures de NFSv4, mais on va se concentrer sur les termes qui s'appliquent à NFS version 4.1.

* Détection de multiplexage se réfère aux façons de décider si deux adresses réseau spécifiques sont connectées au même serveur NFSv4. Les moyens disponibles pour cette détermination dépendent du protocole, et, dans certains cas, de la mise en œuvre de client. Dans le cas de NFS version 4.1 et des versions mineures ultérieures, les moyens de détection du multiplexage sont comme décrit dans le présent document et sont disponibles à chaque client. Deux adresses réseau connectées au même serveur peuvent toujours être utilisées ensembles pour accéder à un serveur particulier mais ne peuvent pas nécessairement être utilisées ensemble pour accéder à une seule session. Voir ci-dessous les définitions des termes "multiplexable pour le serveur" et "multiplexable pour la session".

* La découverte de multiplexage est un processus par lequel un client utilisant une adresse réseau peut obtenir d'autres adresses qui sont connectées au même serveur. Normalement, elle s'appuie sur une facilité de détection de multiplexage en fournissant une ou plusieurs méthodes par lesquelles les adresses candidates sont rendues disponibles au client, qui peut alors utiliser la détection de multiplexage pour les filtrer de façon appropriée. En dépit de la prise en charge de la détection de multiplexage, il n'avait pas de description de de la découverte de multiplexage dans la [RFC5661], rendant nécessaire de fournir ces moyens dans le présent document.

La combinaison d'une adresse réseau de serveur et d'un type particulier de connexion à utiliser par une connexion est appelée un "point d'extrémité de serveur". Bien qu'utiliser différents types de connexion puisse résulter en l'utilisation d'accès différents, l'utilisation d'accès différents par plusieurs connexions à la même adresse réseau dans ces cas n'est pas l'essence de la distinction entre les deux points d'extrémité utilisés. Ceci est différent du cas des points d'extrémité spécifiques d'accès, dans lesquels la spécification explicite de numéros d'accès au sein des adresses réseau est utilisée pour permettre à un seul nœud serveur de prendre en charge plusieurs serveurs NFS.

Deux adresses réseau connectées au même serveur sont dites être multiplexables pour le serveur. Deux telles adresses prennent en charge l'utilisation du multiplexage d'identifiant de client, comme décrit au paragraphe 2.10.5.

Deux adresses réseau connectées au même serveur de telle façon que ces adresses puissent être utilisées pour prendre en charge une seule session commune sont appelées multiplexables pour la session. Noter que deux adresses peuvent être multiplexables pour le serveur sans être multiplexables pour la session, et que, quand deux connexions de types de connexion différents sont faites à la même adresse réseau et sont fondées sur une seule entrée de localisation de système de fichiers, elles sont toujours multiplexables pour la session, indépendamment du type de la connexion, comme spécifié au paragraphe 2.10.5, car leur déduction de la même entrée de localisation de système de fichiers, avec l'identité de leurs adresses réseau, assurent que les deux connexions sont pour le même serveur et vont retourner des informations de propriétaire de serveur, permettant d'utiliser le multiplexage de session.

11.1.2 Terminologie relative à la localisation du système de fichiers

Concernant la terminologie relative à la construction d'espaces de noms multi-serveurs sur un ensemble d'espaces de noms locaux par serveur :

* Chaque serveur a un ensemble de systèmes de fichiers exportés qui peuvent être accédés par les clients NFSv4. Normalement, c'est fait en allouant à chaque système de fichiers un nom dans le pseudo-fs associé au serveur, bien que le pseudo-fs puisse être omis si il y a seulement un seul système de fichiers exporté. Chacun de ces systèmes de fichiers fait partie de l'espace de noms local du serveur, et peut être considéré comme une instance de système de fichiers au sein d'un plus large espace de noms multi-serveurs.

* L'ensemble de tous les systèmes de fichiers exportés pour un serveur donné constitue l'espace de nom local de ce serveur.

* Dans certains cas, un serveur va avoir un espace de noms plus extensif que son espace de noms local en utilisant des caractéristiques associées à des attributs qui fournissent des informations de localisation de système de fichiers. Ces caractéristiques, qui permettent la construction d'un espace de noms multi-serveurs, sont toutes décrites dans les paragraphes individuels qui suivent et incluent des références (paragraphe 11.5.6), des migrations (paragraphe 11.5.5), et des duplications (paragraphe 11.5.4).

* Un système de fichiers présent dans le pseudo-fs d'un serveur peut avoir plusieurs instances de système de fichiers sur différents serveurs qui lui sont associés. Toutes ces instances sont considérées être des répliques les unes des autres. Si de telles répliques peuvent être utilisées simultanément est discuté au paragraphe 11.11.1, tandis que le niveau de coordination entre elles (important quand on les commute) est discuté dans les paragraphes 11.11.2 à 11.11.8.

* Quand un système de fichiers est présent dans le pseudo-fs d'un serveur, mais qu'il n'y a pas de système de fichiers local correspondant, il est dit être "absent". Dans ce cas, toutes les instances associées vont être accédées sur d'autres serveurs.

Concernant la terminologie relative aux attributs utilisés dans la découverte de multiplexage et les autres caractéristiques d'espace de noms multi-serveurs :

* Les attributs de localisation de système de fichiers incluent les attributs fs_locations et fs_locations_info.

* Les entrées de localisation de système de fichiers fournissent les localisations individuelles de système de fichiers au sein des attributs de localisation de système de fichiers. Chacune de ces entrées spécifie un serveur, sous la forme d'un nom d'hôte ou d'une adresse, et d'un nom de fs, qui désigne la localisation du système de fichiers au sein de l'espace de noms local du serveur. Une entrée de localisation de système de fichiers désigne un ensemble de points d'extrémité de serveur avec lesquels le client peut établir des connexions. Il peut y avoir plusieurs points d'extrémité parce que un nom d'hôte peut se transposer en plusieurs adresses réseau et parce que plusieurs types de connexion peuvent être utilisés pour communiquer avec une seule adresse réseau. Cependant, sauf lorsque des numéros d'accès explicites sont utilisés pour désigner un ensemble de serveurs au sein d'un seul nœud serveur, tous ces points d'extrémité DOIVENT désigner un moyen de se connecter à un seul serveur. La forme exacte de l'entrée de localisation varie avec l'attribut particulier de localisation de système de fichiers utilisé, comme décrit au paragraphe 11.2.

  Les adresses réseau utilisées dans les entrées de localisation de système de fichiers apparaissent normalement sans indication de numéro d'accès et sont utilisées pour désigner un serveur à un des accès standard pour l'accès NFS, par exemple, 2049 pour TCP ou 20049 pour RPC sur RDMA. Les numéros d'accès peuvent être utilisés dans les entrées de localisation de système de fichiers pour désigner les serveurs (normalement ceux au niveau utilisateur) auxquels on accède en utilisant d'autres numéros d'accès. Dans le cas où les adresses réseau indiquent des relations de multiplexage, l'utilisation d'un numéro d'accès explicite est inappropriée car le multiplexage est une relation entre des adresses réseau. Voir les détails au paragraphe 11.5.2.

* Les éléments de localisation de système de fichiers sont déduits des entrées de localisation, et chacune décrit un chemin d'accès réseau particulier consistant en une adresse réseau et une localisation au sein de l'espace de noms local du serveur. De tels éléments de localisation n'ont pas besoin d'apparaître dans l'attribut de localisation d'un système de fichiers, mais l'existence de chaque élément de localisation dérive d'une entrée de localisation correspondante. Quand une entrée de localisation spécifie une adresse IP, il y a seulement un élément de localisation correspondant. Les entrées de localisation de système de fichier qui contiennent un nom d'hôte sont résolues en utilisant le DNS, et peuvent résulter en un ou plusieurs éléments de localisation. Tous les éléments de localisation consistent en une adresse de localisation qui inclut l'adresse IP d'une interface à un serveur et un nom de fs, qui est la localisation du système de fichiers au sein de l'espace de noms local du serveur. Le nom de fs peut être vide si le serveur n'a pas de pseudo-fs et seulement un seul système de fichiers exporté à la bride de fichier racine.

* Deux éléments de localisation de système de fichiers sont dits être multiplexables pour le serveur si ils spécifient le même nom de fs et si les adresses de localisation sont telles que les adresses de localisation sont multiplexables pour le

serveur. Quand les chemins de réseau correspondants sont utilisés, le client va toujours être capable d'utiliser le multiplexage d'identifiant de client, mais va seulement être capable d'utiliser le multiplexage de session si les chemins sont aussi multiplexables pour la session.

* Deux éléments de localisation de système de fichiers sont dits être multiplexables pour la session si ils spécifient le même nom de fs et si les adresses de localisation sont telles qu'elles soient multiplexables pour la session. Quand les chemins de réseau correspondants sont utilisés, le client va être capable d'utiliser le multiplexage d'identifiant de client ou le multiplexage de session.

La discussion du terme "réplique" est compliquée par le fait que le terme a été utilisé dans la [RFC5661] avec une signification différente de celle utilisée dans le présent document. En bref, dans la [RFC5661] chaque réplique est identifiée par un seul chemin d'accès réseau, tandis que dans le présent document, un ensemble de chemins d'accès réseau qui ont des adresses réseau multiplexables par le serveur et le même nom de chemin de système de fichiers relatif à la racine est considéré être une seule réplique avec plusieurs chemins d'accès réseau.

Chaque ensemble d'éléments de localisation multiplexables par le serveur définit un ensemble de chemins d'accès réseau disponibles pour un système de fichiers particulier. Quand il y a plusieurs de ces systèmes de fichiers, chacun contenant les mêmes données, ces systèmes de fichiers sont considérés être des répliques les uns des autres. Logiquement, une telle réplication est symétrique, car le fs actuellement utilisé et un autre fs sont des répliques de chaque autre. Souvent, dans d'autres documents, le terme de "réplique" n'est pas appliqué au fs actuellement utilisé, en dépit du fait que la relation de réplication est par nature symétrique.

11.2 Attributs de localisation de système de fichiers

NFSv4.1 contient des attributs qui fournissent des informations sur la façon dont un système de fichiers donné peut être accédé (c'est-à-dire, à quelle adresse réseau et position d'espace de noms). Par suite, les systèmes de fichiers dans l'espace de noms d'un serveur peuvent être associés à une ou plusieurs instances de ce système de fichiers sur d'autres serveurs. Ces attributs contiennent des entrées de localisation de système de fichiers qui spécifient une adresse de serveur cible (un nom DNS représentant une ou plusieurs adresses IP ou une adresse IP spécifique) avec le nom de chemin de ce système de fichiers au sein de l'espace de noms du seul serveur associé.

L'attribut RECOMMANDÉ fs_locations_info permet la spécification de une ou plusieurs localisations d'instance de système de fichiers où les données correspondant à un système de fichiers donné peuvent être trouvées. En plus de la spécification des localisations d'instance de système de fichiers, cet attribut fournit des informations utiles pour faire ce qui suit :

* Guider les choix parmi les diverses instances de système de fichiers fournies (par exemple, priorité d'utilisation, possibilité d'écriture, devises, etc.).
* Aider efficacement le client à effectuer une transition aussi en douceur que possible parmi plusieurs instances de système de fichiers, quand et si ce devrait être nécessaire.
* Guider le choix du type approprié de connexion à utiliser quand on établit une connexion.

Dans l'attribut fs_locations_info, chaque entrée de fs_locations_server4 correspond à une entrée de localisation de système de fichiers : le champ fls_server désigne le serveur, et le champ fl_rootpath du fs_locations_item4 englobant donne le nom de chemin de localisation au sein du pseudo-fs du serveur.

L'attribut fs_locations défini dans NFSv4.0 fait aussi partie de NFSv4.1. Cet attribut permet seulement la spécification des localisations du système de fichiers où les données correspondant à un système de fichiers donné peuvent être trouvées. Les serveurs DEVRAIENT rendre cet attribut disponible chaque fois que fs_locations_info est pris en charge, mais l'utilisation par le client de fs_locations_info est préférable parce que elle fournit plus d'informations.

Dans l'attribut fs_locations, chaque fs_location4 contient une entrée de localisation de système de fichiers avec le champ Serveur qui désigne le serveur et le champ Rootpath qui donne le nom de chemin de la localisation au sein du pseudo-fs du serveur.

11.3 Présence ou absence de système de fichiers

Une localisation donnée dans un espace de noms NFSv4.1 (normalement mais pas nécessairement un espace de noms multi-serveurs) peut avoir un certain nombre de localisations d'instances de système de fichiers qui lui sont associées (via l'attribut fs_locations ou fs_locations_info). Il peut aussi y avoir un système de fichiers réel en cours à cette localisation,

accessible via des opérations normales d'espace de noms (par exemple, LOOKUP). Dans ce cas, le système de fichiers est dit être "présent" à cette position dans l'espace de noms, et les clients vont normalement l'utiliser, réservant l'utilisation des localisations supplémentaires spécifiées via les attributs en relation avec la localisation aux situations dans lesquelles la localisation principale n'est plus disponible.

Quand il y a un système de fichiers non actuel à la localisation d'espace de noms en question, le système de fichiers est dit être "absent". Un système de fichiers absent ne contient pas de fichiers ou répertoires autres que la racine. Toute référence à lui, sauf pour accéder à un petit ensemble d'attributs utiles pour déterminer des localisations de remplacement, va résulter en une erreur, NFS4ERR_MOVED. Noter que si le serveur retourne l'erreur NFS4ERR_MOVED, il DOIT prendre en charge l'attribut fs_locations et DEVRAIT prendre en charge les attributs fs_locations_info et fs_status.

Bien que le nom de l'erreur suggère que nous avons un cas d'un système de fichiers qui a été présent, et est seulement devenu absent ultérieurement, c'est seulement une possibilité. Une position dans l'espace de noms peut être absente en permanence avec l'ensemble de systèmes de fichiers désigné par les attributs de localisation étant la seule réalisation. Le nom NFS4ERR_MOVED reflète une conception antérieure, plus limitée de sa fonction, mais cette erreur va être retournée chaque fois que le système de fichiers référencé est absent, qu'il ait bougé ou non.

Sauf dans le cas des opérations de type GETATTR (discuté plus loin) quand la bride de fichier courante au début d'une opération est dans un système de fichiers absent, cette opération n'est pas effectuée et l'erreur NFS4ERR_MOVED est retournée, pour indiquer que le système de fichiers est absent sur le serveur en cours.

Parce que un GETFH ne peut pas réussir si la bride de fichier courante est dans un système de fichiers absent, les brides de fichier dans un système de fichiers absent ne peuvent pas être transférées au client. Quand un client a des brides de fichier au sein d'un système de fichiers absent, c'est le résultat de les avoir obtenues quand le système de fichiers était présent, et que le système de fichiers est ensuite devenu absent.

On devrait noter que parce que la vérification que la bride de fichier courante se trouve être dans un système de fichiers absent se produit au début de chaque opération, les opérations qui changent la bride de fichier courante de telle sorte qu'elle soit dans un système de fichiers absent ne vont pas résulter en une erreur. Cela permet des combinaisons comme PUTFH-GETATTR et LOOKUP-GETATTR pour obtenir des informations d'attributs, en particulier des informations d'attributs de localisation, comme discuté ci-dessous.

L'attribut RECOMMANDÉ de système de fichiers fs_status peut être utilisé pour interroger sur l'état présent/absent d'un système de fichiers.

11.4 Obtenir des attributs pour système de fichiers absent

Quand un système de fichiers est absent, la plupart des attributs ne sont pas disponibles, mais il est nécessaire de permettre au client d'accéder au petit ensemble d'attributs disponibles, et en particulier ceux qui donnent des informations sur les localisations courantes correctes pour ce système de fichiers : fs_locations et fs_locations_info.

11.4.1 GETATTR dans un système de fichiers absent

Comme mentionné ci-dessus, une exception est faite pour GETATTR en ce que des attributs peuvent être obtenus pour une bride de fichier dans un système de fichiers absent. Cette exception s'applique seulement si le gabarit d'attribut contient au moins un bit d'attribut qui indique que le client est intéressé par un résultat concernant un système de fichiers absent : fs_locations, fs_locations_info, ou fs_status. Si aucun de ces attributs n'est demandé, GETATTR va résulter en une erreur NFS4ERR_MOVED.

Quand un GETATTR est fait sur un système de fichiers absent, l'ensemble d'attributs pris en charge est très limité. De nombreux attributs, y compris ceux qui sont normalement EXIGÉS, ne vont pas être disponibles sur un système de fichiers absent. En plus des attributs mentionnés ci-dessus (fs_locations, fs_locations_info, fs_status) les attributs suivants DEVRAIENT être disponibles sur un système de fichiers absent. Dans le cas d'attributs RECOMMANDÉS, ils devraient être disponibles au moins au même degré qu'ils sont disponibles sur les systèmes de fichiers présents.

change_policy : cet attribut est utile pour les systèmes de fichiers absents et peut être utile pour indiquer au client quand un des attributs relatifs à la localisation change.

fsid : cet attribut devrait être fourni afin que le client puisse déterminer les frontières du système de fichiers, y compris, en

particulier, la frontière entre systèmes de fichiers présents et absents. Cette valeur doit être différente de celle de tout autre fsid sur le serveur courant et doit n'avoir pas de relation particulière avec les fsid sur aucune destination particulière sur laquelle le client pourrait être dirigé.

mounted_on_fileid : pour les objets au sommet d'un système de fichiers absent, cet attribut doit être disponible. Comme le fileid est dans le système de fichiers parent présent, il ne devrait pas être besoin de faire référence au système de fichiers absent pour fournir cette information.

D'autres attributs NE DEVRAIENT PAS être rendus disponibles pour des systèmes de fichiers absents, même quand il est possible de les fournir. Le serveur ne devrait pas supposer que plus d'informations est toujours mieux et devrait éviter de fournir gratuitement des informations supplémentaires.

Quand une opération GETATTR inclut un gabarit binaire pour un des attributs fs_locations, fs_locations_info, ou fs_status, mais où le gabarit binaire inclut des attributs qui ne sont pas pris en charge, GETATTR ne va pas retourner une erreur, mais va retourner le gabarit des attributs réellement pris en charge avec les résultats.

Le traitement de VERIFY/NVERIFY est similaire à celui de GETATTR en ce que si le gabarit d'attribut n'inclut pas de fs_locations, fs_locations_info, ou fs_status, l'erreur NFS4ERR_MOVED va en résulter. Il diffère en ce que toute apparition dans le gabarit d'attribut d'un attribut non pris en charge pour un système de fichiers absent (et noter que cela va inclure des attributs normalement EXIGÉS) va aussi causer un résultat NFS4ERR_MOVED.

11.4.2 READDIR et systèmes de fichiers absents

Un READDIR effectué quand la bride de fichier courante est dans un système de fichiers absent va résulter en une erreur NFS4ERR_MOVED, car, à la différence du cas de GETATTR, aucune exception de ce genre n'est faite pour READDIR.

Les attributs pour un système de fichiers absent peuvent être récupérés via un READDIR pour un répertoire dans un système de fichiers présent, quand ce répertoire contient les répertoires racines de un ou plusieurs systèmes de fichiers absents. Dans ce cas, le traitement est le suivant :

* Si l'ensemble d'attributs demandé inclut un des attributs fs_locations, fs_locations_info, ou fs_status, alors aller chercher les attributs se fait normalement et aucune indication NFS4ERR_MOVED n'est retournée, même quand l'attribut rdattr_error est demandé.

* Si l'ensemble d'attributs demandé n'inclut pas un des attributs fs_locations, fs_locations_info, ou fs_status, alors si l'attribut rdattr_error est demandé, chaque entrée de répertoire pour la racine d'un système de fichiers absent va rapporter NFS4ERR_MOVED comme valeur de l'attribut rdattr_error.

* Si l'ensemble d'attributs demandé n'inclut aucun des attributs fs_locations, fs_locations_info, fs_status, ou rdattr_error, alors l'occurrence de la racine d'un système de fichiers absent au sein du répertoire va résulter en l'échec du READDIR avec une erreur NFS4ERR_MOVED.

* L'indisponibilité d'un attribut à cause de l'absence d'un système de fichiers, même un qui est ordinairement EXIGÉ, ne résulte pas en une indication d'erreur. L'ensemble d'attributs retourné pour le répertoire racine du système de fichiers absent dans ce cas est simplement restreint à ceux réellement disponibles.

11.5 Utilisations des informations de localisation de système de fichiers

Les attributs de localisation de système de fichiers (c'est-à-dire, fs_locations et fs_locations_info) avec la possibilité de systèmes de fichiers absents, fournissent un certain nombre d'importantes facilités pour un accès fiable, gérable, et adaptable aux données.

Quand un système de fichiers est présent, ces attributs peuvent fournir ce qui suit :

* La localisation de répliques de remplacement à utiliser pour accéder aux mêmes données en cas de défaillance de serveur, de problèmes de communications, ou autres difficultés qui rendent impossible la poursuite de l'accès à la réplique actuelle ou autrement impraticable. Le provisionnement et l'utilisation de telles répliques de remplacement est appelé "réplication" et est discuté au paragraphe 11.5.4.

* La ou les adresses réseau à utiliser pour accéder à l'instance courante de système de fichiers ou ses répliques. L'utilisation de ces informations par le client est discutée au paragraphe 11.5.2.

Dans certaines circonstances, plusieurs répliques peuvent être utilisées simultanément pour fournir de meilleures performances d'accès au système de fichiers en question, bien que le manque de partage d'état entre serveurs puisse être un obstacle à une telle utilisation.

Quand un système de fichiers est présent mais devient absent, les clients peuvent avoir l'opportunité de continuer d'avoir accès à leurs données en utilisant une réplique différente. Dans ce cas, la poursuite d'une tentative d'utiliser les données dans le système de fichiers maintenant absent va résulter en une erreur NFS4ERR_MOVED, et alors la réplique qui succède ou l'ensemble de choix de répliques possibles peut être récupéré et utilisé pour continuer l'accès. Le transfert de l'accès à la localisation de la nouvelle réplique est appelée une "migration" et est discutée au paragraphe 11.5.4.

Quand un système de fichiers est actuellement absent, la spécification de la localisation du système de fichiers fournit un moyen par lequel les systèmes de fichiers situés sur un serveur peuvent être associés à un espace de noms défini par un autre serveur, permettant donc une facilité générale d'espace de noms multi-serveurs. Une désignation d'une telle instance distante, à la place d'un système de fichiers non présent précédemment, est appelé une "pure référence" et est discutée au paragraphe 11.5.6.

Parce que la prise en charge par le client des attributs relatifs à la localisation du système de fichiers est FACULTATIVE, un serveur peut choisir de prendre des mesures pour cacher les événements de migration et de référence à ces clients, en agissant comme un mandataire, par exemple. Le serveur peut déterminer la présence de la prise en charge par le client à partir des arguments de l'opération EXCHANGE_ID (voir le paragraphe 18.35.3).

11.5.1 Combinaison de plusieurs utilisations dans un seul attribut

Un attribut de localisation de système de fichiers va parfois contenir des informations relatives à la localisation de plusieurs répliques, qui peuvent être utilisées de différentes façons :

* Des entrées de localisation de système de fichiers qui se rapportent à l'instance de système de fichiers actuellement utilisée fournissent des informations de multiplexage, permettant au client de trouver des adresses réseau supplémentaires pour accéder à l'instance.
* Des entrées de localisation de système de fichiers qui fournissent des informations sur des répliques auxquelles transférer l'accès.
* D'autres entrées de localisation de système de fichiers qui se rapportent aux répliques disponibles au cas où l'accès à la réplique courante ne serait pas satisfaisant.

Afin de simplifier le traitement par le client et permettre l'accès au meilleur choix de répliques, le serveur devrait respecter les lignes directrices suivantes :

* Toutes les entrées de localisation de système de fichiers qui se rapportent à une seule instance de système de fichiers devraient être adjacentes.
* Les entrées de localisation de système de fichiers qui se rapportent à l'instance actuellement utilisée devraient apparaître en premier.
* Les entrées de localisation de système de fichiers qui se rapportent à la ou aux répliques sur lesquelles une migration se produit devraient apparaître avant les répliques qui sont disponibles pour une utilisation ultérieure si la réplique courante devait devenir inaccessible.

11.5.2 Attributs de localisation de système de fichiers et multiplexage

Le multiplexage *(trunking)* est l'utilisation de plusieurs connexions entre un client et un serveur afin d'augmenter la vitesse du transfert de données. Un client peut déterminer l'ensemble des adresses réseau à utiliser pour accéder à un système de fichiers de différentes façons :

* Quand le nom du serveur est connu du client, il peut utiliser le DNS pour obtenir un ensemble d'adresses réseau à utiliser pour accéder au serveur.

* Le client peut aller chercher l'attribut de localisation du système de fichiers pour le système de fichiers. Cela va fournir le nom du serveur (qui peut être mis dans un ensemble d'adresses réseau en utilisant le DNS) ou un ensemble d'entrées de localisation multiplexables par le serveur. En utilisant cette dernière solution, le serveur peut fournir les adresses qu'il considère comme souhaitables pour les utiliser à accéder au système de fichiers en question. Bien que ces entrées

puissent contenir des numéros d'accès, ces numéros d'accès ne sont pas utilisés pour déterminer les relations de multiplexage. Une fois que les adresses candidates ont été déterminées et qu'un EXCHANGE_ID a été fait avec le serveur approprié, seule la valeur du champ so_major_id retourné par les serveurs en question détermine si une relation de multiplexage existe réellement.

Quand le client va chercher un attribut de localisation pour un système de fichiers, on devrait noter que le client peut rencontrer plusieurs entrées pour un certain nombre de raisons, comme quand il détermine des informations de multiplexage, il peut avoir besoin d'outrepasser des adresses non multiplexables avec des adresses déjà connues.

Le serveur peut fournir des entrées de localisation qui incluent des noms ou des adresses réseau. Il pourrait utiliser cette dernière forme à cause de soucis de sécurité relatifs au DNS ou parce que l'ensemble d'adresses à utiliser pourrait exiger une gestion active par le serveur.

Les entrées de localisation utilisées pour découvrir les adresses candidates à utiliser dans le multiplexage sont sujettes à changement, comme discuté au paragraphe 11.5.7. Le client peut répondre à de tels changements en utilisant des adresses supplémentaires une fois qu'elles sont vérifiées ou en cessant d'utiliser celles existantes. Le serveur peut forcer le client à cesser d'utiliser une adresse en retournant un NFS4ERR_MOVED quand cette adresse est utilisée pour accéder à un système de fichiers. Cela permet un transfert de l'accès de client similaire à une migration, bien que l'accès soit à la même instance de système de fichiers.

11.5.3 Attributs de localisation de système de fichiers et choix de type de connexion

À cause du besoin de prendre en charge plusieurs types de connexions, les clients font face à la question de déterminer le type approprié de connexion à utiliser quand ils établissent une connexion à une certaine adresse réseau de serveur. Dans certains cas, cette question peut être réglée par l'utilisation de la connexion de la facilité "step-up" décrite au paragraphe 18.36 *(Non, il n'y a rien sur "step-up" au 18.36)*. Cependant, parce que il y a des cas où cette facilité n'est pas disponible, le client peut devoir choisir un type de connexion sans possibilité de le changer dans la portée d'une seule connexion.

Les deux attributs de localisation de système de fichiers diffèrent quant aux informations disponibles à cet égard. L'attribut fs_locations ne fournit pas d'informations pour prendre en charge le choix du type de connexion. Par suite, les clients qui prennent en charge plusieurs types de connexion vont devoir tenter d'établir des connexions en utilisant plusieurs types de connexion jusqu'à ce que celle préférée par le client réussisse à être établie.

L'attribut fs_locations_info inclut le fanion FSLI4TF_RDMA, qui est pratique pour un client qui souhaite utiliser RDMA. Quand ce fanion est établi, il indique que la prise en charge de RPC sur RDMA est disponible en utilisant l'entrée de localisation spécifiée. Un client peut établir une connexion TCP et alors convertir cette connexion à utiliser RDMA grâce à la facilité "step-up".

Sans considération de l'attribut particulier utilisé, quand il n'y a pas d'indication qu'une opération step-up peut être effectuée, un client qui prend en charge l'opération RDMA peut établir une nouvelle connexion RDMA, et elle peut être liée à la session déjà établie par la connexion TCP, permettant à la connexion TCP d'être éliminée et la session convertie pour être ensuite utilisée en mode RDMA, si le serveur prend cela en charge.

11.5.4 Duplication de système de fichiers

Les attributs fs_locations et fs_locations_info fournissent d'autres localisations de système de fichiers, à utiliser pour accéder aux données à la place de ou en plus de l'instance courante de système de fichiers. Au premier accès à un système de fichiers, le client devrait obtenir l'ensemble de localisations de remplacement en interrogeant l'attribut fs_locations ou fs_locations_info, ce dernier étant préféré.

En cas de défaillances de serveur, de problèmes de communications, ou d'autres difficultés rendant la poursuite de l'accès au système de fichiers courant impossible ou autrement impraticable, le client peut utiliser les localisations de remplacement comme moyen de poursuivre l'accès à ses données.

Les localisations de remplacement peuvent être des répliques physiques des données du système de fichiers (normalement en lecture seule) complétées par une possible propagation de mises à jour asynchrones. Autrement, elles peuvent fournir l'utilisation de diverses formes de regroupement de serveurs dans lesquelles plusieurs serveurs fournissent des moyens de remplacement pour accéder au même système de fichiers physique. Comment la différence entre répliques affecte les

transitions de système de fichiers peut être représenté au sein des attributs fs_locations et fs_locations_info, et comment le client traite les questions de transition de système de fichiers va être discuté en détails dans les paragraphes suivants.

Bien que les attributs de localisation fournissent des informations sur la nature de la transition inter-répliques, de nombreux aspects de la sémantique de possibles mises à jour asynchrones ne sont actuellement pas décrites par le protocole, ce qui rend nécessaire aux clients qui utilisent la réplication de commuter entre les répliques subissant un changement pour se familiariser avec la sémantique de l'approche de mise à jour utilisée. Du fait de ce manque de spécificité, de nombreuses applications peuvent trouver l'utilisation de la migration plus appropriée parce que un serveur peut propager à la nouvelle réplique toutes les mises à jour faites avant un instant établi au titre de l'événement de migration.

11.5.4.1 Multiplexage de système de fichiers présenté comme une duplication

Dans certaines situations, une entrée de localisation de système de fichiers peut indiquer un chemin d'accès de système de fichiers à utiliser comme localisation de remplacement, où le multiplexage, plutôt que la réplication, est à utiliser. Les situations dans lesquelles ceci est approprié sont limitées à celles où les deux propositions suivantes sont vraies :

* Les deux localisations de système de fichiers (c'est-à-dire, celle sur laquelle l'attribut de localisation est obtenu et celle spécifiée dans l'entrée de localisation du système de fichiers) désignent les mêmes localisations au sein de leurs espaces de noms respectifs d'un seul serveur.
* Les deux adresses réseau de serveur (c'est-à-dire, celle utilisée pour obtenir l'attribut de localisation et celle spécifiée dans l'entrée de localisation du système de fichiers) désignent le même serveur (comme indiqué par la même valeur du champ so_major_id du champ eir_server_owner retourné en réponse à EXCHANGE_ID).

Quand ces conditions tiennent, les opérations qui utilisent les deux chemins d'accès sont généralement multiplexées, bien que le multiplexage puisse être interdit quand l'attribut fs_locations_info est utilisé :

* Quand l'attribut fs_locations_info montre que les deux entrées n'ont pas la même classe d'utilisation simultanée, le multiplexage est interdit, et les deux chemins d'accès ne peuvent pas être utilisés ensemble. Dans ce cas, les deux chemins peuvent être utilisés à la suite avec une non transition d'activité exigée de la part du client, et toute transition entre les chemins d'accès est transparente. En transférant l'accès de l'un à l'autre, le client agit comme si la communication était interrompue, établissant une nouvelle connexion et éventuellement une nouvelle session pour continuer l'accès au même système de fichiers.

* Noter que pour deux entrées de localisation de cette sorte, toutes les informations dans l'attribut fs_locations_info qui indiquent le besoin d'une activité spéciale de transition, c'est-à-dire, l'apparition des deux entrées de localisation de système de fichiers avec des brides, identifiant de fichier, vérificateur d'écriture, changement, et classes de readdir différentes, indiquent un problème sérieux. Le client, si il permet la transition à l'instance de système de fichiers, ne doit pas traiter une transition comme transparente. Le serveur NE DEVRAIT PAS indiquer que ces deux entrées (pour le même système de fichiers sur le même serveur) appartiennent à des bride, identifiant de fichier, vérificateur d'écriture, changement, et classe de readdir différentes, si les deux entrées sont montrées comme appartenant à la même classe d'utilisation simultanée. Ces situations ont été reconnues par la [RFC5661], bien que ce document ne fasse pas de mention explicite du multiplexage :

* Elle traitait la situation qu'on décrit comme multiplexage comme une utilisation simultanée de deux instances distinctes de système de fichiers, bien que, dans le cadre explicatif maintenant utilisé pour décrire la situation, le cas soit celui dans lequel un seul système de fichiers est accédé par deux adresses différentes multiplexées.

* Elle traitait la situation dans laquelle deux chemins sont à utiliser en série comme une sorte particulière de "transition transparente". Cependant, dans le cadre explicatif maintenant utilisé pour catégoriser les situations de transition, ceci soit considéré comme un cas de "transition de point d'extrémité de réseau" (voir le paragraphe 11.9).

11.5.5 Migration de système de fichiers

Quand un système de fichiers est présent et devient inaccessible en utilisant le chemin d'accès courant, le protocole NFSv4.1 fournit un moyen par lequel les clients peuvent avoir l'opportunité de continuer d'accéder à leurs données. Cela peut impliquer d'utiliser un chemin d'accès différent à la réplique existante ou de fournir un chemin à une réplique différente. Le nouveau chemin d'accès ou la localisation de la nouvelle réplique est spécifié par un attribut de localisation de système de fichiers. La migration d'accès qui s'ensuit inclut la capacité de conserver les verrous à travers la transition. Selon les circonstances, cela peut impliquer :

* La poursuite de l'utilisation de l'identifiant de client existant quand on accède à la réplique courante en utilisant un nouveau chemin d'accès.

* L'utilisation d'une réclamation de verrou, tirant parti de la période de grâce par système de fichiers.
* L'utilisation d'une migration d'état transparente.

Normalement, un client va accéder au système de fichiers en question, obtenir une erreur NFS4ERR_MOVED, et alors utiliser un attribut de localisation de système de fichiers pour déterminer le nouveau chemin d'accès pour les données. Quand fs_locations_info est utilisé, des informations supplémentaires vont être disponibles pour définir la nature du traitement par le client de la transition à un nouveau serveur.

Dans la plupart des cas, les serveurs vont choisir de faire migrer tous les clients en utilisant un système de fichiers particulier à une réplique suivante au même moment pour éviter le cas où différents clients mettent à jour des répliques différentes. Cependant, la migration d'un client individuel peut aider à assurer l'équilibrage de charge, pour autant que les répliques en question soient telles qu'elles représentent les mêmes données, comme décrit au paragraphe 11.11.8.

* Dans le cas où il n'y a pas de transition entre les répliques (c'est-à-dire, seulement un changement du chemin d'accès) il n'y a pas de difficulté particulière à utiliser ce mécanisme pour effectuer l'équilibrage de charge.

* Dans le cas où les deux répliques sont suffisamment coordonnées pour permettre l'accès cohérent simultané d'un seul client aux deux, il n'y a, en général, pas d'obstacle à l'utilisation de la migration de clients particuliers pour effectuer l'équilibrage de charge. Généralement, une telle utilisation simultanée implique la coopération entre les serveurs pour assurer que les verrous accordés sur les deux répliques coordonnées ne peuvent pas entrer en conflit et peuvent rester effectifs quand ils sont transférés à une réplique commune.

* Dans le cas où un grand ensemble de clients accède à un système de fichiers en lecture seule, il peut être utile de faire migrer simultanément tous les clients avec accès en écriture, tout en utilisant l'équilibrage de charge sur l'ensemble de copies en lecture seule, pour autant que soient suivies les règles du paragraphe 11.11.8, qui sont conçues pour empêcher l'inversion des données.

Dans d'autres cas, le client pourrait ne pas avoir des garanties suffisantes de similarité ou cohérence des données pour fonctionner correctement (par exemple, les données dans les deux répliques sont similaires mais pas identiques) et la possibilité que des clients différents mettent à jour des répliques différentes peut exacerber les difficultés, rendant l'utilisation de l'équilibrage de charge une entreprise périlleuse dans de telles situations.

Le protocole ne spécifie pas comment le système de fichiers va être déplacé entre serveurs ou comment les mises à jour de plusieurs répliques vont être coordonnées. Il est prévu qu'un certain nombre de mécanismes de coordination différents de serveur à serveur pourraient être utilisés, le choix étant laissé au serveur qui le met en œuvre. Le protocole NFSv4.1 spécifie la méthode utilisée pour communiquer l'événement de migration entre client et serveur.

Dans le cas de diverses formes de regroupement de serveurs, la nouvelle localisation peut être un autre serveur fournissant l'accès au même système de fichiers physique. Les responsabilités du client dans le traitement de cette transition vont dépendre de si une commutation s'est produite entre les répliques et des moyens que le serveur a choisi pour fournir la continuité de l'état de verrouillage. Ces questions vont être discutées en détails ci-après.

Bien que une seule localisation suivante soit typique, plusieurs localisations peuvent être fournies. Quand plusieurs localisations sont fournies, le client va normalement utiliser la première fournie. Si elle est inaccessible pour une raison quelconque, les suivantes peuvent être utilisées. Dans ce cas, le client pourrait considérer que la transition à la nouvelle réplique est un événement de migration, même si certains des serveurs impliqués pourraient ne pas savoir que l'utilisation du serveur était inaccessible. Dans ce cas, un client pourrait perdre l'accès à l'état de verrouillage par suite du transfert d'accès.

Quand une localisation de remplacement est désignée comme cible de la migration, elle doit désigner les mêmes données (avec les métadonnées au même degré indiqué par l'attribut fs_locations_info). Lorsque les systèmes de fichiers sont écrivables, un changement fait au système de fichiers original doit être visible sur toutes les cibles de migration. Lorsque un système de fichiers n'est pas écrivable mais représente une copie en lecture seule (éventuellement mise à jour périodiquement) d'un système de fichiers écrivable, des exigences similaires s'appliquent à la propagation des mises à jour. Tout changement visible dans le système de fichiers original doit déjà être effectué sur toutes les cibles de migration, pour éviter toute possibilité qu'un client, en effectuant une transition à la cible de migration, voit une inversion dans l'état du système de fichiers.

11.5.6 Référents

Les référents permettent au serveur d'associer une entrée d'espace de noms de système de fichiers située sur un serveur avec un système de fichiers situé sur un autre serveur. Quand cela inclut l'utilisation de purs référents, les serveurs ont un moyen pour placer un système de fichiers dans une localisation au sein de l'espace de noms essentiellement sans égard à sa localisation physique sur un serveur particulier. Cela permet à un seul serveur ou à un ensemble de serveurs de présenter un espace de noms multi-serveurs qui englobe des systèmes de fichiers situés sur une plus large gamme de serveurs. Certaines utilisations probables de cette facilité incluent l'établissement d'espaces de noms à l'échelle d'un site ou d'une organisation, avec la possibilité éventuelle de les combiner en un espace de noms vraiment global, comme celui fourni par le système de fichiers de Andrew (AFS, *Andrew File System*) [AFS].

Les référents se produisent quand un client détermine, après une première référence à une position dans l'espace de noms courant, qu'il fait partie d'un nouveau système de fichiers et que le système de fichiers est absent. Quand cela se produit, normalement à réception de l'erreur NFS4ERR_MOVED, la localisation réelle du système de fichiers peut être déterminée en allant chercher un attribut de localisation.

L'attribut de localisation de système de fichiers peut désigner une seule localisation de système de fichiers ou plusieurs localisations de système de fichiers, à choisir sur la base des besoins du client. Le serveur, dans l'attribut fs_locations_info, peut spécifier des priorités à associer à divers choix de localisation de système de fichiers. Le serveur peut allouer différentes priorités aux différentes localisations rapportées aux clients individuels, afin de s'adapter à la localisation physique du client ou effectuer l'équilibrage de charge. Quand les deux systèmes de fichiers en lecture seule et en lecture-écriture sont présents, certaines des localisations en lecture seule pourraient n'être pas absolument à jour (comme elles devraient l'être en cas de réplication et migration). Les serveurs peuvent aussi spécifier des localisations de système de fichiers qui incluent des variables substituées par le client de sorte que différents clients se réfèrent à différents systèmes de fichiers (avec des contenus de données différents) sur la base des attributs de client comme l'architecture de CPU.

Si l'attribut fs_locations_info mentionne plusieurs cibles possibles, les relations entre elles peuvent être importantes pour que le client choisisse celle à utiliser. Les règles spécifiées au paragraphe 11.5.5 concernant plusieurs cibles de migration s'appliquent aussi à ces répliques multiples. Par exemple, le client pourrait préférer une cible écrivable sur un serveur qui a des répliques écrivables supplémentaires auxquelles il pourrait ensuite se commuter. Noter que, comme il est distingué du cas de la réplication, il n'est pas besoin de traiter le cas de la propagation de mises à jour faites par le client actuel, car celui-ci n'a pas accédé au système de fichiers en question.

L'utilisation d'espaces de noms multi-serveurs est permise par NFSv4.1 mais n'est pas exigée. L'utilisation d'espaces de noms multi-serveurs et leur portée vont dépendre de l'application utilisée et des préférences de l'administration du système.

Les espaces de noms multi-serveurs peuvent être établis par un seul serveur fournissant un large ensemble de purs référents à tous les systèmes de fichiers inclus. Autrement, un seul espace de noms multi-serveurs peut être segmenté administrativement avec des systèmes de fichiers référents séparés (sur des serveurs séparés) pour chaque portion administrée séparément de l'espace de noms. Le système de fichiers référent de niveau supérieur ou tout segment peut utiliser des systèmes de fichiers référents dupliqués pour une meilleure disponibilité.

Généralement, les espaces de noms multi-serveurs sont pour la plupart uniformes, en ce que les mêmes données disponibles à un client à une localisation donnée dans l'espace de noms sont rendues disponibles à tous les clients à cette localisation d'espace de noms. Cependant, il existe des facilités qui permettent à différents clients d'être dirigés sur différents ensembles de données, pour des raisons comme de permettre l'adaptation à des caractéristiques de client comme l'architecture de CPU. Ces facilités sont décrites au paragraphe 11.17.3.

Noter qu'il est possible, quand on fournit un espace de noms uniforme, de fournir des entrées de localisation différentes à différents clients afin de fournir à chaque client une copie des données physiquement les plus proches de lui ou autrement optimiser l'accès (par exemple, fournir l'équilibrage de charge).

11.5.7 Changements dans l'attribut de localisation de système de fichiers

Bien que les clients aillent normalement chercher un attribut de localisation de système de fichiers quand ils accèdent pour la première fois à un système de fichiers et quand NFS4ERR_MOVED est retourné, un client peut choisir d'aller chercher l'attribut périodiquement ; dans ce cas, la valeur récupérée peut changer dans le temps.

Pour les clients qui ne sont pas prêts à accéder à plusieurs répliques simultanément (voir le paragraphe 11.11.1) le traitement des divers cas de changement de localisation est le suivant :

* Les changements dans la liste de répliques ou dans les adresses réseau associées à des répliques n'exigent pas une action immédiate. Le client va normalement mettre à jour sa liste de répliques pour refléter les nouvelles informations.

* Les ajouts à la liste des adresses réseau pour l'instance actuelle de système de fichiers n'ont pas besoin d'une action rapide. Cependant, pour préparer un événement de migration ultérieur, le client peut choisir de prendre note de la nouvelle adresse et de l'utiliser alors chaque fois qu'il a besoin de changer l'accès à une nouvelle réplique.

* Les suppressions de la liste des adresses réseau pour l'instance courante de système de fichiers n'exigent pas que le client cesse immédiatement d'utiliser les chemins d'accès existants, bien que de nouvelles connexions ne soient pas à établir sur les adresses qui ont été supprimées. Cependant, les clients peuvent choisir d'agir sur ces suppressions en préparant un éventuel glissement d'accès, qui devient inévitable aussitôt que le serveur retourne NFS4ERR_MOVED pour indiquer qu'un chemin d'accès réseau particulier n'est pas utilisable pour accéder au système de fichiers courant.

Pour les clients qui sont prêts à accéder à plusieurs répliques simultanément, les cas supplémentaires suivants doivent être traités. Comme dans les cas discutés ci-dessus, les changements dans l'ensemble de répliques n'appellent pas une action rapide, bien que le client ait l'option d'ajuster son accès même en l'absence de difficultés qui conduiraient au choix d'une nouvelle réplique.

* Quand une nouvelle réplique est ajoutée, à laquelle on peut accéder simultanément avec une actuellement utilisée, le client est libre d'utiliser immédiatement la nouvelle réplique.

* Quand une réplique actuellement utilisée est supprimée de la liste, le client n'a pas besoin de cesser de l'utiliser immédiatement. Cependant, comme le serveur peut ensuite forcer la cessation de l'utilisation (en retournant NFS4ERR_MOVED) les clients pourraient décider de limiter le besoin d'un transfert d'état ultérieur. Par exemple, de nouvelles ouvertures pourraient être faites sur d'autres répliques, plutôt que sur une non présente dans la liste.

11.6 Multiplexage sans information de localisation de système de fichiers

Dans les situations où l'accès à un système de fichiers est fait en utilisant deux adresses multiplexables par le serveur (comme indiqué par la même valeur du champ so_major_id du champ eir_server_owner retourné en réponse au EXCHANGE_ID) l'accès multiplexé est permis même si il ne pourrait pas y avoir d'entrées de localisation indiquant spécifiquement l'utilisation du multiplexage pour ce système de fichiers.

Cette situation a été reconnue par la [RFC5661], bien que ce document ne mentionne pas explicitement le multiplexage et traite la situation comme une d'utilisation simultanée de deux instances distinctes de système de fichiers. Dans le cadre explicatif maintenant utilisé pour décrire la situation, le cas est un de ceux dans lesquels un seul système de fichiers est accédé par deux adresses multiplexées différentes.

11.7 Utilisateurs et groupes dans un espace de noms multi-serveurs

Comme dans le cas d'un environnement d'un seul serveur (voir le paragraphe 5.9) quand un nom de propriétaire ou de groupe de la forme "id@domaine" est alloué à un fichier, il y a une promesse implicite de retourner cette même chaîne quand l'attribut correspondant est ensuite interrogé. Dans le cas d'un espace de noms multi-serveurs, cette même promesse s'applique même si des frontières de serveur ont été franchies. De même, quand l'attribut "owner" d'un fichier est déduit du principal de sécurité qui a créé le fichier, cet attribut devrait avoir la même valeur même si l'interrogation se produit sur un serveur différent de celui de la création du fichier.

De même, l'ensemble de principaux de sécurité reconnu par tous les serveurs participants doit être le même, chacun de ces principaux ayant les mêmes accréditifs, sans considération du serveur particulier accédé.

Afin de satisfaire ces exigences, l'établissement d'espace de noms multi-serveurs devra limiter les serveurs inclus pour que :

* Dans tous les cas où plus d'un seul domaine est pris en charge, les exigences de la [RFC8000] soient respectées.
* Tous les serveurs prennent en charge un ensemble commun de domaines qui inclut tous les domaines que les clients utilisent et s'attendent à voir retournés comme portion domaine d'un propriétaire ou groupe de la forme "id@domaine". Noter que, bien que cet ensemble consiste le plus souvent en un seul domaine, il est possible que plusieurs domaines soient pris en charge.
* Tous les serveurs, pour chaque domaine qu'ils prennent en charge, acceptent le même ensemble d'identifiants d'utilisateur et de groupe comme valides.
* Tous les serveurs reconnaissent le même ensemble de principaux de sécurité. Pour chaque principal, le même accréditif

est exigé, indépendamment du serveur accédé. De plus, les membres du groupe pour chacun de ces principaux doivent être les mêmes, indépendamment du serveur accédé.

Noter qu'il n'est pas exigé en général que les utilisateurs correspondant à des principaux de sécurité particuliers aient la même représentation locale sur chaque serveur, même si c'est le plus souvent le cas.

Quand AUTH_SYS est utilisé, les exigences supplémentaires suivantes doivent être satisfaites :
* Un seul domaine NFSv4 peut être pris en charge par l'utilisation de AUTH_SYS.
* La représentation "locale" de tous les propriétaires et groupes doit être la même sur tous les serveurs. Le mot "local" est utilisé ici car c'est la façon dont les utilisateurs numériques et les identifiants de groupe sont décrits au paragraphe 5.9. Cependant, quand AUTH_SYS ou des chaînes de propriétaires ou groupes numériques sont utilisés, ces identifiants ne sont pas vraiment locaux, car ils sont connus des clients ainsi que du serveur.

De même, quand des chaînes d'identifiants d'utilisateur et groupe numérique sont utilisés, la représentation "locale" de tous les propriétaires et groupes doit être la même sur tous les serveurs, même quand AUTH_SYS n'est pas utilisé.

11.8 Considérations supplémentaires sur le côté client

Quand les clients utilisent des serveurs qui mettent en œuvre des référents, la duplication, et la migration, on devrait faire attention qu'un utilisateur qui monte un système de fichiers incluant un référant ou un système de fichiers relocalisé continue d'avoir une image cohérente de ce système de fichiers côté utilisateur en dépit du fait qu'il contient un certain nombre de systèmes de fichiers côté serveur qui peuvent être sur différents serveurs.

Une question importante est la navigation vers l'amont de la racine d'un système de fichiers côté serveur à son parent (spécifié comme ".." dans UNIX) dans le cas où il passe à ce système de fichiers par suite d'une référence, migration, ou une transition par suite de duplication. Quand le client en est à ce point, et qu'il a besoin de remonter au parent, il doit revenir au parent tel que vu au sein de l'espace de noms multi-serveurs plutôt que d'envoyer une opération LOOKUPP au serveur, qui aurait pour résultat le parent dans l'espace de noms d'un seul serveur de ce serveur. Pour faire cela, le client doit se rappeler des brides de fichier qui représentent de telles racines de système de fichiers et les utiliser au lieu d'envoyer une opération LOOKUPP au serveur courant. Cela va permettre au client de présenter aux applications un espace de noms cohérent, où la navigation vers l'amont et vers l'aval sont cohérentes.

Une autre question concerne le rafraîchissement des localisations de référents. Quand il y a une utilisation intensive de référents, ils peuvent changer avec les changements de configurations de serveur. On s'attend à ce que les clients mettent en antémémoire les informations relatives à la traversée de référents afin que les futures demandes côté client soient résolues localement sans communication avec le serveur. Ceci est généralement enraciné dans la mise en antémémoire de recherche de noms côté client. Les clients devraient périodiquement purger ces données pour les points de référence afin de détecter les changements d'informations de localisation. Quand l'attribut change_policy change pour des répertoires qui détiennent des entrées de référence ou pour les entrées de référence elles-mêmes, les clients devraient considérer que toutes les informations de référence associées en antémémoire sont périmées.

11.9 Vue d'ensemble des transitions d'accès au fichier

Les transitions d'accès au fichier sont de deux types :
* Celles qui impliquent une transition de l'accès à la réplique actuelle à une autre en connexion avec la duplication ou la migration. Comment les traiter est discuté au paragraphe 11.11.
* Celles dans lesquelles l'accès à l'instance courante de système de fichiers est conservé, alors que le chemin de réseau utilisé pour accéder à cette instance est changé. Ce cas est discuté au paragraphe 11.10.

11.10 Effectuer des transitions de point d'extrémité réseau

Les points d'extrémité utilisés pour accéder à une instance particulière de système de fichiers peuvent changer d'un certain nombre de façons, mentionnées ci-dessous. Dans chacun de ces cas, le même identifiant de système de fichiers *(fsid)* les identifiants de client, les brides de fichier, et les identifiants d'état *(stateid)* sont utilisés pour continuer l'accès, avec une continuité de l'état de verrouillage. Dans de nombreux cas, les mêmes sessions peuvent aussi être utilisées.

L'action appropriée dépend des adresses de l'ensemble de remplacement qui sont disponibles à l'utilisation (c'est-à-dire, les points d'extrémité de serveur qui sont multiplexables par le serveur avec une précédemment utilisée).

* Quand l'utilisation d'une adresse particulière va cesser, et qu'il y a aussi une autre adresse actuellement utilisée qui est multiplexable avec elle par le serveur, les demandes qui auraient été produites sur l'adresse dont l'utilisation va cesser peuvent être produites sur la ou les adresses restantes. Quand une adresse est multiplexable par le serveur mais non multiplexable par la session avec l'adresse dont l'utilisation va cesser, la demande pourrait devoir être modifiée pour refléter le fait qu'une session différente va être utilisée.

* Quand l'utilisation d'une connexion particulière va cesser, comme indiqué par la réception de NFS4ERR_MOVED quand on utilise cette connexion, mais que cette adresse est toujours indiquée comme accessible selon les entrées de localisation appropriées du système de fichiers, il est probable que des demandes peuvent être produites sur une nouvelle connexion de type différent une fois que cette connexion est établie. Comme deux points d'extrémité de serveur non spécifiques de l'accès qui partagent une adresse réseau sont par nature multiplexables par la session, le client peut utiliser BIND_CONN_TO_SESSION pour accéder à la session existante avec la nouvelle connexion.

* Quand il n'y a pas d'adresse potentielle de remplacement en utilisation, mais qu'il y a des adresses valides multiplexables par la session avec celle dont l'utilisation va cesser, le client peut utiliser BIND_CONN_TO_SESSION pour accéder à la session existante en utilisant la nouvelle adresse. Bien que la session cible soit généralement accessible, il peut y avoir de rares situations où cette session n'est plus accessible quand une tentative est faite de lui lier la nouvelle connexion. Dans ce cas, le client peut créer une nouvelle session pour permettre de continuer d'accéder à l'instance existante en utilisant la nouvelle connexion, fournissant l'utilisation des brides de fichier, identifiants d'états, et identifiants de client existants tout en assurant la continuité de l'état de verrouillage.

* Quand il n'y a pas d'adresse potentielle de remplacement en utilisation, et qu'il n'y a pas d'adresse valide multiplexable par la session avec celle dont l'utilisation va cesser, d'autres adresses multiplexables par le serveur peuvent être utilisées pour fournir la continuité d'accès. Bien que l'utilisation de CREATE_SESSION soit disponible pour fournir la continuité d'accès à l'instance existante, les serveurs ont l'option de fournir la continuité d'accès à la session existante à travers le nouveau chemin d'accès réseau d'une façon similaire à celle fournie par la migration de session (voir le paragraphe 11.12). Pour tirer parti de cette possibilité, les clients peuvent effectuer un BIND_CONN_TO_SESSION initial, comme dans le cas précédent, et n'utiliser CREATE_SESSION que si cela échoue.

11.11 Effectuer des transitions de système de fichiers

Il y a une gamme de situations dans lesquelles un changement doit être effectué dans l'ensemble des répliques utilisées pour accéder à un système de fichiers particulier. Certaines peuvent impliquer une expansion ou une contraction de l'ensemble des répliques utilisées comme discuté au paragraphe 11.11.1.

Pour les raisons expliquées dans cette section, la plupart des transitions vont impliquer une transition d'une seule réplique à une réplique de remplacement correspondante. Quand on effectue une transition de réplique, certains types de partage entre les répliques peuvent affecter le traitement de la transition comme décrit aux paragraphes 11.11.2 à 11.11.8. L'attribut fs_locations_info fournit des informations utiles pour permettre au client de déterminer le degré de partage inter-répliques.

À l'égard de certains types d'état, le degré de continuité à travers la transition dépend de l'occasion qui invite à la transition, avec des transitions initiées par les serveurs (c'est-à-dire, migration) offrant beaucoup plus de portée pour une transition non perturbante que les cas où le client glisse de son propre chef son accès sur une autre réplique (c'est-à-dire, duplication). Ce problème s'applique potentiellement à l'état de verrouillage et à l'état de session, qui sont traités comme suit :

* Une introduction à de possibles moyens de fournir la continuité dans ces domaines est donnée au paragraphe 11.11.9.
* Une migration d'état transparente est introduite au paragraphe 11.12, qui traite aussi d'un possible transfert d'état de session.
* Le traitement par le client des transitions, incluant de déterminer comment traiter les diverses significations que le serveur pourrait prendre pour fournir une continuité effective de l'état de verrouillage, est discuté au paragraphe 11.13.
* Les responsabilités des serveurs de source et destination pour effectuer une migration d'état transparente d'état de verrouillage et de session sont discutées au paragraphe 11.14.

11.11.1 Transitions de système de fichiers et accès simultané

L'attribut fs_locations_info (décrit au paragraphe 11.17) peut indiquer que deux répliques peuvent être utilisées simultanément, bien que certaines situations dans lesquelles l'accès simultané est permis soient décrites de façon plus appropriée comme des instances de multiplexage (voir le paragraphe 11.5.4.1). Bien que les situations dans lesquelles plusieurs répliques peuvent être accédées simultanément soient assez similaires à celles dans lesquelles une seule réplique est accédée par plusieurs adresses réseau, il y a des différences importantes car l'état de verrouillage n'est pas partagé entre

plusieurs répliques.

À cause de cette différence dans le traitement d'état, de nombreux clients n'auront pas la capacité de tirer parti du fait que de telles répliques représentent les mêmes données. Ces clients ne seront pas prêts à utiliser plusieurs répliques simultanément mais vont accéder à chaque système de fichiers en utilisant seulement une seule réplique, malgré que les répliques choisies pourraient rendre disponibles plusieurs adresses multiplexables par le serveur.

Les clients qui sont prêts à utiliser plusieurs répliques simultanément peuvent cependant diviser les ouvertures parmi les répliques qu'ils choisissent. Une fois que ce choix est fait, toutes les transitions suivantes vont traiter l'ensemble de l'état de verrouillage associé à chaque réplique comme une seule entité.

Par exemple, si une des répliques devient indisponible, l'accès va être transféré à une réplique différente, qui est aussi capable d'accès simultané avec celle encore utilisée.

Quand il n'y a pas une telle réplique, la transition peut être à la réplique déjà en usage. À ce point, le client a un choix entre fusionner l'état de verrouillage pour les deux répliques sous l'égide de la seule réplique en usage ou les traiter séparément jusqu'à ce qu'une autre réplique capable d'accès simultané se présente.

11.11.2 Brides de fichier et transitions de système de fichiers

Il y a un certain nombre de façons de traiter les brides de fichier à travers une transition de système de fichiers. Elles peuvent être divisées en deux grandes classes selon que si les deux systèmes de fichiers à travers lesquels se fait la transition partagent suffisamment d'état pour effectuer une sorte de continuité du traitement de système de fichiers.

Quand il n'y a pas une telle coopération dans l'allocation de bride de fichier, les deux systèmes de fichiers sont rapportés comme étant dans des classes de bride différentes. Dans ce cas, toutes les brides de fichier sont supposées expirer au titre de la transition de système de fichiers. Noter que ce comportement ne dépend pas de l'attribut fh_expire_type et subroge la spécification du bit FH4_VOL_MIGRATION, qui affecte seulement le comportement quand fs_locations_info n'est pas disponible.

Quand il y a coopération dans l'allocation de bride de fichier, les deux systèmes de fichiers sont rapportés comme étant dans la même classe de brides. Dans ce cas, des brides de fichier persistantes restent valides après la transition de système de fichiers, tandis que les brides de fichier volatiles (à l'exclusion de celles qui sont seulement volatiles du fait du bit FH4_VOL_MIGRATION) sont sujettes à expiration sur le serveur cible.

11.11.3 Identifiants de fichier et transitions de système de fichiers

Dans NFSv4.0, la question de continuité des identifiants de fichiers en cas de transition d'un système de fichiers n'était pas traitée. L'attente générale était que dans les situations où les deux instances de système de fichiers sont créées par un seul fabricant utilisant une sorte de copie d'image de système de fichiers, les identifiants de fichiers seraient cohérents à travers la transition, tandis que dans les transitions analogues multi-fabricants, elles ne le seraient pas. Cela crée des difficultés, en particulier pour le client sans connaissance particulière des mécanismes de transition adoptés par le serveur. Noter que bien que l'identifiant de fichier ne soit pas un attribut EXIGÉ, de nombreux serveurs prennent en charge les identifiants de fichier et de nombreux clients fournissent des API qui en dépendent.

Il est important de noter qu'alors que les clients eux-mêmes peuvent ne pas avoir de problème avec un identifiant de fichier qui change par suite d'un événement de transition de système de fichiers, les applications ont normalement accès à l'identifiant de fichier (par exemple, via stat). Le résultat est qu'une application peut fonctionner parfaitement bien si il n'y a pas de transition d'instance de système de fichiers ou si une telle transition est parmi des instances créées par un seul fabricant, tout en étant incapable de faire face à la situation où une transition multi-fabricants se produit au mauvais moment.

Fournir les mêmes identifiants de fichiers dans un environnement multi-fabricants (plusieurs fabricants de serveurs) a généralement été tenu pour assez difficile. Bien qu'il reste du travail à faire, on doit souligner que cette difficulté est partiellement auto-imposée. Les serveurs ont normalement identifié l'identifiant de fichier avec le numéro inode, c'est-à-dire avec une quantité utilisée pour trouver le fichier en question. Cette identification soulève des difficultés particulières pour la migration d'un système de fichiers entre fabricants où l'allocation du même indice à un fichier donné peut n'être pas possible. Noter ici qu'il n'est pas exigé qu'un identifiant de fichier soit utile pour trouver le fichier en question, mais seulement qu'il soit unique au sein du système de fichiers. Les serveurs prêts à accepter un identifiant de fichier comme une

seule pièce de métadonnées et de le mémoriser à part de la valeur utilisée pour indexer les informations du fichier peuvent relativement facilement maintenir une valeur de fileid à travers un événement de migration, permettant un événement de migration vraiment transparent.

En tout cas, lorsque les serveurs peuvent fournir la continuité des identifiants de fichiers, ils le devraient, et le client devrait être capable de découvrir qu'une telle continuité est disponible et prendre l'action appropriée. Les informations sur la continuité (ou son manque) des identifiants de fichiers à travers une transition de système de fichiers est représentée en spécifiant si les systèmes de fichiers en question sont de la même classe d'identifiant de fichier.

Noter que quand des identifiants de fichier cohérents n'existent pas à travers une transition (parce qu'il n'y a pas de continuité des identifiants de fichiers ou parce que fileid est un attribut non pris en charge sur une des instances impliquées) et qu'il n'y a pas de brides de fichier fiables à travers un événement de transition (parce que il n'y a pas de continuité de bride de fichier ou parce que les brides de fichier sont volatiles) le client est dans une position où il ne peut pas vérifier que les fichiers auxquels il accédait avant la transition sont les mêmes objets. Il est forcé de supposer qu'aucun objet n'a été renommé, et, sauf si il y a des garanties qui fournissent cela (par exemple, le système de fichiers est en lecture seule) il peut survenir des problèmes pour les applications. Donc, l'utilisation de telles configurations devrait être limitée aux situations où les problèmes que cela peut causer peuvent être tolérés.

11.11.4 Identifiants de système de fichiers et transitions de système de fichiers

Comme les identifiants de système de fichier *(fsid)* sont généralement seulement uniques sur la base du serveur, il est probable qu'ils vont changer durant une transition de système de fichiers. Les clients ne devraient pas rendre les fsid reçus du serveur visibles aus applications car ils peuvent n'être pas uniques au monde, et parce que ils peuvent changer durant un événement de transition de système de fichiers. Les applications sont mieux servies si elles sont isolées de telles transitions dans la mesure du possible.

Bien que normalement un seul système de fichiers source transite à un seul système de fichiers cible, il y a une disposition pour partager un seul système de fichiers source en plusieurs systèmes de fichiers cibles, en spécifiant le fanion FSLI4F_MULTI_FS.

11.11.4.1 Partage de système de fichiers

Quand une transition de système de fichiers est faite et que fs_locations_info indique que le système de fichiers en question pourrait être partagé en plusieurs systèmes de fichiers (via le fanion FSLI4F_MULTI_FS) le client DEVRAIT faire des GETATTR pour déterminer l'attribut fsid sur tous les objets connus dans le système de fichiers qui subit la transition pour déterminer les frontières du nouveau système de fichiers.

Les clients pourraient choisir de maintenir les fsid passés aux applications existantes en transposant tous les fsid pour le systèmes de fichiers descendant en le fsid commun utilisé pour le système de fichiers original.

Partager un système de fichiers peut être fait sur une transition entre systèmes de fichiers de même classe d'identifiants de fichier, car le fait que les identifiants de fichier sont uniques dans le système de fichiers source assure qu'ils vont être uniques dans chaque système de fichiers cible.

11.11.5 Attribut de changement et transitions de système de fichiers

Comme l'attribut de changement est défini comme spécifique du serveur; les attributs de changement cherchés sur un serveur sont normalement présumés être invalides sur un autre serveur. Une telle présomption est perturbatrice parce que elle invaliderait tous les attributs de changement en antémémoire, obligeant à retourner les chercher. Encore plus perturbant, l'absence de toute assurance de continuité pour l'attribut de changement signifie que même si la même valeur est restituée sur une reprise, aucune conclusion ne peut être tirée sur si l'objet en question a changé. L'attribut de changement identique pourrait être simplement un artifice d'un fichier modifié avec un algorithme de construction d'attribut de changement différent, avec ce nouvel algorithme qui se trouve juste résulter en une valeur de changement identique.

Quand les deux systèmes de fichiers ont des formats d'attribut de changement cohérents, et que ce fait est communiqué au client en rapportant dans la même classe de changement, le client peut supposer une continuité de construction d'attribut de changement et traiter cette situation juste comme il aurait fait sans aucune transition de système de fichiers.

11.11.6 Vérificateurs d'écriture et transitions de système de fichiers

Dans une transition de système de fichiers, les deux systèmes de fichiers pourraient coopérer dans le traitement de données écrites instables. Les clients peuvent déterminer si c'est le cas en regardant si les deux systèmes de fichiers appartiennent à la même classe de vérificateur d'écriture. Quand c'est le cas, les vérificateurs d'écriture retournés d'un système peuvent être comparés à ceux retournés par les autres et des écritures superflues peuvent être évitées.

Quand deux systèmes de fichiers appartiennent à des classes différentes de vérificateur d'écriture, tout vérificateur généré par l'un ne doit pas être comparé à un fourni par l'autre. À la place, les deux vérificateurs devraient être traités comme non égaux même quand les valeurs sont identiques.

11.11.7 Mouchards et vérificateurs READDIR et transitions de système de fichiers

Dans une transition de système de fichiers, les deux systèmes de fichiers pourraient être cohérents dans leur traitement de mouchards et vérificateurs READDIR. Les clients peuvent déterminer si c'est le cas en regardant si les deux systèmes de fichiers appartiennent à la même classe de readdir. Quand c'est le cas, la classe de readdir, les mouchards READDIR, et les vérificateurs provenant d'un système vont être reconnus par l'autre, et les opérations READDIR commencées sur un serveur peuvent être validement continuées sur l'autre simplement en présentant au second le mouchard et le vérificateur retournés par une opération READDIR faite sur le premier système de fichiers.

Quand deux systèmes de fichiers appartiennent à des classes différentes de readdir, tout mouchard READDIR et vérificateur générés par l'un n'est pas valide sur le second et ne doit pas être présenté à ce serveur par le client. Le client devrait agir comme si le vérificateur était rejeté.

11.11.8 Données de système de fichiers et transitions de système de fichiers

Quand plusieurs répliques existent et sont utilisées simultanément ou à la suite par un client, les applications qui les utilisent vont normalement s'attendre à ce qu'elles contiennent les mêmes données ou des données cohérentes avec la sorte normale de changements qui sont faits par les autres clients mettant à jour les données du système de fichiers (avec les métadonnées qui sont les mêmes au degré indiqué par l'attribut fs_locations_info). Cependant, quand plusieurs systèmes de fichiers sont présentés comme des répliques d'un autre, la relation précise entre les données de l'un et les données d'un autre n'est en général pas spécifiée par le protocole NFSv4.1. Il est possible de présenter les répliques de systèmes de fichiers où les données de ces systèmes de fichiers sont suffisamment différentes pour que certaines applications aient des problèmes en traitant la transition entre les répliques. L'espace de noms va normalement être construit de telle façon que les applications puissent choisir un niveau approprié de prise en charge, afin que dans une position dans l'espace de noms, un ensemble varié de répliques pourrait être mentionné, tandis que dans un autre, seuls ceux qui sont à jour seraient considérés être les répliques. Le protocole définit trois cas particuliers de relations parmi les répliques à spécifier par le serveur et sur lesquelles s'appuient les clients :

* Quand plusieurs répliques existent et sont utilisées simultanément par un client (voir la définition de FSLIB4_CLSIMUL dans fs_locations_info) elles doivent désigner les mêmes données. Lorsque les systèmes de fichiers sont écrivables, un changement fait sur une instance doit être visible sur toutes les instances au même moment, sans considération de si l'instance interrogée est celle sur laquelle la modification a été faite. Cela permet à un client d'utiliser ces répliques simultanément sans aucune adaptation particulière au fait qu'il y a plusieurs répliques, au delà de s'adapter au fait que les verrous obtenus sur une réplique sont maintenus séparément (c'est-à-dire, sous un identifiant de client différent). Dans ce cas, les verrous (qu'ils soient de partage de réservation ou de gamme d'octets) et les délégations obtenues sur une réplique sont immédiatement reflétés sur toutes les répliques, en ce sens que cet accès à partir de tout autre serveur est empêché sans considération de la réplique utilisée. Cependant, parce que les serveurs ne sont pas obligés de traiter deux identifiants de client associés comme représentant le même client, il est meilleur pour accéder à chaque fichier de n'utiliser qu'un seul identifiant de client.

* Quand une réplique est désignée comme instance successeur d'une autre instance existante après le retour de NFS4ERR_MOVED (c'est-à-dire, le cas de migration) le client peut dépendre du fait que tous les changements écrits dans une mémorisation stable sur l'instance d'origine sont écrits dans une mémorisation stable du successeur (les écritures non engagées sont traitées comme au paragraphe 11.11.6).

* Lorsque un système de fichiers n'est pas écrivable mais représente une copie en lecture seule (éventuellement périodiquement mise à jour) d'un système de fichiers écrivable, les clients ont des exigences similaires à l'égard de la propagation des mises à jour. Ils peuvent avoir besoin de la garantie que tout changement visible sur l'instance de système de fichiers originale doit être immédiatement visible sur toute réplique avant que le client passe à l'accès à cette

réplique, afin d'éviter toute possibilité qu'un client, en effectuant une transition à une réplique, voit une inversion de l'état du système de fichiers. Les moyens spécifiques de cette garantie varient sur la base de la valeur du champ fss_type qui est rapporté au titre de l'attribut fs_status (voir le paragraphe 11.18). Comme ces systèmes de fichiers sont présumés ne pas convenir pour une utilisation simultanée, on ne spécifie pas comment le verrouillage est traité ; en général, les verrous obtenus dans un système de fichiers vont être séparés de ceux des autres. Comme ils sont supposés être des systèmes de fichiers en lecture seule, cela ne devrait probablement pas poser de problème aux clients ou applications.

Quand aucune de ces situations particulières ne s'applique, il n'y a pas de base dans le protocole pour que le client fasse des hypothèses sur le contenu d'une réplique de système de fichiers ou ses relations avec de précédentes instances de système de fichiers. Donc, commuter entre des systèmes de fichiers nominalement identiques en lecture-écriture ne serait pas possible parce que soit le client n'utilise pas l'attribut fs_locations_info, soit le serveur ne le prend pas en charge.

11.11.9 État verrouillé et transitions de système de fichiers

Lorsque il accède à un système de fichiers, un client obtient les verrous appliqués par le serveur, qui peuvent empêcher des actions d'autres clients qui sont non compatibles avec ces verrous.

Quand l'accès est transféré entre les répliques, les clients doivent être assurés que les actions interdites par la détention de ces verrous ne peuvent pas s'être produites durant la transition. Cela peut être assuré par les méthodes ci-dessous. Sauf si au moins une d'elles est mise en œuvre, les clients ne seront pas assurés de la continuité de la possession de verrous à travers un événement de migration :

* Fournir au client une opportunité de ré-obtenir ses verrous via une période de grâce par système de fichiers sur le serveur de destination, refusant à tous les clients qui utilisent le système de fichiers de destination l'opportunité d'obtenir de nouveaux verrous qui seraient en conflit avec ceux détenus par le client transféré tant que ce client n'a pas terminé sa période de grâce par système de fichiers. Parce que le mécanisme de réclamation de verrou a été défini à l'origine pour prendre en charge le réamorçage du serveur, il suppose implicitement que les brides de fichier vont, lors de la réclamation, être les mêmes que celles à l'ouverture. En cas de migration, cela exige que les serveurs de source et destination utilisent les mêmes brides de fichier, comme prouvé par l'utilisation de la même portée de serveur (voir le paragraphe 2.10.4) ou en montrant cet accord en utilisant fs_locations_info (voir le paragraphe 11.11.2).

  Noter qu'une telle période de grâce peut être mise en œuvre sans interférer avec la capacité des clients non transférés d'obtenir de nouveaux verrous pendant qu'elle court. Tant que le serveur de destination est au courant des verrous transférés, il peut distinguer les demandes d'obtention de nouveaux verrous qui contrastent avec les verrous existants de celles qui ne le font pas, lui permettant de traiter de telles demandes de client sans référence à la période de grâce en cours.

* L'état de verrouillage peut être transféré au titre de la transition en fournissant une migration d'état transparente comme décrit au paragraphe 11.12.

De ces deux méthodes, la migration d'état transparente fournit l'expérience la plus douce aux clients en ce qu'il n'est pas besoin de passer par un processus de réclamation avant que de nouveaux verrous puissent être obtenus ; cependant, elle exige un plus haut degré de coordination inter-serveurs. En général, les serveurs qui prennent part à une migration sont libres de fournir l'une ou l'autre facilité. Cependant, quand les brides de fichier peuvent différer à travers l'événement de migration, la migration d'état transparente est le seul moyen disponible de fournir la fonction nécessaire.

On devrait noter que ces deux méthodes ne sont pas mutuellement exclusives et qu'un serveur pourrait bien fournir les deux. En particulier, si des circonstances empêchent un verrou spécifique d'être transféré de façon transparente, le serveur de destination peut lui permettre d'être réclamé en mettant en œuvre une période de grâce par système de fichiers pour le système de fichiers migré.

11.11.9.1 Considération sur la sécurité relative à la réclamation d'état de verrouillage après transition de système de fichiers

Bien qu'il soit possible qu'un client qui réclame l'état représente mal son état de la même façon que décrit au paragraphe 8.4.2.1.1, la plupart des mises en œuvre qui fournissent de telles réclamations dans le cas de transitions de système de fichiers vont avoir la capacité de détecter de telles mauvaises représentations. Cela limite la capacité de clients non authentifiés d'exécuter des attaques de déni de service dans ces circonstances. Néanmoins, les règles déclarées au paragraphe 8.4.2.1.1 concernant la principale vérification pour les demandes de réclamation s'appliquent aussi à cette situation.

Normalement, les mises en œuvre qui prennent en charge les transitions de système de fichiers vont avoir des informations extensives sur les verrous à transférer. C'est parce que :

* Comme une défaillance n'est pas impliquée, il n'y a pas besoin de mémoriser les informations de verrouillage dans une mémorisation persistante.
* Il n'y a pas besoin, comme dans le cas de défaillance, de mettre à jour plusieurs répertoires contenant l'état de verrouillage pour les garder synchrones. À la place, il y a une unique communication d'état de verrouillage du serveur de source au serveur de destination.
* Fournir cette information évite de potentielles interférences avec les clients existants qui utilisent le système de fichiers de destination en leur refusant d'obtenir de nouveaux verrous durant la période de grâce.

Quand de telles informations de verrouillage détaillées, n'incluant pas nécessairement les identifiants d'état associés, sont disponibles :

* Il est possible de détecter les demandes de réclamation qui tentent de réclamer des verrous qui n'existaient pas avant le transfert, en les rejetant avec NFS4ERR_RECLAIM_BAD (paragraphe 15.1.9.4).
* Il est possible quand on traite des demandes non de réclamation, de déterminer si elles sont en conflit avec les verrous existants, éliminant le besoin de retourner NFS4ERR_GRACE (paragraphe 15.1.9.2) sur des demandes non de réclamation.

Il est possible pour les mises en œuvre de périodes de grâce en connexion avec les transitions de système de fichiers de n'avoir pas d'informations de verrouillage détaillées disponibles au serveur de destination, et dans ce cas, la situation de sécurité est exactement comme décrit au paragraphe 8.4.2.1.1.

11.11.9.2 Prêts et transitions de système de fichiers

Dans le cas de renouvellement de prêt, le client peut ne pas soumettre de demandes pour un système de fichiers qui a été transféré à un autre serveur. Cela peut se produire à cause du mécanisme de renouvellement de prêt. Le client renouvelle le prêt associé à tous les systèmes de fichiers quand il soumet une demande sur une session associée, sans considération du système de fichiers spécifique référencé.

Afin que le client programme le renouvellement de son prêt lorsque il y a un état de verrouillage qui peut avoir été relocalisé sur le nouveau serveur, le client doit trouver la relocalisation du prêt avant son expiration. Pour accomplir cela, l'opération SEQUENCE va retourner le bit d'état SEQ4_STATUS_LEASE_MOVED si la responsabilité d'un état de verrouillage renouvelé a été transférée à un nouveau serveur. Cela va continuer jusqu'à ce que le client reçoive une erreur NFS4ERR_MOVED pour chaque système de fichiers pour lequel il y a eu une relocalisation d'état de verrouillage.

Quand un client reçoit une indication SEQ4_STATUS_LEASE_MOVED d'un serveur, pour chaque système de fichiers du serveur pour lequel le client a un état de verrouillage, le client devrait effectuer une opération. Pour simplifier, le client peut choisir de référencer tous les systèmes de fichiers, mais ce qui est important est qu'il doit référencer tous les systèmes de fichiers pour lesquels il y avait un état de verrouillage qui a bougé. Une fois que le client reçoit une erreur NFS4ERR_MOVED pour chacun de ces systèmes de fichiers, le serveur va supprimer l'indication SEQ4_STATUS_LEASE_MOVED. Le client peut terminer le processus de vérification des systèmes de fichiers une fois que cette indication est supprimée (mais seulement si le client a reçu une réponse pour toutes les demandes SEQUENCE en instance sur toutes les sessions qu'il a avec le serveur) car il n'y en a pas d'autres pour qui l'état de verrouillage a changé.

Un client peut utiliser GETATTR de l'attribut fs_status (ou fs_locations_info) sur tous les systèmes de fichiers pour obtenir des indications d'absence en une seule demande (ou quelques unes) car les systèmes de fichiers absent ne vont pas causer une erreur dans ce contexte. Cependant, il doit quand même faire une opération qui reçoive NFS4ERR_MOVED sur chaque système de fichiers, afin d'éliminer l'indication SEQ4_STATUS_LEASE_MOVED.

Une fois que l'ensemble de systèmes de fichiers avec un état de verrouillage transféré a été déterminé, le client peut suivre le processus normal pour obtenir les nouvelles informations de serveur (par les attributs fs_locations et fs_locations_info) et effectuer le renouvellement de ce prêt sur le nouveau serveur, sauf si les informations dans l'attribut fs_locations_info montrent qu'aucun état n'a pu être transféré. Si le serveur n'a pas eu d'état transféré de façon transparente, le client va recevoir NFS4ERR_STALE_CLIENTID du nouveau serveur, comme décrit ci-dessus, et le client peut alors réclamer les verrous comme on fait en cas de défaillance du serveur.

11.11.9.3 Transitions et attribut Lease_time

Afin que le client puisse gérer de façon appropriée son prêt dans le cas d'une transition de système de fichiers, le serveur de destination doit établir les valeurs appropriées pour l'attribut lease_time.

Quand l'état est transféré de façon transparente, cet état devrait inclure la valeur correcte de l'attribut lease_time. L'attribut lease_time sur le serveur de destination ne doit jamais être moins que celui de la source, car il en résulterait une expiration prématurée d'un prêt accordé par le serveur de source. Lors de transitions dans lesquelles l'état est transféré de façon transparente, le client n'est pas obligé l'aller chercher à nouveau l'attribut lease_time et peut continuer d'utiliser la valeur qu'il est allé chercher précédemment (sur le serveur de source).

Si l'état n'a pas été transféré de façon transparente, parce que les serveurs associés se trouvent avoir des chaînes eir_server_scope différentes ou parce que l'identifiant de client est rejeté quand il est présenté au nouveau serveur, le client devrait aller chercher la valeur de lease_time sur le nouveau serveur (c'est-à-dire, de destination) et l'utiliser pour les demandes de verrouillage suivantes. Cependant, le serveur doit respecter une période de grâce de au moins autant que le lease_time sur le serveur de source, afin d'assurer que les clients ont le temps de réclamer leur verrou avant que des verrous potentiellement en conflit non réclamés soient accordés.

11.12 Transfert d'état sur migration

Quand la transition est un résultat d'une décision initiée par un serveur de faire une transition d'accès, et que les serveurs de source et destination ont mis en œuvre une coopération appropriée, il est possible de faire ce qui suit :

* Transférer l'état de verrouillage du serveur de source à celui de destination de façon similaire à celle fournie par une migration d'état transparente dans NFSv4.0, comme décrit dans la [RFC7931]. Les responsabilités de serveur sont décrites au paragraphe 11.14.2.
* Transférer l'état de session du serveur de source au serveur de destination. Les responsabilités de serveur en effectuant un tel transfert sont décrites au paragraphe 11.14.3.

Les moyens par lesquels le client détermine quels événements de transfert se sont produits sont décrits au paragraphe 11.13.

11.12.1 Migration d'état transparente et pNFS

Quand pNFS est impliqué, le protocole est capable de prendre en charge :

* La migration du serveur de métadonnées (MDS) laissant les serveurs de données (DS) en place.
* La migration du système de fichiers comme un tout, y compris le MDS et les DS associés.
* Le remplacement d'un DS par un autre.
* La migration d'un système de fichiers pNFS dans un dans lequel pNFS n'est pas utilisé.
* La migration d'un système de fichiers n'utilisant pas pNFS vers un dans lequel ces dispositions sont disponibles.

Noter que la migration, en soi, est seulement impliquée dans le transfert de la fonction MDS. Bien que le service d'une disposition puisse être transféré d'un serveur de données à un autre, cela ne fait pas utiliser les attributs de localisation de système de fichiers. Le MDS peut effectuer de tels transferts en rappelant ou révoquant les dispositions existantes et en en accordant de nouvelles sur un serveur de données différent.

La migration de la fonction de MDS est directement prise en charge par la migration d'état transparente. L'état de disposition va normalement être transféré de façon transparente, juste comme l'est un autre état. Par suite, la migration d'état transparente fournit un cadre dans lequel, dans un transfert de données inter MDS approprié, un MDS peut être substitué à un autre.

La migration de la fonction de système de fichiers comme un tout peut être accomplie en rappelant toutes les dispositions au titre de la phase initiale du processus de migration. Par suite, l'entrée/sortie va être faite à travers le MDS durant le processus de migration, et de nouvelles dispositions peuvent être accordées une fois que le client interagit avec le nouveau MDS. Un MDS peut aussi effectuer cette sorte de transition en révoquant toutes les dispositions au titre de la migration d'état transparente, pour autant que le client soit notifié de la perte de l'état de verrouillage.

Afin de permettre la migration sur un système de fichiers sur lequel pNFS n'est pas pris en charge, les clients doivent être prêts à une situation dans laquelle des dispositions ne sont pas disponibles ou prises en charge sur le système de fichiers de destination et donc dirigent les demandes d'entrée/sortie sur le serveur de destination, plutôt que de dépendre de dispositions disponibles.

Le remplacement d'un DS par un autre n'est pas traité par la migration elle-même mais peut être effectué par un MDS rappelant des dispositions pour le DS à remplacer et en produisant de nouvelles à servir par le DS suivant.

La migration peut transférer un système de fichiers d'un serveur qui ne prend pas en charge pNFS à un qui le fait. Afin de bien s'adapter à cette situation, les clients qui prennent en charge pNFS, mais fonctionnent de façon adéquate en son absence, devrait vérifier la prise en charge de pNFS quand un système de fichiers migre et être prêts à utiliser pNFS quand la prise en charge est disponible à la destination.

11.13 Responsabilités du client lors d'une transition d'accès

Pour qu'un client réponde à une transition d'accès, il doit en être informé. Les façons dont cela peut se produire sont discutées au paragraphe 11.13.1, qui discute les indications sur un chemin d'accès qu'à traversé un système de fichiers spécifique ainsi que les situations dans lesquelles une activité supplémentaire est nécessaire pour déterminer l'ensemble de systèmes de fichiers qui ont migré. Le paragraphe 11.13.2 poursuit la discussion de comment l'ensemble de systèmes de fichiers migré pourrait être déterminé. Les paragraphes 11.13.3 à 11.13.5 discutent comment le client devrait traiter chaque transition dont il a connaissance, soit directement, soit par suite de la découverte de migration.

Les termes suivants sont utilisés pour décrire les activités de client :

* "Récupération de transition" se réfère au processus de restauration de l'accès à un système de fichiers sur lequel NFS4ERR_MOVED a été reçu.

* "Récupération de migration" se réfère au sous ensemble de récupération de transition qui s'applique quand le système de fichiers a migré à une réplique différente.

* "Découverte de migration" se réfère au processus de déterminer quels systèmes de fichiers ont migré. Il est nécessaire d'éviter une situation dans laquelle les prêts pourrait expirer quand un système de fichiers n'est pas accédé pendant longtemps, car un client ignorant de la migration pourrait faire référence à un système de fichiers non migré et ne pas renouveler le prêt associé au système de fichiers migré.

11.13.1 Notifications de transition au client

Quand il y a un changement dans le chemin d'accès au réseau que un client va utiliser pour accéder à un système de fichiers, il y a un certain nombre d'indications d'état en rapport que les clients doivent traiter :

* Si une tentative est faite d'utiliser ou retourner une bride de fichier au sein d'un système de fichiers qui n'est plus accessible à l'adresse précédemment utilisée pour y accéder, l'erreur NFS4ERR_MOVED est retournée. Des exceptions sont faites pour permettre que de telles brides de fichier soient utilisées quand on interroge un attribut de localisation de système de fichiers. Cela permet à un client de déterminer la localisation d'une nouvelle réplique ou un nouveau chemin d'accès réseau. Cette condition continue sur les tentatives suivantes pour accéder au système de fichiers en question. La seule façon dont le client peut éviter l'erreur est de cesser d'accéder au système de fichiers en question à son ancienne localisation de serveur et d'y accéder à la place en utilisant l'adresse différente à laquelle il est maintenant disponible.

* Chaque fois que un client envoie une opération SEQUENCE à un serveur qui a généré l'état détenu sur ce client et associé à un système de fichiers qui n'est plus accessible sur ce serveur, la réponse va contenir le bit d'état SEQ4_STATUS_LEASE_MOVED, indiquant qu'il y a eu une migration de prêt. Cette condition se continue jusqu'à ce que le client accuse réception de la notification en allant chercher un attribut de localisation de système de fichiers pour le système de fichiers dont le chemin d'accès réseau est changé. Quand il y a plusieurs de ces systèmes de fichiers, on doit aller chercher un attribut de localisation pour chacun de ces systèmes de fichiers. On doit aller chercher l'attribut de localisation pour tous les systèmes de fichiers migrés afin de supprimer la condition. Même après la suppression de la condition, le client doit répondre en utilisant les informations de localisation pour accéder au système de fichiers à sa nouvelle localisation pour s'assurer que le prêt n'est pas expiré sans nécessité.

À la différence de NFSv4.0, dans lequel les conditions correspondantes sont les deux erreurs et donc mutuellement exclusives, dans NFSv4.1 le client peut, et souvent va, recevoir les deux indications sur la même demande. Par suite, les mises en œuvre doivent traiter la question de comment coordonner les nécessaires actions de récupération quand les deux indications arrivent dans la réponse à la même demande. On devrait noter que quand il traite un COMPOUND NFSv4, le serveur va normalement décider si SEQ4_STATUS_LEASE_MOVED est à établir avant de déterminer quel système de fichiers va être référencé ou si NFS4ERR_MOVED est à retourner.

Comme ces indications ne sont pas mutuellement exclusives dans NFSv4.1, les combinaisons suivantes sont les résultats possibles quand un COMPOUND est produit :

* L'état COMPOUND est NFS4ERR_MOVED, et SEQ4_STATUS_LEASE_MOVED est affirmé. Dans ce cas, la récupération de transition est requise. Bien qu'il soit possible que la découverte de migration est nécessaire en plus, il est probable que seul le système de fichiers accédé a transitionné. En tous cas, parce que adresser NFS4ERR_MOVED est nécessaire pour permettre de traiter les demandes rejetées sur la cible, le faire va normalement avoir la priorité sur la découverte de migration.

* L'état COMPOUND est NFS4ERR_MOVED, et SEQ4_STATUS_LEASE_MOVED est à zéro. Dans ce cas, la récupération de transition est aussi requise. Il est clair que la découverte de migration n'est pas nécessaire pour trouver les systèmes de fichiers qui ont migré, autres que celui qui retourne NFS4ERR_MOVED. Les cas dans lesquels ce résultat peut apparaître incluent un référent ou une migration pour laquelle il n'y a pas d'état de verrouillage non associé. Cela peut aussi arriver dans des cas dans lesquels une transition de chemin d'accès autre qu'une migration se produit dans le même serveur. Dans ce cas, il n'est pas besoin d'établir SEQ4_STATUS_LEASE_MOVED, car le prêt reste associé au serveur courant même si le chemin d'accès a changé.

* L'état COMPOUND n'est pas NFS4ERR_MOVED, et SEQ4_STATUS_LEASE_MOVED est établi. Dans ce cas, aucune activité de récupération de transition n'est requise sur le système de fichiers accédé par la demande. Cependant, pour prévenir une expiration de prêt évitable, la découverte de migration doit être faite.

* L'état COMPOUND n'est pas NFS4ERR_MOVED, et SEQ4_STATUS_LEASE_MOVED est à zéro. Dans ce cas, ni une activité relative à la transition ni la découverte de migration ne sont requises.

Noter que les actions spécifiées ont seulement besoin d'être effectuées si elles ne sont pas déjà en cours. Par exemple, quand NFS4ERR_MOVED est reçu lors de l'accès à un système de fichiers pour lequel la récupération de transition est déjà en cours, le client attend simplement que cette récupération s'achève, tandis que la réception de l'indication SEQ4_STATUS_LEASE_MOVED a seulement besoin d'initier la découverte de migration pour un serveur si une telle découverte n'est pas déjà en cours pour ce serveur.

Le fait qu'une condition de migration de prêt ne résulte pas en une erreur dans NFSv4.1 a un certain nombre de conséquences importantes. En plus du fait que les deux indications ne sont pas mutuellement exclusives, comme discuté ci-dessus, il y a un certain nombre de problèmes qui sont importants dans l'examen d'une mise en œuvre de découverte de migration, comme discuté au paragraphe 11.13.2.

Parce que SEQ4_STATUS_LEASE_MOVED n'est pas une condition d'erreur, il est possible aux systèmes de fichiers dont les chemins d'accès n'ont pas changé d'être accédés avec succès sur un certain serveur même si la récupération est nécessaire pour les autres systèmes de fichiers sur le même serveur. Par suite, l'accès peut avoir lieu si :
* le processus de découverte de migration se produit pour ce serveur,
* le processus de récupération de transition se produit pour les autres systèmes de fichiers connectés à ce serveur.

11.13.2 Découverte de la migration

La découverte de la migration peut être effectuée dans le même contexte que la récupération de transition, permettant la récupération pour chaque système de fichiers migré à invoquer lorsque il est découvert. Autrement, elle peut être faite dans une trame de découverte de migration séparée, permettant de faire la découverte de migration en parallèle avec une ou plusieurs instances de récupération de transition.

Dans l'un et l'autre cas, parce que l'indication de migration de prêt ne résulte pas en une erreur, les autres accès aux systèmes de fichiers sur le serveur peuvent se poursuivre normalement, avec la possibilité que d'autres indications de cette sorte soient reçues, soulevant la question de comment traiter de telles indications. En général :
* Aucune action n'est nécessaire pour de telles indications reçues par une trame effectuant la découverte de migration, car la continuation de ce travail va régler la question.
* Dans les autres cas où la découverte de migration est actuellement en cours, rien de plus n'est nécessaire pour répondre à de telles indications de migration de prêt, pour autant qu'on puisse être certain que le processus de découverte de migration va traiter ces indications. Voir les détails ci-dessous.
* Pour les indications reçues dans tout autre contexte, la réponse appropriée est d'initier ou autrement assurer l'exécution de la découverte de migration pour les systèmes de fichiers associés à l'adresse IP du serveur qui retourne l'indication.

Cela laisse une difficulté potentielle dans des situations où le processus de découverte de migration est prêt de s'achever mais est encore en cours. On ne devrait pas ignorer une indication SEQ4_STATUS_LEASE_MOVED si le processus de découverte de migration n'est pas capable de répondre à la découverte de systèmes de fichiers migrants supplémentaires

sans autre aide. Une complexité supplémentaire relevant du traitement de telles situations est qu'une indication de migration de prêt peut refléter l'état du serveur au moment du traitement de l'opération SEQUENCE, qui peut être différent de celui en vigueur au moment de réception de la réponse. Parce que de nouveaux événements de migration peuvent se produire à tout moment, et parce que une indication SEQ4_STATUS_LEASE_MOVED peut refléter la situation en vigueur un temps considérable avant la réception de l'indication, une attention particulière doit être portée à s'assurer que les indications SEQ4_STATUS_LEASE_MOVED ne sont pas ignorées à tort.

Une approche utile de cette question implique l'utilisation d'états séparés de découverte de migration visibles en externe pour chaque serveur. Des valeurs séparées pourraient représenter les divers états possibles pour le processus de découverte de migration pour un serveur :

* Non fonctionnement, dans lequel la découverte de migration n'est pas effectuée.
* Fonctionnement normal, dans lequel il y a un examen en cours des systèmes de fichiers migrés.
* Achèvement/vérification du traitement de découverte de migration, dans lequel l'achèvement possible du processus de découverte de migration doit être vérifié.

Dans ce cadre, le traitement de découverte de migration se ferait comme suit :

* Dans l'état de fonctionnement normal, la trame qui effectue la découverte va aller chercher, pour les systèmes de fichiers successifs connus du client sur le serveur sur lequel il travaille, un attribut de localisation de système de fichiers plus l'attribut fs_status.

* Si l'attribut fs_status indique que le système de fichiers a migré (c'est-à-dire, fss_absent est vrai, et fss_type != STATUS4_REFERRAL) alors un système de fichiers migré a été trouvé. Dans cette situation, il est probable qu'aller chercher l'attribut de localisation du système de fichiers a supprimé un des systèmes de fichiers qui ont contribué à l'indication du prêt migré.

* Dans les cas où cela est arrivé, la trame ne peut pas savoir si l'indication de prêt migré a été supprimée, et donc elle entre dans l'état achèvement/vérification et procède à produire un COMPOUND pour voir si l'indication SEQ4_STATUS_LEASE_MOVED a été supprimée.

* Quand le processus de découverte est dans l'état achèvement/vérification, si d'autres demandes obtiennent une indication de prêt migré, il note qu'elles ont été reçues. Plus tard, l'existence de telles indications est utilisée quand la demande s'achève, comme décrit ci-dessous.

Quand la demande utilisée dans l'état achèvement/vérification s'achève :

* Si une indication de prêt migré est retournée, la découverte continue normalement. Noter qu'il en est ainsi même si tous les systèmes de fichiers ont été traversés, car de nouvelles migrations pourraient être intervenues pendant que le processus était en cours.

* Autrement, si il y a des enregistrements que d'autres demandes ont vu une indication de prêt migré alors que la demande se produisait, cet enregistrement est supprimé, et la demande de vérification est réessayée. Le processus de découverte reste dans l'état achèvement/vérification.

* Si il n'y a pas eu d'indication de prêt migré, le travail de découverte de migration est considéré comme achevé, et il entre dans l'état non fonctionnement. Une fois entré dans cet état, les indications de prêt migré suivantes vont déclencher un nouveau processus de découverte de migration.

On devrait noter qu'il n'est pas garanti que le processus décrit ci-dessus se termine, car une longue série de nouveaux événements de migration pourrait continuellement retarder la suppression de l' indication de SEQ4_STATUS_LEASE_MOVED. Pour prévenir une expiration de prêt non nécessaire, il est approprié que les clients utilisent la découverte de migration pour effectuer immédiatement le renouvellement de prêt, plutôt que d'attendre la suppression de l'indication de SEQ4_STATUS_LEASE_MOVED quand l'ensemble complet de migrations est disponible.

La découverte de prêt doit être fournie comme décrit ci-dessus. Cela assure que le client découvre les migrations de système de fichier assez tôt pour renouveler ses prêts sur chaque serveur de destination avant qu'ils expirent. Les non renouvellement de prêts peut conduire à la perte de l'état de verrouillage. Bien que les conséquences d'une telle perte puissent être améliorées par la mise en œuvre de verrous de courtoisie, les serveurs ne sont pas obligés de le faire, et une demande de verrou en conflit peut signifier qu'un verrou soit révoqué de façon inattendue. Les clients devraient être conscients de cette possibilité.

11.13.3 Généralités sur la réponse du client à NFS4ERR_MOVED

Ce paragraphe décrit un moyen par lequel un client qui reçoit NFS4ERR_MOVED peut effectuer la récupération de transition en utilisant un nouveau serveur ou point d'extrémité de serveur si un est disponible. Au titre de ce processus, il va déterminer :

* Si le NFS4ERR_MOVED indique qu'une migration s'est produite, ou si il indique une autre sorte de transition d'accès de système de fichiers comme discuté au paragraphe 11.10.
* Dans le cas de migration, si une migration d'état transparente s'est produite.
* Si un état a été perdu durant le processus de migration d'état transparente.
* Si des sessions ont été transférées au titre de la migration d'état transparente.

Durant la première phase de ce processus, le client procède à l'examen des entrées de localisation de système de fichiers pour trouver l'adresse réseau initiale qu'il va utiliser pour continuer d'accéder au système de fichiers ou pour son remplacement. Pour chaque entrée de localisation que le client examine, le processus consiste en cinq étapes :

1. Effectuer un EXCHANGE_ID dirigé sur l'adresse de localisation. Cette opération est utilisée pour enregistrer le propriétaire du client (sous la forme d'un client_owner4) auprès du serveur, pour obtenir un identifiant de client à utiliser ensuite pour communiquer avec lui, pour obtenir l'état de confirmation de cet identifiant de client, et pour déterminer le server_owner4 et sa portée pour les besoins de la détermination de si l'entrée est multiplexable avec l'adresse utilisée précédemment pour accéder au système de fichiers (c'est-à-dire, qu'elle représente un autre chemin d'accès réseau pour le même système de fichiers et peut partager l'état de verrouillage avec lui).

2. Faire une détermination initiale de si une migration s'est produite. La détermination initiale va se fonder sur si le résultat de EXCHANGE_ID indique que l'élément de localisation courant est multiplexable par le serveur avec celui utilisé pour accéder au système de fichiers quand l'accès s'est terminé par la réception de NFS4ERR_MOVED. Si il l'est, alors une migration ne s'est pas produite. Dans ce cas, la transition est traitée, au moins initialement, comme une qui implique un accès continu au même système de fichiers sur le même serveur à travers une nouvelle adresse réseau.

3. Obtenir l'accès à l'état d'une session existante ou créer de nouvelles sessions. Comment c'est fait dépend de la détermination initiale de si une migration s'est produite et peut être fait comme décrit au paragraphe 11.13.4 dans le cas de migration ou comme décrit au paragraphe 11.13.5 dans le cas d'un transfert d'adresse réseau sans migration.

4. Vérifier la relation de multiplexage supposée dans l'étape 2 comme discuté au paragraphe 2.10.5.1. Bien que cette étape confirme généralement la détermination initiale, il est possible que la vérification invalide la détermination initiale de glissement d'adresse réseau (sans migration) et détermine à la place qu'une migration s'est produite. Il n'est pas besoin de refaire l'étape 3, car il va être possible de continue d'utiliser la session déjà établie.

5. Obtenir l'accès à l'état de verrouillage existant et/ou le ré-obtenir. Comment ceci est fait dépend de la détermination finale de si la migration s'est produite et peut être fait comme décrit au paragraphe 11.13.4 dans le cas de migration ou comme décrit au paragraphe 11.13.5 dans le cas d'un transfert d'adresse réseau sans migration.

Une fois que l'adresse initiale a été déterminée, les clients sont libres d'appliquer un processus abrégé pour trouver des adresses multiplexables supplémentaires avec elle (les clients peuvent chercher des adresses multiplexables par la session ou multiplexables par le serveur selon si elles prennent en charge le multiplexage d'identifiant de client). Durant cette dernière phase du processus, les autres entrées de localisation sont examinées en utilisant la procédure abrégée spécifiée ci-dessous :

A : Avant le EXCHANGE_ID, le nom de système de fichiers de l'entrée de localisation est examiné, et si il ne correspond pas à celui actuellement utilisé, l'entrée est ignorée. Autrement, on procède comme spécifié à l'étape 1 ci-dessus.

B : Dans le cas où l'adresse réseau est multiplexable par la session avec une utilisée précédemment, un BIND_CONN_TO_SESSION est utilisé pour accéder à cette session en utilisant la nouvelle adresse réseau. Autrement, ou si l'opération bind échoue, un CREATE_SESSION est fait.

C : La procédure de vérification mentionnée à l'étape 4 ci-dessus est utilisée. Cependant, si elle échoue, l'entrée est ignorée et la prochaine entrée disponible est utilisée.

11.13.4 Obtention de l'accès aux sessions et état après migration

Dans le cas où une migration s'est produite, la récupération de migration va impliquer de déterminer si une migration d'état transparente s'est produite. Cette décision est prise sur la base de l'identifiant de client retourné par le EXCHANGE_ID et

l'état de confirmation rapporté.

* Si l'identifiant de client est un identifiant de client non confirmé inconnu précédemment du client, alors une migration d'état transparente ne s'est pas produite.

* Si l'identifiant de client est un identifiant de client confirmé précédemment connu du client, alors tout état transféré va avoir été fusionné avec un identifiant de client existant représentant le client au serveur de destination. Dans ce cas de fusion d'état, la migration d'état transparente pourrait ou non s'être produite, et la détermination de si elle s'est produite est différée jusqu'à ce que les sessions soient établies et que le client soit prêt à commencer la récupération d'état.

* Si l'identifiant de client est un identifiant de client confirmé non précédemment connu du client, alors le client peut conclure que l'identifiant de client a été transféré au titre d'une migration d'état transparente. Dans ce cas d'identifiant de client transféré, une migration d'état transparente s'est produite, bien que de l'état pourrait avoir été perdu.

Une fois que l'identifiant de client a été obtenu, il est nécessaire d'obtenir l'accès aux sessions pour continuer la communication avec le nouveau serveur. Dans tous les cas où la migration d'état transparente s'est produite, il est possible qu'une session soit aussi transférée. Pour traiter cette possibilité, les clients peuvent, après avoir fait le EXCHANGE_ID, produire un BIND_CONN_TO_SESSION pour se connecter à la session transférée d'une connexion au nouveau serveur. Si cela échoue, c'est une indication que la session n'a pas été transférée et qu'une nouvelle session doit être créée pour prendre sa place.

Dans certaines situations, il est possible qu'un BIND_CONN_TO_SESSION réussisse sans qu'une migration de session se soit produite. Si une fusion d'état a eu lieu, alors l'identifiant de client associé peut avoir déjà eu un ensemble de sessions existantes, avec lesquelles il serait possible que l'identifiant de session d'une session donnée soit le même que celui d'une qui pourrait avoir migré. Dans ce cas, un BIND_CONN_TO_SESSION pourrait réussir, même si il pourrait ne pas y avoir eu de migration de la session avec cet identifiant de session. Dans ce cas, le client va recevoir des erreurs de séquence quand les valeurs de séquence de créneau utilisées ne sont pas appropriées sur la nouvelle session. Quand cela se produit, le client peut créer une nouvelle session et cesser d'utiliser celle existante.

Une fois que le client a déterminé l'état initial de migration, et déterminé qu'il y a eu un glissement à un nouveau serveur, il doit rétablir son état de verrouillage, si possible. Pour permettre que cela se produise sans perte des garanties normalement fournies par le verrouillage, le serveur de destination doit mettre en œuvre une période de grâce par système de fichiers dans tous les cas dans lesquels l'état de verrouillage a été perdu, incluant ceux dans lesquels une migration d'état transparente n'était pas mise en œuvre. Chaque client pour lequel il y a eu un transfert d'état de verrouillage au nouveau serveur va avoir la durée de la période de grâce pour réclamer ses verrous, à partir du moment où ses verrous ont été transférés.

Les clients doivent traiter les cas suivants :

* Dans le cas de fusion d'état, il est possible que le serveur n'ait pas tenté de migration d'état transparente, et que l'état puisse avoir été perdu sans que ce soit reflété dans les bits SEQ4_STATUS. Pour déterminer si cela s'est produit, le client peut utiliser TEST_STATEID pour vérifier si les identifiants d'état créés sur le serveur de source sont toujours accessibles sur le serveur de destination. Une fois que un seul identifiant d'état est trouvé avoir été transféré avec succès, le client peut conclure que la migration d'état transparente a été commencée, et toute défaillance de transport de tous les identifiants d'état va être reflétée dans les bits SEQ4_STATUS. Autrement, la migration d'état transparente ne s'est pas produite.

* Dans le cas où une migration d'état transparente ne s'est pas produite, le client peut utiliser la période de grâce par système de fichiers fournie par le serveur de destination pour réclamer les verrous qui étaient détenus sur le serveur de source.

* Dans le cas où une migration d'état transparente s'est produite, et où aucun état de verrouillage n'est perdu (comme le montrent les fanions SEQ4_STATUS) aucune réclamation de verrou n'est nécessaire.

* Dans le cas où une migration d'état transparente s'est produite, et de l'état de verrouillage a été perdu (comme montré par les fanions SEQ4_STATUS) la validité des identifiants d'état existants doit être vérifiée en utilisant TEST_STATEID, et la réclamation utilisée pour rétablir toux ceux qui n'ont pas été transférés.

Pour tous les cas ci-dessus, un RECLAIM_COMPLETE avec une valeur de rca_one_fs de VRAI doit être fait avant l'utilisation normale du système de fichiers, incluant d'obtenir de nouveaux verrous pour le système de fichiers. Cela s'applique même si aucun verrou n'a été perdu et si aucun ne doit être réclamé.

11.13.5 Obtention de l'accès aux sessions et état après le transfert d'adresse réseau

Le cas où il y a un transfert à une nouvelle adresse réseau sans migration est similaire à celui décrit au paragraphe 11.13.4 en ce qu'il est nécessaire d'obtenir l'accès aux sessions et états de verrouillage nécessaires. Cependant, les détails sont plus simples et vont varier selon le type de multiplexage entre l'adresse recevant NFS4ERR_MOVED et celle à laquelle le transfert est à faire.

Pour rendre une session disponible, un BIND_CONN_TO_SESSION devrait être utilisé pour obtenir l'accès à la session utilisée précédemment. Seulement si cela échoue, un CREATE_SESSION devrait être fait. Bien que cette procédure reflète celle du paragraphe 11.13.4, il y a une importante différence en ce que la préservation de la session n'est pas purement facultative mais dépend du type de multiplexage.

L'accès à l'état de verrouillage approprié ne va généralement pas nécessiter d'autres actions au delà de l'accès à la session. Cependant, les bits SEQ4_STATUS doivent être vérifiés pour un état de verrouillage perdu, incluant le besoin de réclamer des verrous après un réamorçage de serveur, car il y a toujours la possibilité d'une perte d'état de verrouillage.

11.14 Responsabilités du serveur lors d'une migration

Dans le cas d'une migration de système de fichiers, quand le client se connecte au serveur de destination, ce serveur doit être capable de fournir au client un accès continu aux fichiers qu'il a ouvert sur le serveur de source. Il y a deux façons de fournir cela :

* En provisionnant une période de grâce spécifique du système de fichiers, permettant au client de réclamer ses verrous, de façon similaire à ce qui aurait été fait dans le cas de récupération d'un redémarrage de serveur. Voir au paragraphe 11.14.1 une discussion plus complète.

* En mettant en œuvre une migration d'état transparente éventuellement en connexion avec une migration de session, le serveur peut fournir au client l'accès immédiat à l'état incorporé dans le serveur de source sur le serveur de destination.

Ces caractéristiques sont discutées séparément aux paragraphes 11.14.2 et 11.14.3, qui discutent respectivement de la migration d'état transparente et de la migration de session.

Toutes les caractéristiques décrites ci-dessus peuvent impliquer un transfert d'informations relatives aux verrous entre les serveurs de source et destination. Dans certains cas, ce transfert est une partie nécessaire de la mise en œuvre, tandis que dans d'autres cas, c'est une aide à la mise en œuvre, que les serveurs pourraient utiliser ou non. Les paragraphes qui suivent discutent des informations qui seraient transférées mais ne définissent pas les spécificités du protocole de transfert. Ceci est laissé au choix de la mise en œuvre, bien que des normes pourraient être développées ultérieurement dans ce domaine.

11.14.1 Responsabilités du serveur pour effectuer la réclamation d'état après migration

Dans ce cas, le serveur de destination n'a pas besoin de connaître les verrous détenus sur le serveur de source. Il s'appuie sur les clients pour rapporter précisément (via les opérations de réclamation) les verrous précédemment détenus, et ne permet pas que de nouveaux verrous soient accordés sur des systèmes de fichiers migrés avant l'expiration de la période de grâce. Interdire de nouveaux verrous s'applique à tous les clients qui accèdent à ces systèmes de fichiers, tandis que l'expiration de la période de grâce se produit indépendamment pour chaque client migré.

Durant cette période de grâce, les clients ont l'opportunité d'utiliser les opérations de réclamation pour obtenir des verrous pour les objets de système de fichiers au sein du système de fichiers migré, de la même façon que quand ils récupèrent d'un redémarrage de serveur, et les serveurs s'appuient normalement sur les clients pour rapporter précisément leurs verrous, bien qu'ils aient l'option de soumettre ces demandes à une vérification. Si les clients réclament seulement les verrous détenus sur le serveur de source, aucun conflit ne peut survenir. Une fois que le client a réclamé ses verrous, il indique l'achèvement de la réclamation de verrous en effectuant un RECLAIM_COMPLETE spécifiant rca_one_fs comme VRAI.

Bien qu'il ne soit pas nécessaire que les serveurs de source et destination coopèrent au transfert d'informations sur les verrous, les mises en œuvre seront bien avisées de considérer le transfert des informations utiles suivantes :

* Si les informations sur l'ensemble de clients qui ont l'état de verrouillage pour le système de fichiers transféré sont rendues disponibles, le serveur de destination va être capable de terminer la période de grâce une fois que tous ces

clients ont réclamé leurs verrous, permettant que reprenne l'activité normale de verrouillage plus tôt qu'elle ne l'aurait fait autrement.

* Le verrouillage des informations sommaires pour les clients individuels (à divers niveaux de détail possibles) peut détecter des instances dans lesquelles les clients ne représentent pas avec précision les verrous détenus sur le serveur de source.

11.14.2 Responsabilités du serveur dans la migration d'état transparente

La responsabilité de base du serveur de source pour effectuer la migration d'état transparente est de rendre disponible au serveur de destination une description de chaque pièce d'état de verrouillage associée au système de fichiers qui migre. En plus de la chaîne d'identifiant de client et du vérificateur, le serveur de source doit fournir pour chaque identifiant d'état :
* L'identifiant d'état incluant la valeur de séquence courante.
* L'identifiant de client associé.
* La bride du fichier associé.
* Le type du verrou, tel que ouvert, verrou de gamme d'octets, délégation, ou disposition.
* Pour les verrous comme d'ouverture et verrous de gamme d'octets, il va y avoir des informations sur le ou les propriétaires du verrou.
* Pour les types de verrou rappelable/révocable, l'état de rappel courant doit être inclus.
* Pour chaque type de verrou, il va y avoir des informations associées spécifiques du type. Pour les ouvertures, cela va inclure les modes de partage et de refus tandis que pour les verrous de gamme d'octets et de disposition, cela va être un type et une gamme d'octets.

De telles informations vont très probablement être organisées par chaîne d'identifiant de client sur le serveur de destination afin qu'elles puissent être utilisées pour fournir un contexte approprié à chaque client quand il se fait connaître au client. Les questions en rapport avec l'usurpation d'identité d'un client par un autre qui présente la chaîne d'identifiant de client d'un autre client peuvent être traitées en utilisant les caractéristiques de protection d'état NFSv4.1, comme décrit à la Section 21.

Une autre responsabilité du serveur concerne les verrous qui sont révoqués ou autrement perdus durant le processus de migration de système de fichiers. Parce que les verrous qui paraissent être perdus durant le processus de migration vont être réclamés par le client, les serveurs doivent prendre des mesures pour s'assurer que les verrous révoqués peu avant ou peu après la migration ne sont pas par inadvertance admis à être réclamés dans des situations dans lesquelles la continuité de la possession de verrou ne peut pas être assurée.

* Pour les verrous perdus sur la source mais dont la perte n'a pas encore été reconnue par le client (en utilisant FREE_STATEID) la destination doit être informée de cette perte afin de pouvoir refuser une demande les réclamant.

* Pour les verrous perdus sur la destination après le transfert d'état mais avant que soit fait le RECLAIM_COMPLETE du client, le serveur de destination devrait les noter et ne pas leur permettre d'être réclamés.

Une responsabilité supplémentaire de la coopération des serveurs concerne les situations dans lesquelles un identifiant d'état ne peut pas être transféré de façon transparente parce que il est en conflit avec un identifiant d'état existant détenu par le client et associé à un système de fichiers différent. Dans ce cas, deux solutions sont valides :

* Traiter le transfert, comme dans NFSv4.0, comme sans migration d'état transparente. Dans ce cas, des verrous en conflit ne peuvent pas être accordés jusqu'à ce que le client ait fait un RECLAIM_COMPLETE, après avoir réclamé les verrous qu'il avait, à l'exception des réclamations refusées parce que ce sont des tentatives de réclamer des verrous qui étaient perdus.

* Mettre en œuvre une migration d'état transparente, sauf pour le verrou avec l'identifiant d'état en conflit. Dans ce cas, le client va savoir que le verrou est perdu (par les fanions SEQ4_STATUS) et il va lui être permis de le réclamer.

Quand on transfère l'état entre la source et la destination, les questions discutées au paragraphe 7.2 de la [RFC7931] doivent quand même être réglées. Dans ce cas, l'utilisation de NFS4ERR_DELAY peut encore être nécessaire dans NFSv4.1, comme elle l'était dans NFSv4.0, pour empêcher le changement d'état de verrouillage en cours de transfert. Voir au paragraphe 15.1.1.3 des informations sur les approches appropriées de réessais par le client dans le cas où NFS4ERR_DELAY est retourné.

Il y a un certain nombre d'importantes différences dans le contexte NFS4.1 :

* L'absence de RELEASE_LOCKOWNER signifie que le cas dans lequel une opération pourrait n'être pas différée par l'utilisation de NFS4ERR_DELAY n'existe plus.
* Le séquençage des opérations n'est plus fait en utilisant des numéros de séquence d'opération fondés sur le propriétaire. À la place, le séquençage est fondé sur la session.

Par suite, quand les sessions ne sont pas transférées, les techniques discutées au paragraphe 7.2 de la [RFC7931] sont adéquates et ne seront pas discutées plus avant.

11.14.3 Responsabilités du serveur pour effectuer un transfert de session

La responsabilité de base du serveur de source pour effectuer un transfert de session est de rendre disponible au serveur de destination une description de l'état en cours de chaque créneau avec la session, incluant ce qui suit :
* La dernière valeur de séquence reçue pour ce créneau.
* Si il y a des données de réponse en antémémoire pour la dernière demande exécutée et, si oui, la réponse en antémémoire.

Quand les sessions sont transférées, il y a un certain nombre de problèmes qui lancent un défi en termes de rendre l'état transféré non modifiable durant la période où il est collecté et transféré au serveur de destination :
* Une seule session peut être utilisée pour accéder à plusieurs systèmes de fichiers, dont tous ne sont pas transférés.
* Des demandes faites sur une session peuvent, même si elles sont rejetées, affecter l'état de la session en avançant le numéro de séquence associé au créneau utilisé.

Par suite, quand l'état du système de fichiers pourrait autrement être considéré comme non modifiable, le client pourrait avoir un nombre quelconque de demandes en cours, dont chacune est capable de changer l'état de session, qui peut être d'un certain nombre de types :
1. Les demandes qui ont été traitées sur le système de fichiers migrant avant le début de la migration.
2. Les demandes qui ont reçu l'erreur NFS4ERR_DELAY parce que le système de fichiers accédé était en cours de migration.
3. Les demandes qui ont reçu l'erreur NFS4ERR_MOVED parce que le système de fichiers accédé avait migré.
4 Les demandes qui ont accédé au système de fichiers migrant afin d'obtenir des informations de localisation ou d'état.
5. Les demandes qui ne faisaient pas référence au système de fichiers migrant.

On devrait noter que l'historique de tout créneau particulier va probablement inclure un certain nombre de ces classes de demande. Dans le cas où une session migrée est utilisée par des systèmes de fichiers autres que celui migré, les demandes de classe 5 peuvent être communes et peuvent être la dernière demande traitée pour de nombreux créneaux.

Comme l'état de session peut changer même après que l'état de verrouillage a été fixé au titre du processus de migration, l'état de session connu du client pourrait être différent de celui sur le serveur de destination, qui reflète nécessairement l'état de session sur le serveur de source à un instant antérieur. Pour décider comment traiter cette situation, il est utile de distinguer deux sortes de conséquences comportementales du choix des valeurs initiales d'identifiant de séquence :

* L'erreur NFS4ERR_SEQ_MISORDERED est retournée quand l'identifiant de séquence dans une demande n'est ni égal au dernier vu sur le créneau courant ni au prochain supérieur. Vue la difficulté d'arriver à une valeur mutuellement acceptable pour la valeur correcte de dernier numéro de séquence au point de migration, il peut être nécessaire que le serveur montre un certain degré d'indulgence quand l'identifiant de séquence est un qui serait considéré inacceptable si la migration de session n'était pas impliquée.

* Retourner la réponse mise en antémémoire pour une demande précédemment exécutée quand l'identifiant de séquence dans la demande correspond à la dernière valeur enregistrée pour le créneau. Dans les cas où une erreur est retournée et qu'il n'y a pas de possibilité que des opérations non idempotentes aient été exécutées, il peut n'être pas nécessaire d'adhérer à cela aussi strictement qu'il pourrait être approprié si la migration de session n'était pas impliquée. Par exemple, le fait que l'erreur NFS4ERR_DELAY soit retournée peut ne pas aider le client en pratique, alors que le fait que NFS4ERR_MOVED soit retourné par le serveur de source peut n'être pas pertinent quand la demande a été produite à nouveau et dirigée sur le serveur de destination.

Un point important est que la spécification doit prendre note de tous les COMPOUND potentiels, même si ils pourraient être peu probables en pratique. Par exemple, un COMPOUND est permis pour accéder à plusieurs systèmes de fichiers et pourrait effectuer des opérations non idempotentes dans certains d'entre eux avant d'accéder à un système de fichiers en train de migrer. Aussi, un COMPOUND peut retourner une quantité considérable de données dans la réponse avant d'être rejeté avec un NFS4ERR_DELAY ou NFS4ERR_MOVED, et peut de plus être marqué comme sa_cachethis. Cependant,

on note que si le client et le serveur respectent les règles du paragraphe 15.1.1.3, il n'est pas possible que des opérations non idempotentes soient réinjectées de façon parasitaire après avoir reçu une réponse NFS4ERR_DELAY.

Pour traiter ces questions, un serveur de destination PEUT faire tout ce qui suit quand il met en œuvre le transfert de session :

* Éviter d'appliquer une sémantique de séquençage pour un créneau particulier jusqu'à ce que le client ait établi la séquence de démarrage pour ce créneau sur le serveur de destination.

* Pour chaque créneau, éviter de retourner une réponse en antémémoire retournant NFS4ERR_DELAY ou NFS4ERR_MOVED jusqu'à ce que le client ait établi la séquence de démarrage pour ce créneau sur le serveur de destination.

* Jusqu'à ce que le client ait établi la séquence de démarrage pour un créneau particulier sur le serveur de destination, éviter de rapporter NFS4ERR_SEQ_MISORDERED ou de retourner une réponse en antémémoire qui contient NFS4ERR_DELAY ou NFS4ERR_MOVED et consiste seulement en une série d'opérations où la réponse est NFS4_OK jusqu'à l'erreur finale.

À cause des considérations mentionnées ci-dessus, y compris les règles pour le traitement de NFS4ERR_DELAY incluses au paragraphe 15.1.1.3, le serveur de destination peut répondre de façon appropriée aux opérations SEQUENCE reçues du client en adoptant les trois politiques mentionnées ci-dessous :

* Ne pas répondre avec NFS4ERR_SEQ_MISORDERED pour la demande initiale sur un créneau dans une session transférée parce que le serveur de destination ne peut pas avoir connaissance de demandes faites par le client après le transfert du serveur mais avant que le client ait connaissance du glissement. Dans les cas où NFS4ERR_SEQ_MISORDERED aurait normalement été rapporté, la demande est à traiter normalement comme une nouvelle demande.

* Répondre comme pour un nouvel essai chaque fois que le numéro de séquence correspond à celui transféré par le serveur de source, même si cela ne fournit pas de traitement de réessai pour les demandes produites après le transfert du serveur, dans l'hypothèse que, quand de telles demandes sont produites, il ne va jamais leur être répondu d'une façon qui change l'état, rendant inutile la prise en charge du ré-essai.

* Une fois qu'un SEQUENCE non de ré-essai est reçu pour un certain créneau, en utilisant cela comme base pour une vérification plus approfondie du numéro de séquence, sans autre référence à la valeur de séquence transférée par le serveur de source.

11.15 Réalisation des références de système de fichiers

Des références sont effectuées quand un système de fichiers absent est rencontré et que une ou plusieurs localisations de remplacement sont rendues disponibles par les attributs fs_locations ou fs_locations_info. Le client va normalement obtenir une erreur NFS4ERR_MOVED, aller chercher les informations de localisation appropriées, et procéder à l'accession au système de fichiers sur un serveur différent, même si il conserve sa position logique au sein de l'espace de noms original. Les références diffèrent des événements de migration en ce qu'elles arrivent seulement quand le client n'a pas précédemment référencé le système de fichiers en question (donc il n'y a pas lieu à transition). Les références peuvent seulement entrer en vigueur quand un système de fichiers absent est rencontré à sa racine.

Les exemples donnés dans les paragraphes qui suivent sont un peu artificiels en ce qu'un client réel ne va normalement pas faire une recherche multi-composants, mais va avoir des informations en antémémoire concernant les niveaux supérieurs de la hiérarchie de noms. Cependant, ces exemples sont choisis pour rendre clair le comportement requis et facile à mettre dans la portée d'un petit nombre de demandes, sans rentrer dans une discussion des détails de comment des clients spécifiques pourraient choisir de mettre des choses en antémémoire.

11.15.1 Exemple de référence (LOOKUP)

Supposons que le COMPOUND suivant soit envoyé dans un environnement dans lequel /this/is/the/path est absent du serveur cible. Cela peut être pour un certain nombre de raisons. Il se peut que le système de fichiers ait bougé ou il se peut que le serveur cible fonctionne principalement, ou seulement, pour référer des clients aux serveurs sur lesquels divers systèmes de fichiers sont localisés.

* PUTROOTFH
* LOOKUP "this"
* LOOKUP "is"
* LOOKUP "the"
* LOOKUP "path"
* GETFH
* GETATTR (fsid, fileid, size, time_modify)

Dans ces circonstances, le résultat va être :
* PUTROOTFH --> NFS_OK. La bride de fichier *(fh)* courante est maintenant la racine du pseudo-fs.
* LOOKUP "this" --> NFS_OK. La fh courante est pour /this et est dans le pseudo-fs.
* LOOKUP "is" --> NFS_OK. La fh courante est pour /this/is et est dans le pseudo-fs.
* LOOKUP "the" --> NFS_OK. La fh courante est pour /this/is/the et est dans le pseudo-fs.
* LOOKUP "path" --> NFS_OK. La fh courante est pour /this/is/the/path et est dans un nouveau système de fichiers, absent, mais ... le client ne va jamais voir la valeur de cette bride de fichier.
* GETFH --> NFS4ERR_MOVED. Échoue parce que la fh courante est dans un système de fichiers absent au démarrage de l'opération, et la spécification ne fait pas d'exception pour GETFH.
* GETATTR (fsid, fileid, size, time_modify). Non exécutée parce que la défaillance de GETFH stoppe le traitement du COMPOUND.

Étant donnée la défaillance du GETFH, le client a pour tâche de déterminer la racine du système de fichiers absent et où trouver ce système de fichiers, c'est-à-dire, le serveur et le chemin relatif à la bride de fichier racine de ce serveur. Noter que dans cet exemple, le client n'a pas obtenu les brides de fichier et les informations d'attribut (par exemple, fsid) pour les répertoires intermédiaires, de sorte qu'il ne serait pas sûr de l'endroit où commence le système de fichiers absent. Ce pourrait être le cas, par exemple, que /this/is/the soit la racine du système de fichiers déplacé et que la raison por laquelle le verrouillage de "path" a réussi est que le système de fichiers n'était pas absent sur cette opération mais a été déplacé entre le dernier LOOKUP et le GETFH (car COMPOUND n'est pas atomique). Même si on avait les identifiants de système de fichiers pour tous les répertoire intermédiaires, on pourrait n'avoir aucun moyen de savoir que /this/is/the/path était la racine d'un nouveau système de fichiers, parce que on n'a pas encore son identifiant de système de fichiers.

Afin d'obtenir les informations nécessaires, on renvoie la chaîne de LOOKUP avec des GETFH et des GETATTR pour au moins obtenir les identifiants de système de fichiers afin d'être sûrs de l'endroit où sont les frontières appropriées de système de fichiers. Le client pourrait choisir d'obtenir les fs_locations_info en même temps mais dans la plupart de cas le client va avoir bien deviné où sont les frontières de système de fichiers (parce que des NFS4ERR_MOVED ont été ou non, reçues) rendant inutile d'aller chercher fs_locations_info.

OP01 : PUTROOTFH --> NFS_OK * la bride de fichier courante est la racine du pseudo-fs.

OP02 : GETATTR(fsid) --> NFS_OK
* Juste pour être complet. Normalement, les clients connaissent le fsid du pseudo-fs aussitôt qu'ils ont établi la communication avec un serveur.

OP03 : LOOKUP "this" --> NFS_OK

OP04 : GETATTR(fsid) --> NFS_OK
* Obtient le fsid courant pour voir où sont les frontières du système de fichiers. Le fsid va être celui pour le pseudo-fs dans cet exemple, donc non frontière.

OP05 : GETFH --> NFS_OK * La bride de fichier courante est pour /this et est dans le pseudo-fs.

OP06 : LOOKUP "is" --> NFS_OK * La bride de fichier courante est pour /this/is et est dans le pseudo-fs.

OP07 : GETATTR(fsid) --> NFS_OK
* Obtient le fsid courant pour voir où sont les frontières du système de fichiers. Le fsid va être celui pour le pseudo-fs dans cet exemple, donc non frontière.

OP08 : GETFH --> NFS_OK * La bride de fichier courante est pour /this/is et est dans le pseudo-fs.

OP09 : LOOKUP "the" --> NFS_OK * La bride de fichier courante est pour /this/is/the et est dans le pseudo-fs.

OP10 : GETATTR(fsid) --> NFS_OK

* Obtient le fsid courant pour voir où sont les frontières du système de fichiers. Le fsid va être celui pour le pseudo-fs dans cet exemple, donc non frontière.

OP11 : GETFH --> NFS_OK * La bride de fichier courante est pour /this/is/the et est dans le pseudo-fs.

OP12 : LOOKUP "path" --> NFS_OK
* Obtient le fsid courant pour voir où sont les frontières du système de fichiers. Le fsid va être celui pour le pseudo-fs dans cet exemple, donc non frontière.

OP13 : GETATTR(fsid, fs_locations_info) --> NFS_OK
* On obtient le fsid pour savoir où sont les frontières du système de fichiers. Dans cette opération, le fsid va être différent de celui du répertoire parent (qui à son tour était restitué dans OP10). Noter que le fsid qui nous est donné ne va pas nécessairement être préservé à la nouvelle localisation. Ce fsid pourrait être différent, et en fait le fsid qu'on a pour ce système de fichiers pourrait être un fsid valide d'un système de fichiers différent sur ce nouveau serveur.
* Dans ce cas particulier, on est assez sûr de toutes façons que ce qui a bougé est /this/is/the/path plutôt que /this/is/the car on a le fsid du dernier et il est celui du pseudo-fs, qui ne peut probablement pas bouger. Cependant, dans d'autres exemples, on pourrait ne pas avoir ce type d'informations (par exemple, /this/is/the pourrait être un non pseudo système de fichiers séparé de /this/is/the/path) de sorte qu'on a besoin d'avoir une autre source d'informations fiable sur les frontières du système de fichiers qui est déplacé. Si, par exemple, le système de fichiers /this/is avait bougé, on aurait un cas de migration plutôt que de référence, et une fois les frontières du système de fichiers migré établies clairement, on pourrait aller chercher fs_locations_info.
* On va chercher fs_locations_info parce que le fait qu'on ait un NFS4ERR_MOVED à ce point signifie qu'il est très probable que c'est une référence et on a besoin de la destination. Même si il se trouve que /this/is/the est un système de fichiers qui a migré, on va quand même avoir besoin des informations de localisation pour ce système de fichiers.

OP14 : GETFH --> NFS4ERR_MOVED
* Échoue parce que la bride de fichier courante est dans un système de fichiers absent au début de l'opération, et la spécification ne fait pas d'exception pour GETFH. Noter que cela signifie que le serveur ne va jamais envoyer au client une bride de fichier provenant d'un système de fichiers absent.

Étant donné cela, le client sait où est la racine du système de fichiers absent (/this/is/the/path) en notant où le changement de fsid s'est produit (entre "the" et "path"). L'attribut fs_locations_info donne aussi au client la localisation actuelle du système de fichiers absent, de sorte que la référence peut se faire. Le serveur donne au client le minimum d'informations sur le système de fichiers absent de sorte qu'il va y avoir très peu de place pour des problèmes de conflit entre les informations envoyées par le serveur référant et les informations du lieu de rattachement du système de fichiers. Aucune bride de fichier et très peu d'attributs sont présents sur le serveur référant, et le client peut traiter celles qu'il reçoit comme des informations transitoires avec comme fonction de permettre la référence.

11.15.2 Exemple de référence (READDIR)

Un autre contexte dans lequel un client peut rencontrer des référents est quand il fait un READDIR sur un répertoire dans lequel des sous-répertoires sont les racines de systèmes de fichiers absents.

Supposons qu'un tel répertoire soit lu comme suit :
* PUTROOTFH
* LOOKUP "this"
* LOOKUP "is"
* LOOKUP "the"
* READDIR (fsid, size, time_modify, mounted_on_fileid)

Dans ce cas, parce que rdattr_error n'est pas demandé, fs_locations_info n'est pas demandé, et certains des attributs ne peuvent pas être fournis, le résultat va être une erreur NFS4ERR_MOVED sur le READDIR, avec le résultat détaillé suivant :
* PUTROOTFH --> NFS_OK. La bride de fichier courante est à la racine du pseudo système de fichiers.
* LOOKUP "this" --> NFS_OK. La bride de fichier courante est pour /this et est dans le pseudo-fs.
* LOOKUP "is" --> NFS_OK. La bride de fichier courante est pour /this/is et est dans le pseudo-fs.
* LOOKUP "the" --> NFS_OK. La bride de fichier courante est pour /this/est/the et est dans le pseudo-fs.
* READDIR (fsid, taille, time_modify, mounted_on_fileid) --> NFS4ERR_MOVED. Noter que la même erreur aurait été retournée si /this/is/the avait migré, mais elle est retournée parce que le répertoire contient la racine d'un système de fichiers absent.

Donc supposons maintenant qu'on envoie à nouveau avec rdattr_error :
* PUTROOTFH
* LOOKUP "this"
* LOOKUP "is"
* LOOKUP "the"
* READDIR (rdattr_error, fsid, size, time_modify, mounted_on_fileid)

Le résultat va être :
* PUTROOTFH --> NFS_OK. La bride de fichier courante est à la racine du pseudo-fs.
* LOOKUP "this" --> NFS_OK. La bride de fichier courante est pour /this et est dans le pseudo-fs.
* LOOKUP "est" --> NFS_OK. La bride de fichier courante est pour /this/est et est dans le pseudo-fs.
* LOOKUP "the" --> NFS_OK. La bride de fichier courante est pour /this/est/the et est dans le pseudo-fs.
* READDIR (rdattr_error, fsid, size, time_modify, mounted_on_fileid) --> NFS_OK. Les attributs pour l'entrée de répertoire avec le composant nommé "path" vont seulement contenir rdattr_error avec la valeur NFS4ERR_MOVED, avec une valeur de fsid et une valeur pour mounted_on_fileid.

Supposons qu'on fasse un autre READDIR pour obtenir fs_locations_info (bien qu'on pourrait avoir utilisé directement un GETATTR, comme au paragraphe 11.15.1).
* PUTROOTFH
* LOOKUP "this"
* LOOKUP "is"
* LOOKUP "the"
* READDIR (rdattr_error, fs_locations_info, mounted_on_fileid, fsid, size, time_modify)

Le résultat serait :
* PUTROOTFH --> NFS_OK. La bride de fichier courante est à la racine du pseudo-fs.
* LOOKUP "this" --> NFS_OK. La bride de fichier courante est pour /this et est dans le pseudo-fs.
* LOOKUP "est" --> NFS_OK. La bride de fichier courante est pour /this/is et est dans le pseudo-fs.
* LOOKUP "the" --> NFS_OK. La bride de fichier courante est pour /this/is/the et est dans le pseudo-fs.
* READDIR (rdattr_error, fs_locations_info, mounted_on_fileid, fsid, size, time_modify) --> NFS_OK. Les attributs vont être comme montré ci-dessous.

Les attributs pour l'entrée de répertoire avec le composant nommé "path" vont seulement contenir :
* rdattr_error (valeur : NFS_OK)
* fs_locations_info
* mounted_on_fileid (valeur : fileid unique au sein du système de fichiers référent)
* fsid (valeur : valeur unique au sein du serveur référent)

Les attributs pour l'entrée "path" ne vont pas contenir size ou time_modify parce que ces attributs ne sont pas disponibles dans un système de fichiers absent.

11.16 Attribut fs_locations

L'attribut fs_locations est structuré de la façon suivante :

```
struct fs_location4 {
      utf8str_cis    server<>;
      pathname4    rootpath;
};

struct fs_locations4 {
      pathname4      fs_root;
      fs_location4    locations<>;
};
```

Le type de données fs_location4 est utilisé pour représenter la localisation d'un système de fichiers en fournissant un nom de serveur et le chemin pour la racine du système de fichiers au sein de l'espace de noms de ce serveur. Quand un ensemble de serveurs a des systèmes de fichiers correspondants au même chemin au sein de leurs espaces de noms, un dispositif de noms de serveurs peut être fourni. Une entrée dans le dispositif de serveurs est une chaîne UTF-8 et représente un des noms

d'hôte traditionnels du DNS, une adresse IPv4, une adresse IPv6, ou une chaîne de longueur zéro. Une adresse IPv4 ou IPv6 est représentée comme une adresse universelle (voir le paragraphe 3.3.9 et la [RFC5665]) moins le netid, et avec ou sans le suffixe ".p1.p2" en queue qui représente le numéro d'accès. Si le suffixe est omis, alors l'accès par défaut, 2049, DEVRAIT être supposé. Une chaîne de longueur zéro DEVRAIT être utilisée pour indiquer l'adresse courante utilisée pour l'appel RPC. Il n'est pas exigé que tous les serveurs qui partagent le même chemin de racine soient mentionnés dans une instance de fs_location4. Le dispositif de noms de serveurs est fourni comme une facilité. Les serveurs qui partagent le même chemin de racine peuvent aussi être mentionnés dans des entrées séparées de fs_location4 dans l'attribut fs_locations.

Le type de données fs_locations4 et l'attribut fs_locations contiennent chacun un dispositif de telles localisations. Comme l'espace de noms de chaque serveur peut être construit différemment, le champ "fs_root" est fourni. Le chemin représenté par fs_root représente la localisation du système de fichiers dans l'espace de noms du serveur courant, c'est-à-dire, celui du serveur à partir duquel l'attribut fs_locations a été obtenu. Le chemin fs_root est destiné à aider le client en référençant clairement la racine du système de fichiers dont les localisations sont rapportées, sans considération de l'objet au sein du système de fichiers courant que désigne la bride de fichier. La fs_root est simplement le nom de chemin que le client utilise pour atteindre l'objet sur le serveur courant (c'est-à-dire, l'objet auquel l'attribut fs_locations s'applique).

Quand l'attribut fs_locations est interrogé et qu'il n'y a pas de localisation de système de fichiers de remplacement, le serveur DEVRAIT retourner un dispositif de longueur zéro de structures de fs_location4, avec une fs_root valide.

Par exemple, supposons qu'il y ait un système de fichiers dupliqué situé sur deux serveurs (servA et servB). Au servA, le système de fichiers est situé au chemin /a/b/c. Au servB, le système de fichiers est situé au chemin /x/y/z. Si le client veut obtenir la valeur de fs_locations pour le répertoire à /a/b/c/d, il pourrait ne pas nécessairement savoir que la racine du système de fichiers est située dans l'espace de noms de servA à /a/b/c. Quand le client passe à servB, il va devoir déterminer que le répertoire qu'il a d'abord référencé au servA est maintenant représenté par le chemin /x/y/z/d sur servB. Pour faciliter cela, l'attribut fs_locations fourni par servA aurait une valeur de fs_root de /a/b/c et deux entrées dans fs_locations. Une entrée dans fs_locations va être pour lui-même (servA) et l'autre va être pour servB avec un chemin de /x/y/z. Avec cette information, le client est capable de substituer /x/y/z à /a/b/c au début de son chemin d'accès et de construire /x/y/z/d à utiliser pour le nouveau serveur.

Noter qu'il n'est pas exigé que le nombre de composants dans chaque chemin de racine soit le même ; il n'y a pas de relation entre le nombre de composants dans rootpath ou fs_root, et aucun des composants dans un chemin de racine et dans fs_root ne doit être le même. Dans l'exemple ci-dessus, on pourrait avoir eu un troisième élément dans la matrice de localisations, avec serveur égal à "servC" et rootpath égal à "/I/II", et un quatrième élément dans les localisations avec serveur égal à "servD" et rootpath égal à "/aleph/beth/gimel/daleth/he".

La relation entre fs_root et un chemin de racine est que le client remplace le nom de chemin indiqué dans fs_root pour le serveur courant par le substitut indiqué dans le chemin de racine pour le nouveau serveur.

Pour un exemple de système de fichiers référé ou migré, supposons qu'il y ait un système de fichiers situé à serv1. À serv1, le système de fichiers est situé à /az/buky/vedi/glagoli. Le client trouve que cet objet à glagoli a migré (ou est une référence). Le client obtient l'attribut fs_locations, qui contient une fs_root de /az/buky/vedi/glagoli, et un élément dans le dispositif de localisations, avec serveur égal à serv2, et rootpath égal à /izhitsa/fita. Le client remplace /az/buky/vedi/glagoli par /izhitsa/fita, et utilise ce dernier nom de chemin sur serv2.

Donc, le serveur DOIT retourner une fs_root égale au chemin que le client a utilisé pour atteindre l'objet auquel s'applique l'attribut fs_locations. Autrement, le client ne peut pas déterminer le nouveau chemin à utiliser sur le nouveau serveur.

Comme l'attribut fs_locations manque des informations définissant les divers attributs des divers choix de système de fichiers présentés, il DEVRAIT seulement être interrogé et utilisé quand fs_locations_info n'est pas disponible. Quand fs_locations est utilisé, les informations sur les localisations spécifiques devraient être supposées fondées sur les règles suivantes.

Les règles suivantes sont générales et s'appliquent sans considération du contexte.

* Toutes les instances de système de fichiers mentionnées devraient être considérées comme étant de la même classe de brides, si et seulement si, l'attribut courant fh_expire_type n'inclut pas le bit FH4_VOL_MIGRATION. Noter que dans le cas de référents, les questions de bride de fichier ne s'appliquent pas car il peut ne pas y avoir de bride de fichier connue dans le système de fichiers en cours, ni d'accès à l'attribut fh_expire_type sur le système de fichiers référant (absent).

* Toutes les instances de système de fichiers mentionnées devraient être considérées être de la même classe d'identifiant

de fichier si et seulement si l'attribut fh_expire_type indique des brides de fichier persistantes et n'inclut pas de bit FH4_VOL_MIGRATION. Noter que dans le cas de référents, les questions d'identifiant de fichier ne s'appliquent pas parce que il peut n'y avoir aucun identifiant de fichier connu dans le système de fichiers référent (absent) ni d'accès à l'attribut fh_expire_type.

* Tous les serveurs d'instances de système de fichiers devraient être considérés comme de classes de changement différentes.

Pour les autres allocations de classe, le traitement des transitions de système de fichiers dépend des raisons de la transition :
* Quand la transition est due à la migration, c'est-à-dire, le client a été dirigé sur un nouveau système de fichiers après réception d'une erreur NFS4ERR_MOVED, la cible devrait être traitée comme étant de la même classe de vérificateur d'écriture que la source.
* Quand la transition est due à une reprise sur défaillance à une autre réplique, c'est-à-dire, le client a choisi une autre réplique sans recevoir une erreur NFS4ERR_MOVED, la cible devrait être traitée comme étant d'une classe de vérificateur d'écriture différente que la source.

Les choix spécifiques reflètent des schémas typiques de mise en œuvre pour la reprise sur défaillance et la migration contrôlée, respectivement. Comme d'autres choix sont possibles et utiles, cette information est mieux obtenue en utilisant fs_locations_info. Quand une mise en œuvre de serveur a besoin de communiquer d'autres choix, elle DOIT prendre en charge l'attribut fs_locations_info.

Voir à la Section 21 une discussion des recommandations sur la nuance de sécurité à utiliser par toute opération GETATTR qui demande l'attribut fs_locations.

11.17 Attribut fs_locations_info

L'attribut fs_locations_info est destiné à un remplacement plus fonctionnel de l'attribut fs_locations, qui va continuer d'exister et d'être pris en charge. Les clients peuvent l'utiliser pour obtenir un ensemble plus complet de données sur des localisations de remplacement de système de fichiers, incluant des chemins de réseau supplémentaires pour accéder aux répliques utilisées et des répliques supplémentaires. Quand le serveur ne prend pas en charge fs_locations_info, fs_locations peut être utilisé pour obtenir un sous ensemble des données. Un serveur qui prend en charge fs_locations_info DOIT aussi prendre en charge fs_locations.

Il y a des données supplémentaires présentes dans fs_locations_info qui ne sont pas disponibles dans fs_locations :

* Informations de continuité d'attribut. Ces informations vont permettre à un client de choisir une réplique qui satisfait les exigences de transparence des applications qui accèdent aux données et de renforcer les optimisations dues aux garanties du serveur de continuité d'attribut (par exemple, si l'attribut de changement d'un fichier de système de fichiers est continu entre plusieurs répliques, le client n'a pas à invalider l'antémémoire du fichier quand il passe à une réplique différente).

* Les informations d'identité de système de fichier qui indiquent quand plusieurs répliques, du point de vue du client, correspondent au même système de fichiers cible, leur permettant d'être utilisés de façon interchangeable, sans interruption, comme des répliques distinctes synchronisées des mêmes données de fichier.
  Noter qu'avoir deux répliques avec des informations d'identité communes est distinct du cas de deux chemins (multiplexés) de la même réplique.

* Les informations qui vont porter sur la convenance des diverses répliques, selon l'utilisation que le client entend en faire. Par exemple, de nombreuses applications ont besoin d'une copie absolument à jour (par exemple, celles qui écrivent) tandis que d'autres peuvent seulement avoir besoin d'accéder à la copie la plus à jour raisonnablement disponible.

* Les informations de préférence déduites du serveur pour les répliques, qui peuvent être utilisées pour mettre en œuvre l'équilibrage de charge tout en donnant au client la liste entière des systèmes de fichiers à utiliser en cas d'échec du principal.

L'attribut fs_locations_info est structuré de façon similaire à l'attribut fs_locations. Une structure de niveau supérieur (fs_locations_info4) contient l'attribut entier y compris le nom de chemin racine du système de fichiers et un dispositif de structures de niveau inférieur qui définit les répliques qui partagent un chemin de racine commun sur leurs serveurs respectifs. La structure de niveau inférieur (fs_locations_item4) contient à son tour un nom de chemin spécifique et des

informations sur un ou plusieurs chemins d'accès réseau individuels. Pour ces dernières, de niveau inférieur, fs_locations_info a une structure fs_locations_server4 qui contient des informations de réplique par serveur en plus de l'entrée de localisation du système de fichiers. Ces informations de réplique par serveur incluent un dispositif nominalement opaque, fls_info, au sein duquel des éléments d'information spécifiques sont situés aux indices spécifiques mentionnés ci-dessous.

Deux entrées de fs_location_server4 qui sont dans des structures différentes de fs_location_item4 ne sont jamais multiplexables, tandis que deux entrées au sein de la même structure fs_location_item4 pourraient ou non être multiplexables. Deux entrées qui sont multiplexables vont avoir des informations d'identité identiques, bien que, comme noté ci-dessus, l'inverse ne soit pas vrai.

L'attribut va toujours contenir au moins une seule entrée fs_locations_server. Normalement, il va y avoir une entrée avec le fanion FS4LIGF_CUR_REQ établi, bien que dans le cas d'un référent il ne va y avoir aucune entrée avec ce fanion établi.

On devrait noter que les attributs fs_locations_info retournés par les serveurs pour diverses répliques peuvent différer pour diverses raisons. Un serveur peut connaître un ensemble de répliques qui ne sont pas connues des autres serveurs. De plus, les attributs de compatibilité peuvent différer. Les brides de fichiers pourraient être de même classe allant de la réplique A à la réplique B mais n'allant pas dans la direction inverse. Cela pourrait arriver parce que les brides de fichier sont les mêmes, mais la mise en œuvre de serveur de la réplique B ne pourrait pas avoir de disposition pour noter et rapporter cette équivalence.

L'attribut fs_locations_info consiste en un nom de chemin de racine (fli_fs_root, tout comme fs_root dans l'attribut fs_locations) avec un dispositif de structures fs_location_item4. Les structures fs_location_item4 consistent à leur tour en un nom de chemin de racine (fli_rootpath) avec un dispositif (fli_entries) d'éléments de type de données fs_locations_server4, toutes définies comme suit :

```
/** Définit un chemin d'accès de serveur individuel */
struct  fs_locations_server4 {
        int32_t         fls_currency;
        opaque          fls_info<>;
        utf8str_cis     fls_server;
};

/* * Octets d'indices d'éléments au sein de fls_info : champs de fanion, numéros de classe, octets indiquant le rang et
    l'ordre. */
const FSLI4BX_GFLAGS            = 0;
const FSLI4BX_TFLAGS            = 1;
const FSLI4BX_CLSIMUL           = 2;
const FSLI4BX_CLHANDLE          = 3;
const FSLI4BX_CLFILEID          = 4;
const FSLI4BX_CLWRITEVER        = 5;
const FSLI4BX_CLCHANGE          = 6;
const FSLI4BX_CLREADDIR         = 7;
const FSLI4BX_READRANK          = 8;
const FSLI4BX_WRITERANK         = 9;
const FSLI4BX_READORDER         = 10;
const FSLI4BX_WRITEORDER        = 11;

/* * Bits définis dans l'octet de fanions général. */
const FSLI4GF_WRITABLE          = 0x01;
const FSLI4GF_CUR_REQ           = 0x02;
const FSLI4GF_ABSENT            = 0x04;
const FSLI4GF_GOING             = 0x08;
const FSLI4GF_SPLIT             = 0x10;

/** Bits définis dans l'octet de fanions de transport. */
const FSLI4TF_RDMA              = 0x01;

/** Définit un ensemble de répliques partageant une valeur commune de chemin racine dans l'espace de noms
    correspondant d'un seul serveur. */
struct  fs_locations_item4 {
```

```
        fs_locations_server4    fli_entries<>;
        pathname4               fli_rootpath;
};

/* * Définit la structure globale de l'attribut fs_locations_info. */
struct  fs_locations_info4 {
        uint32_t                fli_fanions;
        int32_t                 fli_valid_for;
        pathname4               fli_fs_root;
        fs_locations_item4      fli_items<>;
};

/* * Bits fanions dans fli_flags.  */
const FSLI4IF_VAR_SUB           = 0x00000001;

typedef fs_locations_info4 fattr4_fs_locations_info;
```

Comme noté ci-dessus, l'attribut fs_locations_info, quand il est pris en charge, peut être demandé de systèmes de fichiers absents sans causer le retour de NFS4ERR_MOVED. On s'attend généralement à ce qu'il soit disponible pour les systèmes de fichiers présents et absents même si seulement une entrée fs_locations_server4 est présente, désignant le système de fichiers courant (présent) ou si deux entrées de fs_locations_server4 désignant la localisation précédente d'un système de fichiers absent (celui qui vient juste d'être référencé) et la localisation de son successeur. Les serveurs sont fortement invités à prendre en charge cet attribut sur tous les systèmes de fichiers si ils prennent en charge les systèmes de fichiers.

Les données présentées dans l'attribut fs_locations_info peuvent être obtenues par le serveur de différentes façons, incluant la spécification par l'administrateur ou par les protocoles courants pour transférer les données entre les répliques et protocoles non encore développés. NFSv4.1 définit seulement comment ces informations sont présentées par le serveur au client.

11.17.1 Structure fs_locations_server4

La structure fs_locations_server4 consiste en les éléments suivants en plus du champ fls_server, qui spécifie une adresse réseau ou ensemble d'adresses à utiliser pour accéder au système de fichiers spécifié. Noter que ces deux éléments (c'est-à-dire, fls_currency et fls_info) spécifient des attributs de la réplique de système de fichiers et ne devraient pas être différents quand il y a plusieurs structures fs_locations_server4, chacune spécifiant un chemin réseau pour la réplique choisie, pour la même réplique.

Quand ces valeurs sont différentes dans deux structures fs_locations_server4, un client n'a pas de base pour choisir l'une plutôt que l'autre et le mieux est simplement d'ignorer les deux entrées, si ces entrées s'appliquent à une duplication ou référence de migration. Quand il y a plus de deux de ces entrées, le vote majoritaire peut être utilisé pour exclure la prise en compte d'une seule entrée erronée. Dans le cas où des informations de multiplexage sont fournies pour une réplique dont l'accès est en cours, les adresses multiplexées supplémentaires peuvent être ignorées alors que l'accès se continue sur l'adresse actuellement utilisée, même si l'entrée correspondant à ce chemin pourrait être considérée comme invalide.

* Une indication de comment mettre à jour le système de fichiers est (fls_currency) en secondes. Cette valeur est relative à la copie maîtresse. Une valeur négative indique que le serveur est incapable de donner une valeur raisonnablement utile. Une valeur de zéro indique que le système de fichiers est la copie réelle écrivable des données ou une copie fiable cohérente et pleinement à jour. Des valeurs positives indiquent de combien cette copie peut être dépassée avant que sa mise à jour soit normalement considérée. Cette valeur n'est pas une garantie que de telles mises à jour vont toujours être effectuées selon la programmation requise mais sert plutôt d'indication sur la distance que la copie des données devrait être supposée conserver par rapport à la copie la plus à jour.

* Un dispositif de compteur de valeurs de un octet (fls_info) contenant des informations sur l'instance particulière de système de fichiers. Ces données incluent des fanions généraux, des fanions de capacité de transport, des informations de classe d'équivalence de système de fichiers, et des informations de priorité de choix. Le codage va être discuté plus loin.

* La chaîne de serveur (fls_server). Pour le cas de la réplique actuellement accédée (via GETATTR) une chaîne de longueur zéro PEUT être utilisée pour indiquer l'adresse actuellement utilisée pour l'appel RPC. Le champ fls_server peut aussi être une adresse IPv4 ou IPv6, formatée de la même façon qu'une adresse IPv4 ou IPv6 dans le champ

"server" du type de données fs_location4 (voir le paragraphe 11.16).

À l'exception du champ Fanions de transport (au décalage FSLI4BX_TFLAGS avec dispositif fls_info) toutes les données définies dans cette spécification s'appliquent à la réplique spécifiée par l'entrée, plutôt que le chemin réseau spécifique utilisé pour y accéder. Le classement des données dans les extensions à ces données est discuté ci-dessous.

Les données dans le dispositif fls_info sont sous la forme d'éléments de données de 8 bits avec des constantes donnant les décalages au sein du dispositif de diverses valeurs qui décrivent cette instance particulière de système de fichiers. Ce style de définition a été choisi, de préférence à des définitions explicites de structure XDR pour ces valeurs, pour un certain nombre de raisons :

* Les sortes de données dans le dispositif fls_info, représentant des fanions, des classes de système de fichiers, et des priorités parmi des ensembles de systèmes de fichiers représentant les même données, sont telles que 8 bits fournissent une gamme acceptable de valeurs. Même lorsque il pourrait y avoir plus de 256 de ces instances de système de fichiers, avoir plus de 256 classes ou priorités distinctes est peu probable.
* Une définition explicite des divers éléments de données spécifiques en XDR limiterait la capacité d'expansion en ce que toute extension exigerait encore un autre attribut, conduisant à la complication de la spécification et de la mise en œuvre. Dans le contexte du modèle d'extension de NFSv4 en vigueur au moment où fs_locations_info a été conçu (c'est-à-dire, celui qui est décrit dans la [RFC5661]) cela aurait nécessité une nouvelle version mineure pour effectuer l'extension sur la voie de la normalisation aux données dans fls_info.

L'ensemble de données fls_info est sujet à expansion dans une future version mineure ou dans une RFC sur la voie de la normalisation dans le contexte d'une seule version mineure. Le serveur NE DEVRAIT PAS envoyer et le client NE DOIT PAS utiliser des indices dans le dispositif fls_info ou des bits fanions qui ne sont pas définis dans une RFC sur la voie de la normalisation.

À la lumière du nouveau modèle d'extension défini dans la [RFC8178] et du fait que les éléments individuels dans fls_info ne sont pas explicitement référencés dans le XDR, les pratiques suivantes devraient être suivies quand on étend ou change autrement la structure des données retournées dans fls_info dans la portée d'une seule version mineure :

* Toutes les extensions doivent être décrites par des documents sur la voie de la normalisation. Il n'est pas nécessaire que ces documents soient marqués comme mettant à jour la [RFC5661] ou le présent document.
* Il doit apparaître clairement si les informations dans un élément de données ajouté s'appliquent à la réplique spécifiée par l'entrée ou aux chemins réseau spécifiques spécifiés dans l'entrée.
* Il doit être défini un moyen fiable pour déterminer si le serveur a connaissance de l'extension. Cela peut être fondé sur le champ Longueur du dispositif fls_info, mais il est plus souple de fournir des attributs de portée système de fichiers ou serveur pour indiquer quelles extensions sont fournies.

Ce schéma de codage peut être adapté à la spécification de valeurs numériques multi octets, même si aucune n'est actuellement définie. Si des extensions sont faites par des RFC sur la voie de la normalisation, des quantités multi octets seront codées comme une gamme d'octets avec une gamme d'indices, avec l'octet interprété en ordre gros boutien. De plus, toutes ces allocations d'indices vont être contraintes par la nécessité que les quantités pertinentes ne dépassent pas les limites de mots XDR.

Le dispositif fls_info contient actuellement :

* Deux champs fanions de 8 bits, un destiné aux caractéristiques générales de système de fichiers et le second réservé pour les capacités relatives au transport.
* Six valeurs de classe de 8 bits qui définissent les diverses classes d'équivalence de système de fichiers comme expliqué ci-dessous.
* Quatre valeurs de priorité de 8 bits qui gouvernent le choix de système de fichiers comme expliqué ci-dessous.

Le fanion de caractéristiques générales de système de fichiers (à l'indice d'octet FSLI4BX_GFLAGS) a les bits suivants définis en son sein :

* FSLI4GF_WRITABLE indique que ce système de fichiers cible est écrivable, lui permettant d'être choisi par des clients qui peuvent avoir besoin d'écrire sur ce système de fichiers. Quand l'instance courante de système de fichiers est écrivable et est définie comme étant de la même classe d'utilisation simultanée (comme spécifié par la valeur de l'indice FSLI4BX_CLSIMUL) à laquelle le client écrivait précédemment, il doit alors incorporer dans ses données toute écriture engagée sur l'instance de système de fichiers de source. Voir au paragraphe 11.11.6, qui discute de la classe de vérificateur d'écriture. Bien qu'il n'y ait pas de dommage à ne pas établir ce fanion pour un système de fichiers qui se révèle être écrivable, établir le fanion pour un système de fichiers en lecture seule peut causer des problèmes aux clients qui choisissent une cible de migration ou de duplication sur la base du fanion et se trouvent alors incapables d'écrire eux-mêmes.

* FSLI4GF_CUR_REQ indique que cette réplique est celle sur laquelle la demande est faite. Une seule entrée de serveur peut avoir ce fanion établi et, dans le cas d'une référence, aucune entrée ne l'aura établi. Noter que ce fanion pourrait être établi même si la demande était faite sur un chemin d'accès réseau différent de tous ceux spécifiés dans l'entrée courante.

* FSLI4GF_ABSENT indique que cette entrée correspond à une réplique de système de fichiers absent. Il peut seulement être établi si FSLI4GF_CUR_REQ est établi. Quand ces deux bits sont établis, il indique qu'une instance de système de fichiers n'est pas utilisable mais que les informations de l'entrée peuvent être utilisées pour déterminer la sorte de continuité disponible quand on passe de cette réplique à une autre réplique possible. Comme ce bit peut seulement être vrai si FSLI4GF_CUR_REQ est vrai, la valeur pourrait être déterminée en utilisant l'attribut fs_status, mais l'information est aussi rendue disponible ici pour l'agrément du client. Une entrée avec ce bit, comme il représente un vrai système de fichiers (bien qu'absent) n'apparaît pas en cas de référence, mais seulement quand un système de fichiers a été accédé à cette localisation et a ensuite migré.

* FSLI4GF_GOING indique qu'une réplique, bien qu'encore disponible, ne devrait plus être utilisée. Le client, si il l'utilise, devrait faire un transfert ordonné à une autre instance de système de fichiers aussi rapidement que possible. On s'attend à ce que les systèmes de fichiers qui cessent le service vont être annoncés comme FSLI4GF_GOING un peu avant la perte de service réelle. On s'attend aussi à ce que la valeur fli_valid_for soit suffisamment petite pour permettre aux clients de détecter et agir sur les événements programmés, et suffisamment grande pour que le coût des demandes pour aller chercher les valeurs de fs_locations_info ne soit pas excessif. Des valeurs de l'ordre de dix minutes semblent raisonnables. Quand ce fanion est vu au titre d'une transition dans un nouveau système de fichiers, un client pourrait choisir de se transférer immédiatement à une autre réplique, ou il peut faire référence au système de fichiers courant et faire seulement la transition quand un événement de migration se produit. De même, quand ce fanion apparaît comme une réplique dans le référent, les clients vont probablement éviter d'être référencés par cette instance chaque fois qu'il y a un autre choix. Ce fanion, comme les autres éléments dans fls_info, s'applique à la réplique plutôt qu'à un chemin particulier vers cette réplique. Quand il apparaît, une transition à une nouvelle réplique, plutôt que à un chemin différent pour la même réplique, est indiqué.

* FSLI4GF_SPLIT indique que quand une transition se produit de l'instance de système de fichiers courant à celle-ci, le remplacement peut consister en plusieurs systèmes de fichiers. Dans ce cas, le client doit être prêt à la possibilité que des objets sur le même système de fichiers avant migration soient sur des systèmes différents après elle. Noter que FSLI4GF_SPLIT n'est pas incompatible avec des systèmes de fichiers appartenant à la même classe d'identifiants de fichiers car, si l'un a un ensemble d'identifiants de fichiers qui sont uniques au sein d'un système de fichiers, chaque sous ensemble alloué à un plus petit système de fichiers après migration ne va pas avoir de conflit interne avec ce système de fichiers.

Un client, dans le cas de système de fichiers partagé, va interroger les fichiers existants avec lesquels il a une connexion continue (il est libre de simplement oublier les brides de fichier en antémémoire). Si le client se souvient de la bride de fichier de répertoire associée à chaque fichier ouvert, il peut procéder vers l'amont en utilisant LOOKUPP pour trouver les frontières du nouveau système de fichiers. Noter qu'en cas de référence, il ne va pas y avoir de tels fichiers et donc ces actions ne seront pas effectuées. À la place, une référence à une portion du système de fichiers original maintenant partagé en d'autres systèmes de fichiers va rencontrer un changement de fsid et éventuellement un autre référent.

Une fois que le client a reconnu qu'un système de fichiers a été partégé en deux, il peut empêcher la perturbation des applications qui courent en présentant les deux systèmes de fichiers comme un seul jusqu'à ce qu'un point convenable reconnaisse la transition, comme un redémarrage. Cela exigerait une transposition des fsids du serveur en des fsids vus par le client, mais ceci est déjà nécessaire pour d'autres raisons. Comme noté ci-dessus, les fileids existants au sein des deux systèmes de fichiers descendants ne vont pas entrer en conflit. Fournir des fileids non en conflit pour des fichiers nouvellement créés sur les systèmes de fichiers partagés est de la responsabilité du serveur (ou des serveurs qui fonctionnent de concert). Le serveur peut coder des brides de fichier de telle sorte que les brides de fichier générées avant l'événement de partage puissent être discernées de celles générées après le partage, permettant au serveur de déterminer quand le besoin d'émuler deux systèmes de fichiers comme un seul est terminé.

Bien qu'il soit possible que ce fanion soit présent en cas de référent, il va généralement être de peu d'intérêt pour le client, car le client n'est pas supposé avoir des informations concernant le contenu actuel du système de fichiers absent.

Le champ Fanions de transport (à l'indice d'octet FSLI4BX_TFLAGS) contient les bits suivants relatifs aux capacités de transport des chemins spécifiques de réseau) spécifiées par l'entrée :

* FSLI4TF_RDMA indique que tous les chemins réseau spécifiés fournissent aux clients NFSv4.1 l'accès en utilisant un

transport capable de RDMA.

La continuité d'attribut et les informations d'identité de système de fichiers sont exprimées en définissant des relations d'équivalence sur les ensembles de systèmes de fichiers présentés au client. Chacune de ces relations est exprimée comme un ensemble de classes d'équivalence de système de fichiers. Pour chaque relation, un système de fichiers a un numéro de classe de 8 bits. Deux systèmes de fichiers appartiennent à la même classe si les deux ont un numéro de classe identique non zéro. Zéro est traité comme non correspondant. Le plus souvent, la question pertinente pour le client va être si une certaine réplique est identique/continue avec celle en cours à un certain égard, mais les informations devraient être disponibles aussi sur si deux autres répliques correspondent aussi à cet égard.

Les champs suivants spécifient les numéros de classe de système de fichiers pour les relations d'équivalence utilisées pour déterminer la nature des transitions de système de fichiers. Voir les paragraphes 11.9 à 11.14 et leurs divers sous paragraphes pour les détails sur la façon dont ces informations sont à utiliser. Les serveurs peuvent allouer ces valeurs comme ils souhaitent, pour autant que les instances de système de fichiers qui partagent la même valeur aient la relation spécifiée de l'une à l'autre ; à l'inverse, les systèmes de fichiers qui ont la relation spécifiée de l'un à l'autre partagent une valeur de classe commune. Lorsque chaque entrée d'instance est ajoutée, la relation de cette instance aux instances entrées précédemment peut être consultée, et si il en est trouvé une qui porte la relation spécifiée, cette valeur de classe d'entrée peut être copiée à la nouvelle entrée. Quand il n'existe pas une telle entrée précédente, une nouvelle valeur pour cet indice d'octet (non utilisé précédemment) peut être choisie, très probablement en incrémentant la valeur de la dernière valeur de classe allouée pour cet indice.

* Le champ avec l'indice d'octet FSLI4BX_CLSIMUL définit la classe d'utilisation simultanée pour le système de fichiers.
* Le champ avec l'indice d'octet FSLI4BX_CLHANDLE définit la classe de bride pour le système de fichiers.
* Le champ avec l'indice d'octet FSLI4BX_CLFILEID définit la classe de fileid pour le système de fichiers.
* Le champ avec l'indice d'octet FSLI4BX_CLWRITEVER définit la classe de vérificateur d'écriture pour le système de fichiers.
* Le champ avec l'indice d'octet FSLI4BX_CLCHANGE définit la classe de changement pour le système de fichiers.
* Le champ avec l'indice d'octet FSLI4BX_CLREADDIR définit la classe de readdir pour le système de fichiers.

Des informations de préférence spécifiée par le serveur sont aussi fournies via des valeurs de 8 bits dans le dispositif fls_info. Les valeurs fournissent un rang et un ordre (voir ci-dessous) à utiliser avec des valeurs séparées spécifiables pour les systèmes de fichiers en lecture seule et écrivables. Ces valeurs sont comparées pour les différents systèmes de fichiers pour établir la préférence spécifiée par le serveur, avec les valeurs inférieures qui indiquent la "plus préférée".

Le rang est utilisé pour exprimer un ordre strict imposé par le serveur aux clients, avec les valeurs inférieures qui indiquent la plus grande préférence. Les clients devraient tenter d'utiliser toutes les répliques avec un rang donné avant d'en utiliser une de rang supérieur. C'est seulement si tous ces systèmes de fichiers sont indisponibles que le client devrait traiter ceux de rang supérieur. Parce que spécifier un rang outrepasse les préférences de client, les serveurs devrait être prudents dans l'utilisation de ce mécanisme, en particulier dans un environnement où les caractéristiques de communication du client ne sont ni étroitement contrôlées ni visibles par le serveur.

Dans un rang, la valeur d'ordre est utilisée pour spécifier la préférence du serveur pour guider le choix du client quand les propres préférences du client ne sont pas contrôlantes, avec les valeurs inférieurs d'ordre qui indiquent celui préféré. Si les répliques sont approximativement égales sous tous les aspects, les clients devraient respecter l'ordre spécifié par le serveur. Quand les clients regardent la latence du serveur au titre de leur choix, ils sont libres d'utiliser ce critère, mais il est suggéré que quand les différences de latence ne sont pas significatives, l'ordre spécifié par le serveur devrait guider le choix.

* Le champ avec l'indice d'octet FSLI4BX_READRANK donne la valeur de rang à utiliser pour l'accès en lecture seule.
* Le champ avec l'indice d'octet FSLI4BX_READORDER donne la valeur d'ordre à utiliser pour l'accès en lecture seule.
* Le champ avec l'indice d'octet FSLI4BX_WRITERANK donne la valeur de rang à utiliser pour l'accès en écriture.
* Le champ avec l'indice d'octet FSLI4BX_WRITEORDER donne la valeur d'ordre à utiliser pour l'accès en écriture.

Selon le besoin potentiel d'accès en écriture pour un client donné, une des paires de valeurs de rang et ordre est utilisée. Le rang et ordre en lecture devrait seulement être utilisé si le client sait que seulement une lecture sera faite ou si il est prêt à passer à une réplique différente au cas où une capacité d'accès en écriture serait requise à un moment donné.

11.17.2 Structure fs_locations_info4

La structure fs_locations_info4, codant l'attribut fs_locations_info, contient ce qui suit :

* Le champ fli_flags contient des fanions généraux qui affectent l'interprétation de cette structure fs_locations_info4 et toutes les structures fs_locations_item4 dans elle. Le seul fanion actuellement défini est FSLI4IF_VAR_SUB. Tous les bits dans le champ fli_flags qui ne sont pas définis devraient toujours être retournés à zéro.

* Le champ fli_fs_root contient le nom de chemin de la racine du système de fichiers courant sur le serveur courant, juste comme il le fait dans la structure fs_locations4.

* Un dispositif appelé fli_items des structures fs_locations4_item, qui contient des informations sur les répliques du système de fichiers courant. Lorsque le système de fichiers courant est réellement présent, ou a été présent, c'est-à-dire, ce n'est pas une situation de référent, une des structures fs_locations_item4 va contenir un fs_locations_server4 pour le serveur courant. Cette structure va avoir FSLI4GF_ABSENT établi si le système de fichiers courant est absent, c'est-à-dire, l'accès normal va retourner NFS4ERR_MOVED.

* Le champ fli_valid_for spécifie un temps en secondes pendant lequel il est raisonnable pour un client d'utiliser l'attribut fs_locations_info sans retourner le chercher. La valeur fli_valid_for ne donne pas de garantie de validité parce que les serveurs peuvent être hors service de façon inattendue ou devenir inaccessibles pour toutes sortes de raisons. Les clients seront bien avisés de retourner chercher cette information pour un système de fichiers accédé de façon active toutes les fli_valid_for secondes. Ceci est particulièrement important quand les répliques de système de fichiers peuvent cesser le service de façon incontrôlée en utilisant le fanion FSLI4GF_GOING pour communiquer un changement en cours. Le serveur devrait établir fli_valid_for à une valeur qui permette à des clients au bon comportement de remarquer le fanion FSLI4GF_GOING et faire un passage ordonné avant que la perte de service devienne effective. Si cette valeur est zéro, alors l'intervalle de non recherche est appropriée et le client n'a pas besoin de retourner chercher ces données selon une programme particulier. En cas de transition à une nouvelle instance de système de fichiers, une nouvelle valeur de l'attribut fs_locations_info va être allée cherchée à la destination. Il faut s'attendre à ce que cela puisse avoir une valeur différente de fli_valid_for, que le client devrait alors utiliser de la même façon que la précédente valeur. Parce que retourner chercher l'attribut cause la reprise des informations provenant de toutes les entrées de composant, le serveur va normalement fournir une faible valeur pour ce champ si une des répliques va probablement être hors service dans un bref délai. Noter que, à cause de la capacité du serveur de retourner NFS4ERR_MOVED pour déclencher l'utilisation de chemins différents, quand des chemins de multiplexage de remplacement sont disponibles, il n'est généralement pas besoin d'utiliser de faibles valeurs de fli_valid_for en connexion avec la gestion de chemins de remplacement pour la même réplique.

Le fanion FSLI4IF_VAR_SUB dans fli_flags contrôle si une substitution de variable doit être activée. Voir au paragraphe 11.17.3 l'explication de la substitution de variable.

11.17.3 Structure fs_locations_item4

La structure fs_locations_item4 contient un nom de chemin (dans le champ fli_rootpath) qui code le chemin des répliques du système de fichiers cible sur l'ensemble de serveurs désigné par les entrées incluses de fs_locations_server4. La façon précise dont cette localisation de cible est spécifiée dépend de la valeur du fanion FSLI4IF_VAR_SUB dans la structure fs_locations_info4 associée.

Si ce fanion n'est pas établi, alors fli_rootpath désigne simplement la localisation du système de fichiers cible au sein de l'espace de noms d'un seul serveur de chaque serveur tout comme il le fait pour le chemin de racine dans la structure fs_location4. Quand ce bit est établi, cependant, les entrées de composant d'une certaine forme sont soumises à une substitution de variable spécifique du client afin de permettre un certain degré de non uniformité d'espace de noms afin de s'accommoder du choix de cibles de système de fichiers spécifique du client de différentes architectures de client ou autres caractéristiques.

Quand une telle substitution est en vigueur, une variable commençant pas la chaîne "${" et se terminant par la chaîne "}" et contenant un caractère "deux-points" est à remplacer par une valeur spécifique du client associée à cette variable. La chaîne "inconnu" devrait être utilisée par le client quand il n'a pas de valeur pour une telle variable. Le nom de chemin résultant de ces substitutions est utilisé pour désigner le système de fichiers cible, afin que différents clients puissent avoir des systèmes de fichiers différents, correspondant à cette localisation dans l'espace de noms multi-serveurs.

Comme mentionné ci-dessus, ces variables de nom de chemin substituées contiennent un caractère "deux-points". La partie avant les deux points doit être un nom de domaine DNS, et la partie après doit être une chaîne alphanumérique insensible à la casse

Lorsque le domaine est "ietf.org", seuls les noms de variable définis dans le présent document ou des RFC sur la voie de la normalisation suivantes sont sujettes à une telle substitution. Les organisations sont libres d'utiliser leur nom de domaine pour créer leur propre ensemble de variables spécifique du client, pour être sujettes à une telle substitution. Dans des cas où de telles variables sont destinées à être utilisées plus largement qu'une seule organisation, la publication d'une RFC pour information définissant ces variables est RECOMMANDÉE.

La variable ${ietf.org:CPU_ARCH} est utilisée pour noter que les fichiers d'objets d'architecture de CPU sont compilés. La présente spécification ne limite pas les valeurs acceptables (sauf qu'elles doivent être des chaînes UTF-8 valides) mais des valeurs comme "x86", "x86_64", et "sparc" sont supposées être utilisées en ligne avec les pratiques de l'industrie.

La variable ${ietf.org:OS_TYPE} est utilisée pour noter le système d'exploitation, et donc les API de noyau et de bibliothèque, pour lesquelles le code pourrait être compilé. La présente spécification ne limite pas les valeurs acceptables (sauf qu'elles doivent être des chaînes UTF-8 valides) mais des valeurs comme "linux" et "freebsd" ont supposées être utilisées en ligne avec les pratiques de l'industrie.

La variable ${ietf.org:OS_VERSION} est utilisée pour noter la version du système d'exploitation, et donc les détails spécifiques des interfaces qui ont un numéro de version, pour lesquelles du code pourrait être compilé. La présente spécification ne limite pas les valeurs acceptables (sauf qu'elles doivent être des chaînes UTF-8 valides). Cependant, des combinaisons de nombres et lettres avec des points intercalés seraient supposés être utilisés en ligne avec les pratiques de l'industrie, avec les détails du format de version dépendant de la valeur spécifique de la variable ${ietf.org:OS_TYPE} avec laquelle il est utilisé.

L'utilisation de ces variables pourrait résulter en l'acheminement de différents clients sur différents systèmes de fichiers sur le même serveur, comme approprié pour les clients particuliers. Dans des cas où les systèmes de fichiers cibles sont situés sur des serveurs différents, un seul serveur pourrait servir de point de référence afin que chaque combinaison valide de valeurs de variable désigne un référent hébergé sur un seul serveur, avec les cibles de ces référents sur un certain nombre de serveurs différents.

Parce que l'administration d'espace de noms est affectée par les valeurs choisies pour se substituer aux diverses variables, les clients devraient fournir des moyens convenables de déterminer quelles substitutions de variables un client va mettre en œuvre ainsi que, lorsque approprié, de fournir des moyens de contrôler les substitutions à utiliser. Les moyens exacts par lesquels ceci va être fait sortent du domaine d'application de cette spécification.

Bien que l'utilisation de la substitution de variable soit surtout convenable dans le contexte de référents, elle peut être utilisée dans les contextes de duplication et de migration. Si elle est utilisée dans ces contextes, le serveur doit s'assurer que quelle que soient les valeurs que le client présente pour les variables substituées, le résultat est toujours un successeur valide de l'instance de système de fichiers dont une transition se produit, c'est-à-dire, que les données sont identiques ou représentent une image ultérieure d'un système de fichiers écrivable.

Noter que quand fli_rootpath est un nom de chemin nul (c'est-à-dire, une avec des composants zéro) le système de fichiers désigné est à la racine du serveur spécifié, que le fanion FSLI4IF_VAR_SUB dans la structure associée fs_locations_info4 soit établi ou non.

11.18 Attribut fs_status

Dans un environnement dans lequel plusieurs copies du même ensemble de base de données sont disponibles, les informations concernant la source particulière de ces données et les relations entre les différentes copies peuvent être très utiles en fournissant des données cohérentes aux applications.

```
enum fs4_status_type {
    STATUS4_FIXED        = 1,
    STATUS4_UPDATED      = 2,
    STATUS4_VERSIONED    = 3,
    STATUS4_WRITABLE     = 4,
    STATUS4_REFERRAL     = 5.
};

struct fs4_status {
    bool              fss_absent;
    fs4_status_type   fss_type;
```

```
    utf8str_cs      fss_source;
    utf8str_cs      fss_current;
    int32_t         fss_age;
    nfstime4        fss_version;
};
```

Le booléen fss_absent indique si le système de fichiers est actuellement absent. Cette valeur va être établie si le système de fichiers était précédemment présent et devient absent, ou si le système de fichiers n'a jamais été présent et si le type est STATUS4_REFERRAL. Quand ce booléen est établi et que le type n'est pas STATUS4_REFERRAL, les informations restantes dans le fs4_status reflètent les dernières valides quand le système de fichiers était présent.

Le champ fss_type indique la sorte d'image de système de fichiers représentée. Ceci est d'une particulière importance quand on utilise les valeurs de version pour déterminer la succession appropriée d'images de système de fichiers. Quand fss_absent est établi, et que le système de fichiers était précédemment présent, la valeur de fss_type reflétée est celle de quand le fichier était présent. Cinq valeurs sont distinguées :

* STATUS4_FIXED indique une image en lecture seule en ce sens qu'elle ne va jamais changer. Il est permis que, par suite d'une migration ou du passage à une image différente, on puisse accéder à des données changées, mais dans les limites de cette instance, aucun changement n'est permis. Le client peut utiliser ce fait pour une mise en antémémoire agressive.

* STATUS4_VERSIONED indique que l'image, comme dans le cas de STATUS4_UPDATED, est mise à jour en externe, mais fournit la garantie que le serveur va soigneusement mettre à jour une valeur de version associée afin que le client puisse se protéger contre une situation dans laquelle il lit des données provenant d'une version du système de fichiers et lit alors plus tard des données provenant d'une version antérieure du même système de fichiers. Voir ci-dessous une discussion de comment cela peut être fait.

* STATUS4_UPDATED indique une image qui ne peut pas être mise à jour par l'utilisateur qui écrit sur elle mais qui peut être changé en externe, normalement parce que c'est une copie périodiquement mise à jour d'un autre système de fichiers écrivable ailleurs. Dans ce cas, les informations de version ne sont pas fournies, et le client n'a pas la responsabilité de s'assurer que cette version avance seulement sur une transition d'instance de système de fichiers. Dans ce cas, il est de la responsabilité du serveur de s'assurer que les données présentées après une transition d'instance de système de fichiers sont une image suivante appropriée et incluent tous les changements vus par le client et tout changement fait avant tous ces changements.

* STATUS4_WRITABLE indique que le système de fichiers est réellement écrivable. Le client n'a pas besoin, bien sûr, de réellement écrire sur le système de fichiers, mais une fois qu'il le fait, il ne devrait pas accepter une transition à autre chose qu'une instance écrivable de ce même système de fichiers.

* STATUS4_REFERRAL indique que le système de fichiers en question est absent et n'a jamais été présent sur ce serveur.

Noter que dans les cas de STATUS4_UPDATED et STATUS4_VERSIONED, le serveur est responsable du traitement approprié des verrous qui ne sont pas cohérents avec les changements externes aux délégations. Si un serveur épuise les délégations, elles DEVRAIENT être rappelées avant que soit fait un changement non cohérent aux données, et elles DOIVENT être révoquées si ce n'est pas possible. De même, si un OPEN est incohérent avec des données qui ont changé (le OPEN a OPEN4_SHARE_DENY_WRITE/OPEN4_SHARE_DENY_BOTH et les données sont changées) cet OPEN DEVRAIT être considéré comme administrativement révoqué.

Les chaînes opaques fss_source et fss_current fournissent un moyen de présenter des informations sur la source de l'image de système de fichiers présente. Il n'est pas prévu que le client fasse quelque chose de ces informations autre que de les rendre disponibles à des outils administratifs. Il est prévu que ces informations soient utile quand on recherche de possibles problèmes avec une image de système de fichiers qui pourraient survenir quand il n'est pas clair si l'image correcte est accédée et, sinon, comment cette image est venue. Cette sorte d'informations de diagnostic va être utile si, comme il semble probable, des copies de systèmes de fichiers sont faites de nombreuses façons différentes (par exemple, simples copies de niveau utilisateur, copies de niveau système de fichier à un instant donné, clones de la mémorisation sous-jacente) sous divers arrangements administratifs. Dans ces environnements, déterminer comment un certain ensemble de données a été construit peut être très utile pour résoudre les problèmes.

La chaîne opaque fss_source est utilisée pour indiquer la source d'un certain système de fichiers avec l'espoir que des outils capables de créer une image de système de fichiers propagent ces informations, quand c'est possible. Il est admis que cela

n'est pas toujours possible car une copie de niveau utilisateur peut être vue comme la création d'un nouvel ensemble de données et les outils utilisés peuvent n'avoir pas de mécanisme pour propager ces données. Quand un système de fichiers est initialement créé, il est souhaitable de lui associer des données sur la façon dont le système de fichiers a été créé, où il a été créé, qui l'a créé, etc. Rendre ces informations disponibles dans cet attribut dans une chaîne lisible par l'homme va être utile pour les administrateurs d'applications et de systèmes et va aussi servir à les rendre disponibles quand le système de fichiers original est utilisé pour faire des copies suivantes.

La chaîne opaque fss_current devrait fournir toutes les informations disponibles sur la source de la copie courante. Ces informations incluent les outils de création, tous les paramètres pertinents de ces outils, le moment où la copie a été faite, l'utilisateur qui fait le changement, le serveur sur lequel le changement a été fait, etc. Toutes les informations devraient être dans une chaîne lisible par l'homme.

Le champ fss_age fournit une indication de combien de temps est dépassé le système de fichiers par rapport à ses dernières données de source (en cas de mises à jour de données en cascade). Cela complète le champ fls_currency de fs_locations_server4 (voir le paragraphe 11.17) de la façon suivante : les informations dans fls_currency donnent une limite sur le niveau de péremption que les données dans un système de fichiers pourraient normalement avoir, alors que la valeur dans fss_age donne une limite sur la péremption actuelle des données. Des valeurs négatives impliquent qu'aucune information n'est disponible. Un zéro signifie que ces données sont connues pour être actuelles. Une valeur positive signifie que ces données sont connues pour n'être pas plus vieilles que le nombre de secondes par rapport aux dernières données de source. En utilisant cette valeur, le client peut être capable de décider qu'une copie des données est trop vieille, et donc qu'il peut chercher une version plus récente à utiliser.

Le champ fss_version fournit une identification de version, sous la forme d'une valeur de temps, telle que les versions successives aient toujours des valeurs de temps postérieures. Quand le fs_type est autre chose que STATUS4_VERSIONED, le serveur peut fournir une telle valeur, mais il n'est pas garanti qu'elle soit valide et les clients ne vont pas l'utiliser sauf pour fournir des informations supplémentaires pour ajouter à fss_source et fss_current.

Quand fss_type est STATUS4_VERSIONED, les serveurs DEVRAIENT fournir une valeur de fss_version qui croisse de façon monotone chaque fois que une nouvelle version des données est établie. Cela permet au client, si une progression fiable de l'image est importante pour lui, d'aller chercher cet attribut au titre de chaque COMPOUND où des données ou métadonnées provenant du système de fichiers sont utilisées.

Quand il est important pour le client de s'assurer que seulement des images suivantes valides sont acceptées, il doit s'assurer qu'il ne lit pas des données ou métadonnées provenant du système de fichiers sans mettre à jour son sens de l'état actuel de l'image. C'est pour éviter la possibilité que le fs_status que le client détient soit celui d'une image antérieure, qui causerait l'acceptation par le client d'une nouvelle instance de système de fichiers qui serait antérieure à celle là mais encore plus ancienne que les données mises à jour lues par le client.

Afin d'accepter fiablement des images valides, le client doit faire un GETATTR de l'attribut fs_status qui suit toute interrogation de données ou métadonnées dans le système de fichiers en question. Souvent ceci est fait en ajoutant un tel GETATTR après toutes les autres opérations qui font référence à un certain système de fichiers. Quand des erreurs se produisent entre la lecture des données du système de fichiers et la réalisation de ce GETATTR, on doit veiller à s'assurer que les données en question ne sont pas utilisées avant d'avoir obtenu la valeur appropriée de fs_status. Dans cette perspective, quand un OPEN est fait dans un tel système de fichiers avec numéro de version et que le GETATTR associé de fs_status n'est pas achevé avec succès, le fichier ouvert en question ne doit pas être accédé avant que le fs_status soit ramené.

La procédure ci-dessus va assurer qu'avant d'utiliser aucune donnée provenant du système de fichiers le client a en main une version courante nouvellement ramenée de l'image du système de fichiers. Plusieurs valeurs pour plusieurs demandes en vol peuvent être résolues en les assemblant dans l'ordre partiel requis (et les éléments devraient former un ordre total au sein de l'ordre partiel) et en utilisant ce dernier. Le client peut alors, quand il passe d'une instance de système de fichiers à l'autre, refuser d'utiliser une instance qui n'a pas un fss_type de STATUS4_VERSIONED ou dont le champ fss_version est plus tôt que le dernier obtenu de l'instance de système de fichiers précédente.

12. NFS parallèle (pNFS)

12.1 Introduction

pNFS est une caractéristique FACULTATIVE au sein de NFSv4.1 ; l'ensemble de caractéristiques de pNFS permet un accès de client direct aux appareils de mémorisation qui contiennent des données de fichier. Quand des données de fichier

pour un seul serveur NFSv4 sont mémorisées sur plusieurs appareils de mémorisation et/ou à plus haut débit (par rapport à la capacité de débit du serveur) le résultat peut être des performances d'accès au fichier significativement meilleures. Les relations entre plusieurs clients, un seul serveur, et plusieurs appareils de mémorisation pour pNFS (serveur et clients ont accès à tous les appareils de mémorisation) sont montrées à la Figure 1.

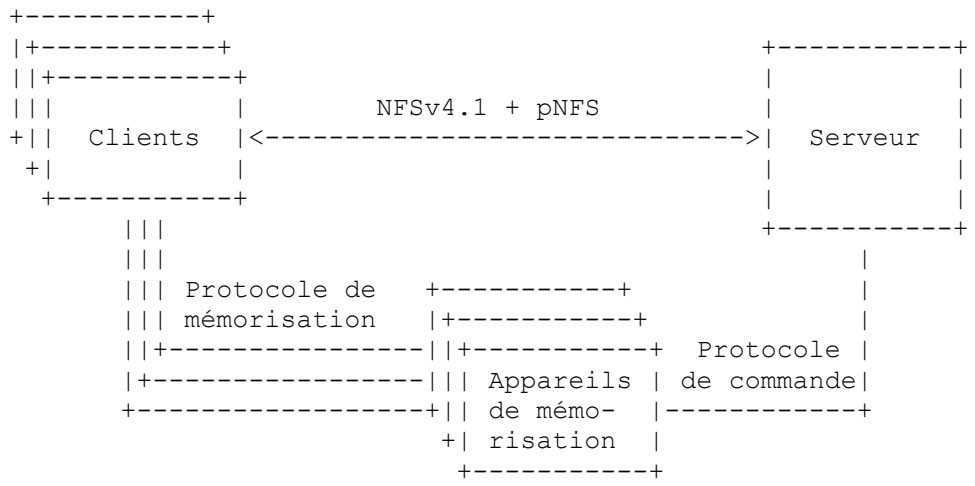


Figure 1

Dans ce modèle, les clients, le serveur, et les appareils de mémorisation sont responsables de la gestion de l'accès aux fichiers. Ceci est différent de NFSv4 sans pNFS, où c'est principalement la responsabilité du serveur ; une partie de cette responsabilité peut être déléguée au client dans des conditions strictement spécifiées. Voir au paragraphe 12.2.5 la discussion du protocole de mémorisation. Voir au paragraphe 12.2.6 la discussion du protocole de contrôle.

pNFS prend la forme d'opérations FACULTATIVES qui gèrent des objets de protocole appelés des "dispositions" (paragraphe 12.2.7) qui contiennent des informations de gamme d'octets et de localisation de mémorisation. La disposition est gérée d'une façon similaire aux délégations de données NFSv4.1. Par exemple, la disposition est prêtée, rappelable, et révocable. Cependant, les dispositions sont des abstractions distinctes et sont manipulées avec de nouvelles opérations. Quand un client détient une disposition, il lui est accordé la capacité d'accéder directement à la gamme d'octets à la localisation de mémorisation spécifiée dans la disposition.

Il y a des interactions entre les dispositions et d'autres abstractions NFSv4.1 comme les délégations de données et le verrouillage de gamme d'octets. Les questions de délégation sont discutées au paragraphe 12.5.5. Les questions de verrouillage de gamme d'octet sont discutées aux paragraphes 12.2.9 et 12.5.1.

12.2 Définitions de pNFS

La caractéristique pNFS de NFSv4.1 fournit l'accès en parallèle aux données d'un système de fichiers qui étale son contenu sur plusieurs serveurs de mémorisation. La première instanciation de pNFS, au titre de NFSv4.1, sépare le traitement du protocole de système de fichiers en deux parties : traitement de métadonnées et traitement des données. Les données consistent en le contenu des fichiers réguliers qui sont étalés sur des serveurs de mémorisation. L'étalement des données survient au moins de deux façons : fichier par fichier et, dans les fichiers suffisamment grands, bloc par bloc. À l'opposé, l'accès étalé aux métadonnées par les clients pNFS n'est pas fourni dans NFSv4.1, même si l'extrémité arrière du système de fichiers d'un serveur pNFS pourrait étaler les métadonnées. Les métadonnées consistent en tout le reste, incluant le contenu de fichiers non réguliers (par exemple, des répertoires) ; voir le paragraphe 12.2.1. La fonction des métadonnées est mise en œuvre par un serveur NFSv4.1 qui prend en charge pNFS et les opérations décrites à la Section 18 ; un tel serveur est appelé un serveur de métadonnées (paragraphe 12.2.2).

La fonction de données est mise en œuvre par un ou plusieurs appareils de mémorisation, auxquels le client accède via un protocole de mémorisation. Un sous ensemble (défini au paragraphe 13.6) de NFSv4.1 est un tel protocole de mémorisation. De nouveaux termes sont introduits dans la nomenclature de NFSv4.1 et les termes existants sont précisés pour permettre la description des caractéristiques de pNFS.

12.2.1 Métadonnées

Informations sur un objet de système de fichiers, telles que son nom, sa localisation dans l'espace de noms, le propriétaire, l'ACL, et d'autres attributs. Les métadonnées peuvent aussi inclure des informations de localisation de mémorisation, et cela va varier selon le mécanisme de mémorisation sous-jacent utilisé.

12.2.2 Serveur de métadonnées

Un serveur NFSv4.1 qui prend en charge la caractéristique pNFS. Divers choix d'architecture existent pour le serveur de métadonnées et son utilisation des informations de système de fichiers détenues par le serveur. Certains serveurs peuvent contenir des métadonnées seulement pour les objets de fichier résidants au serveur de métadonnées, tandis que les données de fichier résident sur les appareils de mémorisation associés. D'autres serveurs de métadonnées peuvent détenir les métadonnées et un degré variable de données de fichier.

12.2.3 Client pNFS

Un client NFSv4.1 qui prend en charge les opérations pNFS et prend en charge au moins un protocole de mémorisation pour effectuer l'entrée/sortie aux appareils de mémorisation.

12.2.4 Appareil de mémorisation

Un appareil de mémorisation mémorise les données d'un fichier régulier, mais laisse la gestion des métadonnées au serveur de métadonnées. Un appareil de mémorisation pourrait être un autre serveur NFSv4.1, un appareil de mémorisation fondé sur l'objet (OSD, *object-based storage device*) un appareil de bloc à accès par réseau à zone de mémorisation (SAN, *Storage Area Network*) (un SAN, par exemple, canal fibre ou iSCSI) ou quelque autre entité.

12.2.5 Protocole de mémorisation

Comme noté à la Figure 1, le protocole de mémorisation est la méthode utilisée par le client pour mémoriser et restituer directement les données des appareils de mémorisation.

La caractéristique pNFS de NFSv4.1 a été structurée de façon à permettre que divers protocoles de mémorisation soient définis et utilisés. Un exemple de protocole de mémorisation est NFSv4.1 lui-même (comme expliqué à la Section 13). D'autres options pour le protocole de mémorisation sont décrites ailleurs et incluent :

* Des protocoles de bloc/volume comme le SCSI Internet (iSCSI) [RFC7143] et FCP [FCP-2]. La prise en charge de protocoles de bloc/volume peut être indépendante de la structure d'adressage du protocole de bloc/volume utilisé, permettant plus d'un protocole pour accéder aux mêmes données de fichier et permettant l'extensibilité à d'autres protocoles de bloc/volume. Voir dans la [RFC5663] la spécification d'une disposition qui permet à pNFS d'utiliser des protocoles de mémorisation de bloc/volume.
* Des protocoles d'objet comme OSD sur iSCSI ou canal fibre [OSD]. Voir dans la [RFC5664] la spécification d'une disposition qui permet à pNFS d'utiliser des protocoles de mémorisation d'objet.

Il est possible que divers protocoles de mémorisation soient disponibles à la fois pour le client et le serveur et il est possible qu'un client et un serveur n'aient pas de protocole de mémorisation disponible en commun. À cause de cela, le serveur pNFS DOIT prendre en charge l'accès NFSv4.1 normal à tout fichier accessible par la caractéristique pNFS ; cela va permettre une interopérabilité continue entre un client et un serveur NFSv4.1.

12.2.6 Protocole de contrôle

Comme noté dans la Figure 1, le protocole de contrôle est utilisé par le système de fichiers exporté entre le serveur de métadonnées et les appareils de mémorisation. La spécification de ces protocoles sort du domaine d'application du protocole NFSv4.1. Ces protocoles de contrôle vont être utilisés pour contrôler des activités comme l'allocation et la désallocation de mémorisation, la gestion d'état requise des appareils de mémorisation pour effectuer le contrôle d'accès du client, et, selon le protocole de mémorisation, l'application de l'authentification et de l'autorisation afin que les restrictions qui seront appliquées par le serveur de métadonnées soient aussi appliquées par l'appareil de mémorisation.

Un protocole de contrôle particulier n'est pas EXIGÉ par NFSv4.1 mais des exigences portent sur le protocole de contrôle pour maintenir des attributs comme l'instant de modification, l'attribut de changement, et la position de fin de fichier (EOF,

end-of-file). Noter que si pNFS est mis en couches sur un système de fichiers en grappes, parallèle (par exemple, [PVFS]) les mécanismes qui permettent la mise en grappes et le parallélisme dans ce système de fichiers peuvent être considérés comme le protocole de contrôle.

12.2.7 Types de disposition

Une disposition décrit la transposition des données d'un fichier en les appareils de mémorisation qui détiennent les données. Une disposition est dite appartenir à un type de disposition spécifique (type de données layouttype4, voir le paragraphe 3.3.13). Le type de disposition permet que des variantes traitent différents protocoles de mémorisation, comme ceux associés aux types de disposition de bloc/volume [RFC5663], objet [RFC5664], et fichier (Section 13). Un serveur de métadonnées, avec son protocole de contrôle, DOIT prendre en charge au moins un type de disposition. Une sous gamme privée de l'espace de noms de type de disposition est aussi définie. Les valeurs provenant de la gamme de type de disposition privée PEUVENT être utilisée pour des essais ou expérimentation interne (voir le paragraphe 3.3.13).

Par exemple, l'organisation du type de disposition fichier pourrait être un dispositif de couples (par exemple, identifiant d'appareil, bride de fichier) avec une définition de comment les données sont mémorisées sur les appareils (par exemple, en bandes). Une disposition de bloc/volume pourrait être un dispositif de triplets qui mémorisent <identifiant d'appareil, numéro de bloc, compte de blocs> avec des informations sur la taille de bloc et le décalage de fichier associé au numéro de bloc. Une disposition d'objet pourrait être un dispositif de couples <identifiant d'appareil, identifiant d'objet> et une structure supplémentaire (c'est-à-dire, la carte d'agrégation) qui définisse comment la séquence d'octets logique des données du fichier est mise à la suite dans les différents objets. Noter que les dispositions réelles sont normalement plus complexes que ces simple exemples.

Les demandes pour les opérations en relation avec pNFS spécifient souvent un type de disposition. Des exemples de telles opérations sont GETDEVICEINFO et LAYOUTGET. La réponse à ces opérations va inclure des structures comme device_addr4 ou layout4, dont chacune inclut un type de disposition. Le type de disposition envoyé par le serveur DOIT toujours être le même que celui demandé par le client. Quand un serveur envoie une réponse qui inclut un type de disposition différent, le client DEVRAIT ignorer la réponse et se comporter comme si le serveur avait retourné une réponse d'erreur.

12.2.8 Disposition

Une disposition définit comment les données d'un fichier sont organisées sur un ou plusieurs appareils de mémorisation. Il y a de nombreux types de disposition potentiels ; chaque type de disposition est différencié par le protocole de mémorisation utilisé pour accéder aux données et par le schéma d'agrégation qui dispose les données du fichier sur les appareils de mémorisation sous-jacents. Une disposition est précisément identifiée par le quintuplet <identifiant de client, bride de fichier, type de disposition, iomode, gamme>, où bride de fichier se réfère à la bride de fichier du fichier sur le serveur de métadonnées.

Il est important de définir quand des dispositions se chevauchent et/ou entrent en conflit les unes avec les autres. Pour que deux dispositions avec des gammes d'octets en chevauchement se chevauchent réellement l'une l'autre, les deux dispositions doivent être du même type de disposition, correspondre à la même bride de fichier, et avoir le même mode d'entrée/sortie *(iomode)*. Les dispositions sont en conflit quand elles se chevauchent et diffèrent dans le contenu de la disposition (c'est-à-dire, les paramètres de transposition d'appareil de mémorisation en fichier diffèrent). Noter que des iomodes différents ne conduisent pas à des dispositions en conflit. Il est permis que des dispositions avec des iomodes différents, appartenant à la même gamme d'octets, soient détenues par le même client. Un exemple serait la fonction de copie sur écriture pour un type de disposition de bloc/volume.

12.2.9 Disposition Iomode

La disposition iomode (type de données layoutmode4, voir le paragraphe 3.3.20) indique au serveur de métadonnées l'intention du client d'effectuer juste des opérations READ ou un mélange d'opérations READ et WRITE. Pour certains types de disposition, il est utile à un client de spécifier cette intention au moment où il envoie LAYOUTGET (paragraphe 18.43). Par exemple, pour des protocoles fondés sur le bloc/volume, l'allocation des blocs pourrait avoir lieu quand un iomode LAYOUTIOMODE4_RW est spécifié. Un iomode spécial LAYOUTIOMODE4_ANY est défini et peut seulement être utilisé pour LAYOUTRETURN et CB_LAYOUTRECALL, mais pas pour LAYOUTGET. Il spécifie que des dispositions relevant des deux iomodes LAYOUTIOMODE4_READ et LAYOUTIOMODE4_RW sont retournées ou rappelée
s, respectivement.

Un appareil de mémorisation peut valider l'entrée/sortie à l'égard du iomode ; cela dépend de la mise en œuvre d'appareil de mémorisation et du type de disposition. Donc, si le iomode de disposition du client n'est pas cohérent avec les entrées/sorties effectuées, l'appareil de mémorisation peut rejeter les entrées/sorties du client avec une erreur indiquant qu'une nouvelle disposition avec le iomode correct devrait être obtenue via un LAYOUTGET. Par exemple, si un client obtient une disposition avec un iomode LAYOUTIOMODE4_READ et effectue un WRITE sur un appareil de mémorisation, il est permis à cet appareil de mémorisation de rejeter ce WRITE.

L'utilisation du iomode de disposition n'est pas en conflit avec les modes de partage OPEN ou les opérations LOCK de gamme d'octets ; les conflits de mode de partage d'ouverture et de verrou de gamme d'octets sont appliqués comme ils sont sans l'utilisation de pNFS et sont logiquement séparés du niveau de disposition pNFS. Les modes de partage d'ouverture et les verrous de gamme d'octets sont la méthode préférée pour restreindre l'accès de l'utilisateur aux fichiers de données. Par exemple, un OPEN de OPEN4_SHARE_ACCESS_WRITE n'est pas en conflit avec un LAYOUTGET contenant un iomode de LAYOUTIOMODE4_RW effectué par un autre client. Les applications qui dépendent d'une écriture concurrente dans le même fichier peuvent utiliser le verrouillage de gamme d'octets pour mettre en série leurs accès.

12.2.10 Identifiants d'appareil

L'identifiant d'appareil (type de données deviceid4, voir le paragraphe 3.3.14) identifie un groupe d'appareils de mémorisation. La portée d'un identifiant d'appareil est la paire <identifiant de client, type de disposition>. En pratique, une quantité significative d'informations peut être requise pour adresser pleinement un appareil de mémorisation. Plutôt que d'incorporer toutes ces informations dans une disposition, les dispositions incorporent les identifiants d'appareil. L'opération GETDEVICEINFO (paragraphe 18.40) de NFSv4.1 est utilisée pour restituer les informations d'adresse complètes (incluant toutes les adresses d'appareil pour l'identifiant d'appareil) concernant l'appareil de mémorisation en accord avec son type de disposition et identifiant d'appareil. Par exemple, l'adresse d'un serveur de données NFSv4.1 ou d'un appareil de mémorisation fondé sur l'objet pourrait être une adresse et accès IP. L'adresse d'un appareil de mémorisation de bloc pourrait être une étiquette de volume.

Les clients ne peuvent pas s'attendre à ce que la transposition entre un identifiant d'appareil et la ou les adresses de l'appareil de mémorisation persistent à travers le redémarrage d'un serveur de métadonnées. Voir au paragraphe 12.7.4 la description de comment fonctionne la récupération dans cette situation.

Un identifiant d'appareil vit tant qu'il y a une disposition qui se réfère à l'identifiant d'appareil. Si il n'y a pas de disposition se référant à l'identifiant d'appareil, le serveur est libre de supprimer à tout moment l'identifiant d'appareil. Une fois qu'un identifiant d'appareil est supprimé par le serveur, le serveur NE DOIT PAS réutiliser l'identifiant d'appareil pour le même type de disposition et identifiant de client. Cette exigence est faisable parce que l'identifiant d'appareil fait 16 octets, laissant suffisamment de place pour mémoriser un numéro de génération si la mise en œuvre de serveur exige de réutiliser la plus grande partie du reste du contenu de l'identifiant d'appareil. Cette exigence est nécessaire parce que autrement les conditions de compétition entre notification asynchrone d'ajout et de suppression d'identifiant d'appareil seraient trop difficiles à trier.

Les transpositions d'identifiant d'appareil en adresse d'appareil ne sont pas des prêts, et peuvent être changées à tout moment. (Noter qu'alors que des transpositions d'identifiant d'appareil en adresse d'appareil vont probablement changer après le redémarrage du serveur de métadonnées, le serveur n'est pas obligé de changer les transpositions.) Un serveur a un choix pour changer les transpositions. Il peut rappeler toutes les dispositions se référant à l'identifiant d'appareil ou il peut utiliser un mécanisme de notification.

Le protocole NFSv4.1 n'a pas de façon optimale de rappeler toutes les dispositions qui se réfèrent à un identifiant d'appareil particulier (sauf si le serveur associe un seul identifiant d'appareil avec un seul identifiant de système de fichier ou à un seul identifiant de client ; dans ce cas, CB_LAYOUTRECALL a des options pour rappeler toutes les dispositions associées au fsid, à la paire d'identifiants de client, ou juste à l'identifiant de client).

Via un mécanisme de notification (voir le paragraphe 20.12) les transpositions d'identifiant d'appareil à adresse d'appareil peuvent changer pendant la durée de fonctionnement du serveur sans rappeler ou révoquer les dispositions qui se réfèrent à l'identifiant d'appareil. Le mécanisme de notification peut aussi supprimer un identifiant d'appareil, mais seulement si le client n'a pas de dispositions se référant à l'identifiant d'appareil. Une notification de changement à une transposition d'identifiant d'appareil en adresse d'appareil va immédiatement ou finalement invalider une partie ou toutes les transpositions d'identifiant d'appareil. Le serveur DOIT prendre en charge les notifications et le client doit les demander avant qu'elles puissent être utilisées. Pour plus d'information s sur les types de notification, voir le paragraphe 20.12.

12.3 Opérations de pNFS

NFSv4.1 a plusieurs opérations qui sont nécessaires pour les serveurs pNFS, sans considération du type de disposition ou du protocole de mémorisation. Ces opérations sont toutes envoyées à un serveur de métadonnées et résumées ici. Bien que pNFS soit une caractéristique FACULTATIVE, si pNFS est mis en œuvre, certaines opérations sont EXIGÉES afin de s'y conformer. Voir la Section 17.

Voici les opérations pNFS de canal avant :

GETDEVICEINFO (paragraphe 18.40) comme noté précédemment (paragraphe 12.2.10) retourne la transposition d'identifiant d'appareil en adresse d'appareil de mémorisation.

GETDEVICELIST (paragraphe 18.41) permet aux clients d'aller chercher tous les identifiants d'appareil pour un système de fichiers spécifique.

LAYOUTGET (paragraphe 18.43) est utilisé par un client pour obtenir une disposition pour un fichier.

LAYOUTCOMMIT (paragraphe 18.42) est utilisé pour informer le serveur de métadonnées de l'intention du client d'engager des données qui ont été écrites sur l'appareil de mémorisation (l'appareil de mémorisation originellement indiqué dans la valeur de retour de LAYOUTGET).

LAYOUTRETURN (paragraphe 18.44) est utilisé pour retourner des dispositions pour un fichier, un identifiant de système de fichiers (FSID) ou un identifiant de client.

Voici les opérations pNFS de canal de retour :

CB_LAYOUTRECALL (paragraphe 20.3) rappelle une disposition, toutes les dispositions appartenant à un système de fichiers, ou toutes les dispositions appartenant à un identifiant de client.

CB_RECALL_ANY (paragraphe 20.6) dit à un client qu'il doit retourner un certain nombre d'objets rappelables, incluant des dispositions, au serveur de métadonnées.

CB_RECALLABLE_OBJ_AVAIL (paragraphe 20.7) dit à un client qu'un objet rappelable qui lui a été refusé (dans le cas de pNFS, une disposition refusée par LAYOUTGET) du fait d'un épuisement des ressources, est maintenant disponible.

CB_NOTIFY_DEVICEID (paragraphe 20.12) notifie au client des changements d'identifiants d'appareil.

12.4 Attributs de pNFS

Un certain nombre d'attributs spécifiques de pNFS sont énumérés et décrits au paragraphe 5.12.

12.5 Sémantique de disposition

12.5.1 Garanties fournies par les dispositions

Les dispositions accordent au client la capacité d'accéder aux données situées à un appareil de mémorisation avec le protocole de mémorisation approprié. Il est garanti au client que la disposition sera rappelée quand une des deux choses suivantes se produit : soit une disposition en conflit est demandée, soit l'état encapsulé par la disposition devient invalide (cela peut arriver quand un événement modifie directement ou indirectement la disposition). Quand une disposition est rappelée et retournée par le client, le client continue avec la capacité d'accéder aux données du fichier avec les opérations normales de NFSv4.1 à travers le serveur de métadonnées. Seule la capacité d'accéder aux appareils de mémorisation est affectée.

L'exigence de NFSv4.1 que tous les droits d'accès d'utilisateur DOIVENT être obtenus par les opérations OPEN, LOCK, et ACCESS appropriées n'est pas modifiée par l'existence de dispositions. Les dispositions sont fournies aux clients NFSv4.1, et l'accès d'utilisateur suit toujours les règles du protocole comme si elles n'existaient pas. Il est exigé que pour qu'un client accède à un appareil de mémorisation, une disposition doit être détenue par le client. Si un appareil de mémorisation reçoit une demande d'entrée/sortie pour une gamme d'octets pour laquelle le client ne détient pas une disposition, l'appareil de mémorisation DEVRAIT rejeter cette demande d'entrée/sortie. Noter que l'acte de modifier un fichier pour lequel une disposition est détenue n'est pas nécessairement en conflit avec la détention de la disposition qui décrit le fichier modifié.

Donc, c'est l'exigence du protocole de mémorisation ou du type de disposition qui détermine le comportement nécessaire. Par exemple, les types de disposition de bloc/volume exigent que le iomode de la disposition s'accorde avec le type d'entrées/sorties effectuées.

Selon le type de disposition et le protocole de mémorisation utilisés, les permissions d'accès de l'appareil de mémorisation peuvent être accordées par un LAYOUTGET et peuvent être codées au sein d'une disposition spécifique du type. Pour un exemple de permissions d'accès de l'appareil de mémorisation, voir un protocole fondé sur l'objet comme [OSD]. Si les permissions d'accès sont codées dans la disposition, le serveur de métadonnées DEVRAIT rappeler la disposition quand ces permissions deviennent invalides pour une raison quelconque -- par exemple, quand un fichier devient non écrivable ou inaccessible à un client. Noter qu'il est quand même exigé des clients qu'ils effectuent les opérations OPEN, LOCK, et ACCESS appropriées comme décrit ci-dessus. Le degré auquel il est possible au client de circonvenir ces opérations et les conséquences de cela doivent être clairement spécifiées par les spécifications du type de disposition individuel. De plus, ces spécifications doivent être claires sur les exigences et non exigences pour les vérifications effectuées par le serveur.

En présence de la fonctionnalité pNFS, les verrous de gamme d'octets obligatoires DOIVENT se comporter comme ils le feraient sans pNFS. Donc, si des verrous et dispositions de fichiers obligatoires sont fournis simultanément, l'appareil de mémorisation DOIT être capable d'appliquer les verrous de gamme d'octets obligatoires. Par exemple, si un client obtient un verrou de gamme d'octets obligatoire et si un second client accède à l'appareil de mémorisation, l'appareil de mémorisation DOIT restreindre de façon appropriée les entrées/sorties pour la gamme de verrous de gamme d'octets obligatoires. Si l'appareil de mémorisation est incapable de fournir cette vérification en présence de verrous de gamme d'octets obligatoires, alors le serveur de métadonnées NE DOIT PAS accorder simultanément de dispositions et verrous de gamme d'octets obligatoires.

12.5.2 Obtention d'une disposition

Un client obtient une disposition avec l'opération LAYOUTGET. Le serveur de métadonnées va accorder des dispositions d'un type particulier (par exemple, bloc/volume, objet, ou fichier). Le client choisit un type approprié de disposition que le serveur prend en charge et que le client est prêt à utiliser. La disposition retournée au client pourrait ne pas correspondre exactement à la gamme d'octets demandée, comme décrit au paragraphe 18.43.3. Si nécessaire, un client peut envoyer plusieurs opérations LAYOUTGET ; il pourrait en résulter plusieurs dispositions en chevauchement, non conflictuelles (voir le paragraphe 12.2.8).

Afin d'obtenir une disposition, le client doit d'abord avoir ouvert le fichier via l'opération OPEN. Quand un client n'a pas la disposition d'un fichier, il DOIT présenter un identifiant d'état d'ouverture, une délégation d'identifiant d'état OPEN, ou un identifiant d'état de verrou de gamme d'octets dans l'argument loga_stateid. Un résultat réussi de LAYOUTGET inclut un identifiant d'état de disposition. Le premier LAYOUTGET réussi traité par le serveur en utilisant un identifiant d'état non de disposition comme argument DOIT avoir le champ "seqid" de l'identifiant d'état de la disposition dans la réponse réglé à un. Ensuite, le client DOIT utiliser un identifiant d'état de disposition (voir le paragraphe 12.5.3) sur les futures invocations de LAYOUTGET sur le fichier, et le "seqid" NE DOIT PAS être réglé à zéro. Une fois que la disposition a été restituée, elle peut être détenue à travers plusieurs séquences de OPEN et CLOSE. Donc, un client peut détenir une disposition pour un fichier qui n'est pas ouvert actuellement par un utilisateur sur le client. Cela permet la mise en antémémoire de dispositions au delà de CLOSE.

Le protocole de mémorisation utilisé par le client pour accéder aux données sur l'appareil de mémorisation est déterminé par le type de disposition. Le client est responsable de faire correspondre le type de disposition avec une méthode disponible pour interpréter et utiliser la disposition. La méthode pour le choix de ce type de disposition sort du domaine d'application de la fonction pNFS.

Bien que le serveur de métadonnées ait le contrôle de la disposition pour un fichier, le client pNFS peut fournir des indications au serveur quand un fichier est ouvert ou créé sur le type préféré de disposition et de schémas d'agrégation. pNFS introduit un attribut layout_hint (paragraphe 5.12.4) que le client peut établir au moment de la création du fichier pour fournir au serveur une indication sur les nouveaux fichiers. Établir cet attribut séparément, après la création du fichier pourrait rendre difficile, ou impossible, que la mise en œuvre de serveur s'y conforme.

Parce que le createmode4 EXCLUSIVE4 ne permet pas de réglage des attributs moment de la créatio du fichier, NFSv4.1 introduit le createmode4 EXCLUSIVE4_1, qui permet bien l'établissement des attributs au moment de la création du fichier. De plus, si la session est créé avec des antémémoires de réponse persistantes, EXCLUSIVE4_1 n'est ni nécessaire ni permis. À la place, GUARDED4 fonctionne mieux et est prescrit. Le Tableau 18 au paragraphe 18.16.3 résume comment il est permis à un client d'envoyer une création exclusive.

12.5.3 Identifiant d'état de disposition

Comme avec tous les autres identifiants d'état, l'identifiant d'état de disposition consiste en les champs "seqid" et "autre". Une fois qu'un identifiant d'état de disposition est établi, le champ "autre" va rester constant sauf si l'identifiant d'état est révoqué ou si le client retourne toutes les dispositions sur le fichier et si le serveur dispose de l'identifiant d'état. Le champ "seqid" est initialement réglé à un, et n'est jamais zéro sur une opération NFSv4.1 qui utilise des identifiants d'état de disposition, si c'est une opération de canal avant ou de canal de retour. Après l'établissement de l'identifiant d'état de disposition, le serveur incrémente de un la valeur du "seqid" dans chaque réponse LAYOUTGET et LAYOUTRETURN suivante, et dans chaque demande CB_LAYOUTRECALL.

Le but de conception de pNFS étant de fournir le parallélisme, l'identifiant d'état de disposition diffère des autres types d'identifiant d'état en ce que le client est supposé envoyer les opérations LAYOUTGET et LAYOUTRETURN en parallèle. La valeur de "seqid" est utilisée par le client pour trier correctement les réponses à LAYOUTGET et LAYOUTRETURN. Le "seqid" est aussi utilisé pour prévenir des conditions de concurrence entre LAYOUTGET et CB_LAYOUTRECALL. Étant donné que les règles de traitement diffèrent pour les identifiants d'état de disposition des autres types d'identifiant d'état, seules les sections pNFS du présent document devrait être considérées déterminer le traitement approprié d'identifiant d'état de disposition.

Une fois que le client a reçu un identifiant d'état de disposition, il DOIT utiliser le "seqid" correct pour les opérations LAYOUTGET ou LAYOUTRETURN suivantes. Le "seqid" correct est défini comme la plus forte valeur de "seqid" provenant des réponses des opérations LAYOUTGET ou LAYOUTRETURN pleinement traitées ou des arguments d'une opération CB_LAYOUTRECALL pleinement traitée. Comme le serveur incrémente la valeur de "seqid" sur chaque opération de disposition, le client peut déterminer l'ordre de traitement des opérations en inspectant la valeur de "seqid". Dans le cas de chevauchement de gammes de dispositions, les informations d'ordre vont fournir au client la connaissance des gammes de dispositions qui sont détenues. Noter que le chevauchement de gammes de disposition peut survenir parce que le client a des demandes spécifiques ou parce que il est permis au serveur d'étendre la gamme d'une disposition demandée et de le notifier au client dans le résultat du LAYOUTRETURN. Des exigences supplémentaires de séquençage d'identifiant d'état de disposition sont fournies au paragraphe 12.5.5.2.

La réception par le client d'un "seqid" n'est pas suffisante pour l'utilisation ultérieure. Le client doit pleinement traiter les opérations avant que le "seqid" puisse être utilisé. Pour les résultats de LAYOUTGET, si le client n'utilise pas le modèle oublieux (paragraphe 12.5.5.1) il DOIT d'abord mettre à jour son enregistrement des gammes de disposition du fichier qu'il a avant d'utiliser le seqid. Pour les résultats de LAYOUTRETURN, le client DOIT supprimer la gamme de son enregistrement des gammes de disposition du fichier qu'il avait avant d'utiliser le seqid. Pour les arguments CB_LAYOUTRECALL, le client DOIT envoyer une réponse au rappel avant d'utiliser le seqid. L'exigence fondamentale du traitement du client est que le "seqid" soit utilisé pour fournir l'ordre de traitement. Les résultats de LAYOUTGET peuvent être traités en parallèle. Les résultats de LAYOUTRETURN peuvent être traités en parallèle. Les réponses de LAYOUTGET et LAYOUTRETURN peuvent être traitées en parallèle tant que les gammes ne se chevauchent pas. Le traitement de la demande CB_LAYOUTRECALL DOIT toujours être effectué dans l'ordre de "seqid" .

Une fois qu'un client n'a plus de dispositions sur un fichier, l'identifiant d'état de disposition n'est plus valide et NE DOIT PAS être utilisé. Toute tentative d'utiliser un tel identifiant d'état de disposition va résulter en NFS4ERR_BAD_STATEID.

12.5.4 Engagement d'une disposition

Permettant diverses capacités de protocole de mémorisation, le protocole pNFS n'exige pas du serveur de métadonnées ni des appareils de mémorisation d'avoir une vue cohérente des transpositions d'attributs de fichier et de localisation de données. La transposition de localisation de données se réfère à des aspects comme à quels décalages mémoriser des données par opposition à la mémorisation de trous (voir la discussion du paragraphe 13.4.4). Des questions en relation se posent pour les protocoles de mémorisation où une disposition peut détenir des blocs alloués de façon provisoire lorsque l'allocation de ces blocs ne survit pas à un redémarrage complet du client et du serveur. À cause de cette imprécision, il est nécessaire de resynchroniser le client avec le serveur de métadonnées et ses appareils de mémorisation et rendre tout changement potentiel disponible aux autres clients. Ceci est réalisé en utilisant l'opération LAYOUTCOMMIT.

L'opération LAYOUTCOMMIT est chargée d'engager une disposition modifiée au serveur de métadonnées. Les données devraient être écrites et engagées aux appareils de mémorisation appropriés avant que le LAYOUTCOMMIT se produise. La portée de l'opération LAYOUTCOMMIT dépend du protocole de mémorisation utilisé. Il est important de noter que le niveau de synchronisation est du point de vue du client qui a envoyé le LAYOUTCOMMIT. L'état mis à jour sur le serveur de métadonnées doit seulement refléter l'état lors de la dernière opération du client précédant le LAYOUTCOMMIT. Le serveur de métadonnées n'est pas obligé de maintenir une vue globale tenant compte des entrées/sorties des autres clients qui pourraient s'être produites dans le même temps.

Pour les dispositions fondées sur le bloc/volume, LAYOUTCOMMIT peut exiger de mettre à jour le liste de blocs qui composent le fichier et d'engager cette disposition dans une mémorisation stable. Pour les dispositions fondées sur le fichier, la synchronisation des attributs entre les métadonnées et les appareils de mémorisation, principalement de l'attribut de taille, est requise.

Le protocole de contrôle est libre de synchroniser les attributs avant qu'il reçoive un LAYOUTCOMMIT ; cependant, à l'achèvement réussi d'un LAYOUTCOMMIT, l'état qui existe sur le serveur de métadonnées qui décrit le fichier DOIT être synchronisé avec l'état qui existe sur les appareils de mémorisation qui comportent ce fichier comme de la dernière opération envoyée du client. Donc, un client qui interroge la taille d'un fichier entre un WRITE à un appareil de mémorisation et le LAYOUTCOMMIT pourrait observer une taille qui ne reflète pas les données écrites réelles.

Le client DOIT avoir une disposition afin d'envoyer une opération LAYOUTCOMMIT.

12.5.4.1 LAYOUTCOMMIT et change/time_modify

Les attributs change et time_modify peuvent être mis à jour par le serveur quand l'opération LAYOUTCOMMIT est traitée. La raison en est que certains types de disposition ne prennent pas en charge la mise à jour de ces attributs quand les appareils de mémorisation traitent les opérations d'entrée/sortie. Si un client a une disposition avec le iomode LAYOUTIOMODE4_RW sur le fichier, le client PEUT fournir une valeur suggérée au serveur pour time_modify dans les arguments de LAYOUTCOMMIT. Sur la base du type de disposition, la valeur fournie peut ou non être utilisée. Le serveur devrait faire une vérification de bonne santé des valeurs fournies par le client avant qu'elles soient utilisées. Par exemple, le serveur devrait s'assurer que le temps ne s'écoule pas à contre sens. Le client a toujours l'option de régler time_modify par une opération SETATTR explicite.

Pour certains protocoles de disposition, l'appareil de mémorisation est capable de notifier au serveur de métadonnées l'occurrence d'une entrée/sortie ; par suite, les attributs change et time_modify peuvent être mis à jour au serveur de métadonnées. Pour un serveur de métadonnées qui est capable de surveiller les mises à jour des attributs change et time_modify, le traitement de LAYOUTCOMMIT n'est pas obligé de mettre à jour l'attribut change. Dans ce cas, le serveur de métadonnées doit s'assurer qu'aucune autre mise à jour des données ne s'est produite depuis la dernière mise à jour des attributs ; les protocoles fondés sur le fichier peuvent avoir assez d'informations pour cette détermination ou peuvent mettre à jour l'attribut change à chaque modification de fichier. Cela s'applique aussi pour l'attribut time_modify. Si la mise en œuvre de serveur est capable de déterminer que le fichier n'a pas été modifié depuis la dernière mise à jour de time_modify, le serveur n'a pas besoin de mettre à jour time_modify à LAYOUTCOMMIT. À l'achèvement de LAYOUTCOMMIT, les attributs mis à jour devraient être visibles si ce fichier a été modifié depuis le dernier LAYOUTCOMMIT ou LAYOUTGET précédent.

12.5.4.2 LAYOUTCOMMIT et size

La taille d'un fichier peut être mise à jour quand l'opération LAYOUTCOMMIT est utilisée par le client. Un des champs dans l'argument à LAYOUTCOMMIT est loca_last_write_offset ; ce champ indique le plus fort décalage d'octets écrits mais non encore engagés avec l'opération LAYOUTCOMMIT. Le type de données de loca_last_write_offset est newoffset4 et est commuté sur une valeur booléenne, no_newoffset, qui indique si une écriture précédente s'est produite ou non. Si no_newoffset est FAUX, un décalage n'est pas donné. Si le client a une disposition avec l'iomode LAYOUTIOMODE4_RW sur le fichier, avec une gamme d'octets (notée par les valeurs de lo_offset et lo_length) qui chevauchent loca_last_write_offset, alors le client PEUT régler no_newoffset à VRAI et fournir un décalage qui va mettre à jour la taille du fichier. On se souviendra que le décalage n'est pas la même chose que la longueur bien qu'ils soient en relation. Par exemple, une valeur de loca_last_write_offset de zéro signifie que un octet a été écrit au décalage zéro, et donc que la longueur du fichier est d'au moins un octet.

Le serveur de métadonnées peut faire une des choses suivantes :

1. Mettre à jour la taille de fichier en utilisant le dernier décalage d'écriture fourni par le client comme vraie taille du fichier ou comme une indication de la taille du fichier. Si le serveur de métadonnées a une méthode disponible, toute nouvelle valeur pour la taille du fichier devrait faire l'objet d'une vérification de bonne santé. Par exemple, le fichier ne doit pas être tronqué si le client présente un décalage de dernière écriture de moins que la taille courante du fichier.

2. Ignorer le décalage de dernière écriture fourni par le client ; le serveur de métadonnées doit avoir une connaissance suffisante provenant d'autres sources pour déterminer la taille du fichier. Par exemple, le serveur de métadonnées interroge les appareils de mémorisation avec le protocole de contrôle.

La méthode choisie pour mettre à jour la taille du fichier va dépendre des capacités de l'appareil de mémorisation et/ou du protocole de contrôle. Par exemple, si les appareils de mémorisation sont des appareils de blocs sans connaissance de la taille de fichier, le serveur de métadonnées doit s'appuyer sur le client pour régler de façon appropriée le dernier décalage d'écriture.

Les résultats de LAYOUTCOMMIT contiennent une nouvelle valeur de taille sous la forme d'un type de données d'union newsize4. Si la taille du fichier est établie par suite de LAYOUTCOMMIT, le serveur de métadonnées doit répondre avec la nouvelle taille ; autrement, la nouvelle taille n'est pas fournie. Si la taille du fichier est mise à jour, le serveur de métadonnées DEVRAIT mettre à jour les appareils de mémorisation de telle façon que la nouvelle taille de fichier soit reflétée quand le traitement de LAYOUTCOMMIT est achevé. Par exemple, le client devrait être capable de lire jusqu'à la nouvelle taille de fichier.

Le client peut étendre la longueur d'un fichier ou tronquer un fichier en envoyant une opération SETATTR au serveur de métadonnées avec l'attribut de taille spécifié. Si la taille spécifiée est supérieure à la taille du fichier courante, le fichier est "étendu de zéros", c'est-à-dire, des zéros sont implicitement ajoutés entre la fin de fichier (EOF) du fichier précédent et le nouveau EOF. (Dans de nombreuses mises en œuvre, la gamme d'octets étendue de zéros du fichier consiste en trous non alloués dans le fichier.) Quand le client écrit après le EOF via WRITE, l'opération SETATTR n'a pas besoin d'être utilisée.

12.5.4.3 LAYOUTCOMMIT et mise à jour de disposition

L'argument LAYOUTCOMMIT contient un champ loca_layoutupdate (paragraphe 18.42.1) de type de données layoutupdate4 (paragraphe 3.3.18). Cet argument est une structure spécifique du type de disposition. La structure peut être utilisée pour passer des informations arbitraires spécifiques du type de disposition du client au serveur de métadonnées à l'instant LAYOUTCOMMIT. Par exemple, si il utilise une disposition de bloc/volume, le client peut indiquer au serveur de métadonnées quels blocs réservés ou alloués le client a ou n'a pas utilisés. Le contenu de loca_layoutupdate (champ lou_body) n'a pas besoin d'être le même contenu spécifique du type de disposition retourné par LAYOUTGET (paragraphe 18.43.2) dans le champ loc_body du champ lo_content du champ logr_layout. Le contenu de loca_layoutpdate est défini par la spécification de type de disposition et est opaque pour LAYOUTCOMMIT.

12.5.5 Rappel d'une disposition

Comme une disposition protège l'accès d'un client à un fichier via un chemin direct du client à l'appareil de mémorisation, une disposition a seulement besoin d'être rappelée quand elle est sémantiquement incapable de servir à cette fonction. Normalement, cela arrive quand la disposition n'encapsule plus la vraie localisation du fichier sur la gamme d'octets qu'elle représente. Toute opération ou action, comme un re-marquage ou un équilibrage de charge piloté par le serveur, qui change la disposition va résulter en un rappel de la disposition. Une disposition est rappelée par l'opération de rappel CB_LAYOUTRECALL (voir le paragraphe 20.3) et retournée avec LAYOUTRETURN (voir le paragraphe 18.44). L'opération CB_LAYOUTRECALL peut rappeler une disposition identifiée par une gamme d'octets, toutes les dispositions associées à un identifiant de système de fichiers (FSID) ou toutes les dispositions associées à un identifiant de client. Le paragraphe 12.5.5.2 discute les questions de séquençage autour de l'obtention, du retour, et du rappel de dispositions.

Un iomode est aussi spécifié quand on rappelle une disposition. Généralement, le iomode dans la demande de rappel doit correspondre à la disposition retournée ; par exemple, un rappel avec un iomode de LAYOUTIOMODE4_RW devrait causer le retour par le client de seulement des dispositions LAYOUTIOMODE4_RW et non LAYOUTIOMODE4_READ. Cependant, une énumération spéciale LAYOUTIOMODE4_ANY est définie pour permettre de rappeler une disposition de tout iomode ; autrement dit, le client doit retourner les deux dispositions LAYOUTIOMODE4_READ et LAYOUTIOMODE4_RW.

Une opération REMOVE DEVRAIT causer le rappel par le serveur de métadonnées de la disposition pour empêcher le client d'accéder à un fichier non existant et de réclamer l'état mémorisé sur le client. Comme un REMOVE peut être retardé jusqu'à ce que la dernière clôture du fichier se soit produite, le rappel peut aussi être retardé jusqu'à ce moment. Après la libération de la dernière référence sur le fichier et la suppression du fichier, le client devrait ne plus être capable d'effectuer d'entrée/sortie en utilisant la disposition. Dans le cas d'une disposition fondée sur le fichier, le serveur de données DEVRAIT retourner NFS4ERR_STALE en réponse à toute opération sur le fichier supprimé.

Une fois qu'une disposition a été retournée, le client NE DOIT PAS envoyer d'entrées/sorties aux appareils de mémorisation pour le fichier, gamme d'octets, et iomode représentés par la disposition retournée. Si un client envoie une entrée/sortie à un appareil de mémorisation pour lequel il ne détient pas une disposition, l'appareil de mémorisation DEVRAIT rejeter l'entrée/sortie.

Bien que pNFS n'altère pas les capacités de mise en antémémoire des données de fichier des clients, ni leur sémantique, il reconnaît que certains clients peuvent effectuer une mise en antémémoire plus agressive d'écriture derrière pour optimiser les avantages fournis par pNFS. Cependant, la mise en antémémoire d'écriture derrière peut avoir des effets contraires sur la latence en retournant une disposition en réponse à un CB_LAYOUTRECALL ; ceci est similaire à la délégation de fichier et à l'impact que la mise en antémémoire de données de fichier a sur DELEGRETURN. Les mises en œuvre de client DEVRAIENT limiter la quantité de données non écrites qu'elles ont en instance à tout moment afin d'empêcher des réponses excessivement longues à CB_LAYOUTRECALL. Une fois qu'une disposition est rappelée, un serveur DOIT attendre une période de prêt avant d'effectuer une autre action. Aussitôt qu'une période de prêt est passée, le serveur peut choisir de bloquer l'accès du client aux appareils de mémorisation si le serveur perçoit que le client a pris trop de temps pour retourner une disposition. Cependant, juste comme dans le cas de délégation de données et de DELEGRETURN, le serveur peut choisir d'attendre, étant donné que le client montre des progrès sur la voie d'un retour d'une disposition. Ces progrès peuvent prendre la forme d'une interaction réussie avec les appareils de mémorisation ou de portions de la disposition retournées par le client. Le serveur peut aussi limiter l'exposition de ces problèmes en limitant les gammes d'octets fournies initialement dans les dispositions et donc la quantité de données modifiées en instance.

12.5.5.1 Robustesse du rappel de disposition

On a supposé jusqu'ici que l'état du client pNFS (gammes de disposition et iomode) pour un fichier correspond exactement à celui du serveur pNFS pour ce fichier. Cette hypothèse conduit à l'implication que tout rappel résulte en un LAYOUTRETURN ou ensemble de LAYOUTRETURN qui correspond exactement à la gamme dans le rappel, car le client et le serveur s'accordent tous deux sur l'état à maintenir. Cependant, il peut être utile que cette hypothèse ne tienne pas toujours. Par exemple :

* Si les conflits qui exigent des rappels sont très rares, et si un serveur peut utiliser un rappel multi-fichiers pour récupérer des ressources par client (par exemple, via un rappel de FSID ou un rappel multi-fichiers au sein d'un seul CB_COMPOUND) le résultat peut être significativement moins de trafic pNFS client-serveur.

* Il peut être utile aux serveurs de conserver des informations sur les gammes détenues par un client sur une base grossière, conduisant à ce que les gammes de disposition du serveur soient au delà de celles réellement détenues par le client. À la limite, un serveur pourrait gérer des conflits sur la base du fichier, envoyant seulement des rappels sur tout un fichier même si les clients peuvent demander et avoir accordées des gammes de sous fichiers.

* Il peut être utile aux clients "d'oublier" les détails de quelles dispositions et gammes le client a réellement, conduisant à ce que les gammes de disposition du serveur soient au delà de ce que le client "pense" qu'il a. Tant que le client ne suppose pas qu'il a des dispositions qui vont au delà de ce que le serveur a accordé, c'est une pratique sûre. Quand un client oublie quelles gammes et dispositions il a, et qu'il reçoit une opération CB_LAYOUTRECALL, le client DOIT suivre avec un LAYOUTRETURN pour ce que le serveur a rappelé, ou autrement retourner l'erreur NFS4ERR_NOMATCHING_LAYOUT si il n'a pas de disposition à retourner dans la gamme rappelée.

* Afin d'éviter des erreurs, il est vital qu'un client n'alloue pas lui-même des permissions de disposition au delà de ce que le serveur a accordé, et que le serveur n'oublie pas les permissions de disposition qui ont été accordées. Par ailleurs, si un serveur estime qu'un client détient une disposition dont le client n'a pas connaissance, il est utile que le client indique proprement l'achèvement du rappel demandé soit en envoyant une opération LAYOUTRETURN pour la gamme entière demandée, soit en retournant une erreur NFS4ERR_NOMATCHING_LAYOUT au CB_LAYOUTRECALL.

Donc, à la lumière de ceci, il est utile pour un serveur d'être capable d'envoyer des rappels pour des gammes de disposition qu'il n'a pas accordées à un client, et pour un client de retourner des gammes qu'il ne détient pas. Un client pNFS DOIT toujours retourner les dispositions qui comprennent la gamme complète spécifiée par le rappel. Noter que la pleine gamme de dispositions rappelée n'a pas besoin d'être retournée au titre d'une seule opération, mais peut être retournée en portions. Cela permet au client de programmer la purge des résidus de données et d'engager et retourner des dispositions. Aussi, cela indique au serveur de métadonnées que le client fait des progrès.

Quand une disposition est retournée, le client NE DOIT PAS avoir en instance de demandes d'entrée/sortie aux appareils de mémorisation impliqués dans la disposition. Autrement dit, le client NE DOIT PAS retourner la disposition pendant qu'il a en instance des demandes d'entrée/sortie à l'appareil de mémorisation.

Même avec cette exigence pour le client, il est possible que des demandes d'entrée/sortie puissent être présentées à un appareil de mémorisation qui n'est plus autorisé à les effectuer. Comme le serveur n'a pas un contrôle strict sur quand le client va retourner la disposition, le serveur peut ultérieurement décider unilatéralement de révoquer l'accès du client aux appareils de mémorisation comme fourni par la disposition. En choisissant de révoquer l'accès, le serveur doit traiter la possibilité de demandes d'entrée/sortie en retard, c'est-à-dire, des demandes d'entrée/sortie qui sont encore en cours pour les

appareils de mémorisation identifiés par la disposition révoquée. Toutes les spécifications de type de disposition DOIVENT définir si une révocation unilatérale de disposition par le serveur de métadonnées est prise en charge ; si elle l'est, la spécification doit aussi décrire comment sont traitées les écritures en retard. Par exemple, les appareils de mémorisation identifiés par la disposition révoquée pourraient être bloqués au client qui détient la disposition.

Afin d'assurer la convergence client/serveur à l'égard de l'état de disposition, l'opération LAYOUTRETURN finale dans une séquence d'opérations LAYOUTRETURN pour un rappel particulier DOIT spécifier la gamme entière rappelée, faisant écho au type de disposition rappelée, à l'iomode, au type de rappel/retour (FILE, FSID, ou ALL) et à la gamme d'octets, même si des dispositions relevant de gammes partielles étaient précédemment retournées. De plus, si le client détient des non dispositions qui chevauchent la gamme rappelée, le client devrait retourner à CB_LAYOUTRECALL le code d'erreur NFS4ERR_NOMATCHING_LAYOUT. Cela permet au serveur de mettre à jour sa vue de l'état de disposition du client.

12.5.5.2 Séquençage des opérations de disposition

Comme avec les autres opérations à états pleins, pNFS exige le séquençage correct des opérations de disposition. pNFS utilise le "seqid" dans l'identifiant d'état de la disposition pour fournir le séquençage correct entre les opérations régulières et les rappels. Il est de la responsabilité du serveur d'éviter des incohérences sur les dispositions fournies et de la responsabilité du client de mettre correctement à la suite ses demandes et retours de disposition.

12.5.5.2.1 Rappel de disposition et séquençage de retour

Une question critique à l'égard du séquençage des opérations de disposition concerne les rappels. Le protocole doit défendre contre la compétition entre la réponse à une opération LAYOUTGET ou LAYOUTRETURN et un CB_LAYOUTRECALL suivant. Un client NE DOIT PAS traiter un CB_LAYOUTRECALL qui implique une ou plusieurs opérations LAYOUTGET ou LAYOUTRETURN en instance auxquelles le client n'a pas encore reçu de réponse. Le client détecte un tel CB_LAYOUTRECALL en examinant le champ "seqid" de l'identifiant d'état de disposition du rappel. Si le "seqid" n'est pas exactement supérieur de un à ce que le client a actuellement enregistré, et si le client a au moins une opération LAYOUTGET et/ou LAYOUTRETURN en instance, le client sait que le serveur a envoyé le CB_LAYOUTRECALL après l'envoi d'une réponse à un LAYOUTGET ou LAYOUTRETURN en instance. Le client DOIT attendre avant de traiter un tel CB_LAYOUTRECALL d'avoir traité toutes les réponses pour des opérations LAYOUTGET et LAYOUTRETURN en instance pour le fichier correspondant avec un seqid inférieur au seqid donné par le CB_LAYOUTRECALL (lor_stateid ; voir le paragraphe 20.3.)

En plus du mécanisme fondé sur le seqid, le paragraphe 2.10.6.3 décrit le mécanisme de session pour permettre au client de détecter les conditions de concours de rappel et de retarder le traitement d'un tel CB_LAYOUTRECALL. Le serveur PEUT faire référence à des opérations en conflit dans le CB_SEQUENCE qui précède le CB_LAYOUTRECALL. Parce que le serveur a déjà envoyé des réponses pour ces opérations avant d'envoyer le rappel, les réponses peuvent entrer en concurrence avec le CB_LAYOUTRECALL. Le client DOIT attendre que tous les appels référencés soient achevés et mettre à jour sa vue de l'état de la disposition avant de traiter le CB_LAYOUTRECALL.

12.5.5.2.1.1 Séquençage de Get/Return

Le protocole permet au client d'envoyer des opérations LAYOUTGET et LAYOUTRETURN concurrentes au serveur. Le protocole ne fournit aucun moyen pour que le serveur traite les demandes dans le même ordre que celui de leur création. Cependant, par l'utilisation du champ "seqid" dans l'identifiant d'état de la disposition, le client peut déterminer l'ordre dans lequel les opérations parallèles en instance ont été traitées par le serveur. Donc, quand une disposition restituée par une opération LAYOUTGET en instance se recoupe avec une disposition retournée par un LAYOUTRETURN en instance sur le même fichier, l'ordre dans lequel les deux opérations en conflit sont traitées détermine l'état final de la disposition en chevauchement. L'ordre est déterminé par le "seqid" retourné dans chaque opération : l'opération avec le plus haut seqid a été exécutée la dernière.

Il est permis au client d'envoyer plusieurs opérations LAYOUTGET en parallèle pour le même fichier ou plusieurs opérations LAYOUTRETURN en parallèle pour le même fichier ou un mélange des deux.

Il est permis au client d'utiliser l'identifiant d'état courant (voir le paragraphe 16.2.3.1.2) pour les opérations LAYOUTGET, par exemple, quand il compose des LAYOUTGET ou des OPEN et des LAYOUTGET. Il est aussi permis d'utiliser l'identifiant d'état courant quand il compose des LAYOUTRETURN.

Il est permis au client d'utiliser l'identifiant d'état courant quand il combine des opérations LAYOUTRETURN et LAYOUTGET pour le même fichier dans la même demande COMPOUND car le serveur DOIT les traiter dans l'ordre. Cependant, si un client envoie de telles demandes COMPOUND, il NE DOIT PAS en avoir plus d'une en instance pour le

même fichier au même moment, et il NE DOIT PAS avoir d'autre opération LAYOUTGET ou LAYOUTRETURN en instance au même moment pour ce même fichier.

12.5.5.2.1.2 Considérations de client

Considérons un client pNFS qui a envoyé un LAYOUTGET, et avant qu'il reçoive la réponse au LAYOUTGET, qu'il reçoive un CB_LAYOUTRECALL pour le même fichier avec une gamme en chevauchement. Il y a deux possibilités, que le client peut distinguer via l'identifiant d'état de disposition dans le rappel :

1. Le serveur a traité le LAYOUTGET avant d'envoyer le rappel, donc le LAYOUTGET doit être attendu parce que il peut porter des informations de disposition qui devront être retournées pour traiter le CB_LAYOUTRECALL.
2. Le serveur a envoyé le rappel avant de recevoir le LAYOUTGET. Le serveur ne va pas répondre au LAYOUTGET tant que le CB_LAYOUTRECALL n'est pas traité.

Si ces possibilités ne peuvent pas être distinguées, il pourrait en résulter une impasse, car le client doit attendre la réponse au LAYOUTGET avant de traiter le rappel dans le premier cas, mais cette réponse ne va arriver qu'après de traitement du rappel dans le second cas. Noter que dans le premier cas, le "seqid" dans l'identifiant d'état de disposition du rappel est supérieur de deux à ce que le client a enregistré ; dans le second cas, le "seqid" est supérieur de un à ce que le client a enregistré. Cela permet au client de distinguer entre les deux cas. Le client sait donc précisément quelle possibilité s'applique.

Dans le cas 1, le client sait qu'il doit attendre la réponse au LAYOUTGET avant de traiter le rappel (ou le client peut retourner NFS4ERR_DELAY).

Dans le cas 2, le client ne va pas attendre la réponse à LAYOUTGET avant de traiter le rappel parce que attendre causerait une impasse. Donc, l'action chez le client exige seulement d'attendre dans le cas où le client n'a pas encore vu les réponses antérieures du serveur à la ou les opérations LAYOUTGET.

Le processus de rappel peut être considéré comme terminé quand l'opération finale LAYOUTRETURN pour la gamme rappelée est terminée. Le LAYOUTRETURN utilise l'identifiant d'état de disposition (avec seqid) spécifié dans CB_LAYOUTRECALL. Si le client utilise plusieurs LAYOUTRETURN dans le traitement du rappel, le premier LAYOUTRETURN va utiliser l'identifiant d'état de disposition comme spécifié dans CB_LAYOUTRECALL. Les LAYOUTRETURN suivants vont utiliser le plus haut seqid comme est le cas habituel.

12.5.5.2.1.3 Considérations de serveur

Considérons une compétition du point de vue du serveur de métadonnées. Le serveur de métadonnées a envoyé un CB_LAYOUTRECALL et reçoit un LAYOUTGET en chevauchement pour le même fichier avant le ou les LAYOUTRETURN qui répondent au CB_LAYOUTRECALL. Il y a trois cas :

1. Le client a envoyé le LAYOUTGET avant de traiter le CB_LAYOUTRECALL. Le "seqid" dans l'identifiant d'état de disposition des arguments de LAYOUTGET est un de moins que le "seqid" dans CB_LAYOUTRECALL. Le serveur retourne NFS4ERR_RECALLCONFLICT au client, ce qui indique au client qu'il y a un rappel en instance.

2. Le client a envoyé le LAYOUTGET après le traitement du CB_LAYOUTRECALL, mais le LAYOUTGET est arrivé avant le LAYOUTRETURN et la réponse au CB_LAYOUTRECALL qui terminait ce traitement. Le "seqid" dans l'identifiant d'état de disposition de LAYOUTGET est égal ou supérieur à celui du "seqid" dans CB_LAYOUTRECALL. Le serveur n'a pas reçu de réponse au CB_LAYOUTRECALL, donc il retourne NFS4ERR_RECALLCONFLICT.

3. Le client a envoyé le LAYOUTGET après le traitement du CB_LAYOUTRECALL ; le serveur a reçu la réponse au CB_LAYOUTRECALL, mais le LAYOUTGET est arrivé avant le LAYOUTRETURN qui terminait ce traitement. Le "seqid" dans l'identifiant d'état de disposition du LAYOUTGET est égal à celui du "seqid" dans CB_LAYOUTRECALL. Le serveur a reçu une réponse au CB_LAYOUTRECALL, donc il retourne NFS4ERR_RETURNCONFLICT.

12.5.5.2.1.4 Retour à zéro et validation du seqid

Les règles pour le traitement de l'identifiant d'état de disposition diffèrent de celles des autres identifiants d'état dans le protocole parce que la valeur de "seqid" ne peut pas être zéro et la valeur du "seqid" de l'identifiant d'état change dans une opération CB_LAYOUTRECALL. L'exigence de non zéro combinée avec le parallélisme inhérent des opérations de disposition signifie que un ensemble d'opérations LAYOUTGET et LAYOUTRETURN peut contenir la même valeur pour

"seqid". Le serveur utilise une version légèrement modifiée de l'arithmétique de modulo comme décrite au paragraphe 2.10.6.1 quand il incrémenté le "seqid" de l'identifiant d'état de disposition. La différence est que zéro n'est pas une valeur valide pour "seqid" ; quand la valeur d'un "seqid" est 0xFFFFFFFF, la prochaine valeur valide va être 0x00000001. L'arithmétique de modulo est aussi utilisée pour les comparaisons de valeurs de "seqid" dans le traitement des événements CB_LAYOUTRECALL comme décrit au paragraphe 12.5.5.2.1.3.

Tout comme le serveur valide le "seqid" en cas d'usage de CB_LAYOUTRECALL, comme décrit au paragraphe 12.5.5.2.1.3, le serveur valide aussi la valeur de "seqid" pour s'assurer qu'il est dans une gamme appropriée. Cette gamme représente le degré de parallélisme que le serveur prend en charge pour les identifiants d'état de disposition. Si le client envoie en parallèle plusieurs opérations de disposition au serveur, par définition, la valeur de "seqid" dans l'identifiant d'état fourni ne va pas être le "seqid" courant détenu par le serveur. La gamme de parallélisme s'étend de la plus haute valeur de "seqid" ou du "seqid" courant à une valeur de "seqid" passée. Pour aider à la discussion, la valeur du "seqid" courant du serveur pour un identifiant d'état de disposition est définie comme SERVER_CURRENT_SEQID. La valeur du plus bas "seqid" qui est acceptable au serveur est représentée par PAST_SEQID. Et la valeur pour la gamme de "seqid" valides ou gamme de parallélisme est VALID_SEQID_RANGE. Donc, l'équation suivante tient : VALID_SEQID_RANGE = SERVER_CURRENT_SEQID - PAST_SEQID. Dans la suite, toute l'arithmétique est celle de modulo, comme décrit ci-dessus.

Le serveur DOIT prendre en charge une VALID_SEQID_RANGE minimum. Le minimum est défini comme : VALID_SEQID_RANGE = somme de 1 à N de (ca_maxopérations(i) - 1), où N est le nombre de canaux de session avant et ca_maxopérations(i) est la valeur des ca_maxopérations retournées de CREATE_SESSION de la ième session. La raison du "- 1" est de permettre l'opération SEQUENCE requise. Le serveur PEUT prendre en charge une valeur de VALID_SEQID_RANGE supérieure au minimum. Le maximum de VALID_SEQID_RANGE est (2^(32) - 2) (tenant compte de ce que zéro n'est pas une valeur valide de "seqid").

Si le serveur trouve que le "seqid" est zéro, l'erreur NFS4ERR_BAD_STATEID est retournée au client. Le serveur valide de plus le "seqid" pour s'assurer qu'il est dans la gamme de parallélisme, VALID_SEQID_RANGE. Si la valeur de "seqid" n'est pas dans cette gamme, l'erreur NFS4ERR_OLD_STATEID est retournée au client. À réception de NFS4ERR_OLD_STATEID, le client met à jour l'identifiant d'état dans la demande de disposition fondée sur le traitement des autres demandes de disposition et envoie à nouveau l'opération au serveur.

12.5.5.2.1.5 Rappel et retour en vrac

pNFS prend en charge le rappel et le retour de toutes les dispositions qui sont pour les fichiers qui appartiennent à un fsid (LAYOUTRECALL4_FSID, LAYOUTRETURN4_FSID) ou identifiant de client (LAYOUTRECALL4_ALL, LAYOUTRETURN4_ALL) particulier. Il n'y a pas d'identifiant d'état "en vrac", de sorte que la détection de concurrence via le seqid n'est pas possible. Le serveur NE DOIT PAS initier de rappel en vrac alors qu'un autre rappel est en cours, ou lorsque le LAYOUTRETURN correspondant est en cours ou en instance. Dans le cas où le serveur envoie un rappel en vrac alors que le client a un LAYOUTRETURN, CB_LAYOUTRECALL, ou LAYOUTGET en instance ou en cours, le client retourne NFS4ERR_DELAY. Au cas où le client envoie un LAYOUTGET ou LAYOUTRETURN alors qu'un rappel en vrac est en cours, le serveur retourne NFS4ERR_RECALLCONFLICT. Si le client envoie un LAYOUTGET ou LAYOUTRETURN après que le serveur reçoit NFS4ERR_DELAY provenant d'un rappel en vrac, alors pour assurer la progression, le serveur PEUT retourner NFS4ERR_RECALLCONFLICT.

Une fois qu'un CB_LAYOUTRECALL de LAYOUTRECALL4_ALL est envoyé, le serveur NE DOIT PAS permettre que le client utilise d'identifiant d'état de disposition sauf pour les opérations LAYOUTCOMMIT. Une fois que le client a reçu un CB_LAYOUTRECALL de LAYOUTRECALL4_ALL, il NE DOIT PAS utiliser d'identifiant d'état de disposition sauf pour des opérations LAYOUTCOMMIT. Une fois qu'un LAYOUTRETURN de LAYOUTRETURN4_ALL est envoyé, tous les identifiants d'état de disposition accordés à l'identifiant de client sont libérés. Le client NE DOIT PAS utiliser à nouveau d'identifiants d'état de disposition. Il DOIT utiliser LAYOUTGET pour obtenir de nouveaux identifiants d'état de disposition.

Une fois qu'un CB_LAYOUTRECALL de LAYOUTRECALL4_FSID est envoyé, le serveur NE DOIT PAS permettre au client d'utiliser d'identifiant d'état de disposition qui se réfère à un fichier avec le fsid spécifié sauf pour des opérations LAYOUTCOMMIT. Une fois que le client a reçu un CB_LAYOUTRECALL de LAYOUTRECALL4_ALL, il NE DOIT PAS utiliser d'identifiant d'état de disposition qui se réfère à un fichier avec le fsid spécifié sauf pour des opérations LAYOUTCOMMIT. Une fois qu'un LAYOUTRETURN de LAYOUTRETURN4_FSID est envoyé, tous les identifiants d'état de disposition accordés au fsid référencé sont libérés. Le client NE DOIT PAS utiliser à nouveau ces identifiants d'état de disposition libérés pour des fichiers avec le fsid référencé. Ensuite, pour tout fichier avec le fsid référencé, pour utiliser une disposition, le client DOIT d'abord envoyer une opération LAYOUTGET afin d'obtenir un nouvel identifiant d'état de disposition pour ce fichier.

Si le serveur a envoyé un CB_LAYOUTRECALL en vrac et reçoit un LAYOUTGET, ou un LAYOUTRETURN avec un identifiant d'état, le serveur DOIT retourner NFS4ERR_RECALLCONFLICT. Si le serveur a envoyé un CB_LAYOUTRECALL en vrac et reçoit un LAYOUTRETURN avec un lr_returntype qui n'est pas égal au lor_recalltype du CB_LAYOUTRECALL, le serveur DOIT retourner NFS4ERR_RECALLCONFLICT.

12.5.6 Révocation de dispositions

NFS parallèle permet aux serveurs de révoquer les dispositions provenant de clients qui manquent à répondre aux rappels et/ou manquent à renouveler leur prêt à temps. Selon le type de disposition, le serveur pourrait révoquer la disposition et pourrait prendre certaines mesures par rapport aux entrées/sorties du client sur les serveurs de données.

12.5.7 Propagation d'écriture de serveur de métadonnées

Des écritures asynchrones à travers le serveur de métadonnées peuvent être propagées paresseusement aux appareils de mémorisation. Pour les données écrites de façon asynchrone à travers le serveur de métadonnées, il n'est pas garanti à un client qui effectue une lecture sur l'appareil de mémorisation approprié qu'il va voir les données nouvellement écrites avant qu'un COMMIT se produise chez le serveur de métadonnées. Bien que l'écriture soit en instance, les lectures sur l'appareil de mémorisation peuvent donner les vieilles données, les nouvelles données, ou un mélange de nouvelles et d'anciennes. À l'achèvement d'un WRITE ou COMMIT synchrone (pour des données écrites de façon asynchrone) le serveur de métadonnées DOIT s'assurer que les appareils de mémorisation rendent les nouvelles données et que les données ont été écrites sur une mémorisation stable. Si le serveur met en œuvre sa mémorisation d'une façon telle qu'il ne puisse pas respecter ces contraintes, alors il DOIT rappeler les dispositions pour empêcher que des lectures soient faites qui ne peuvent pas être traitées correctement. Noter que les dispositions DOIVENT être rappelées avant que le serveur réponde aux opérations WRITE associées.

12.6 Mécanique de pNFS

Ce paragraphe décrit les flux d'opérations d'un client pNFS à un serveur de métadonnées et appareil de mémorisation.

Quand un client pNFS rencontre un nouveau FSID, il envoie un GETATTR au serveur NFSv4.1 pour l'attribut fs_layout_type (paragraphe 5.12.1). Si l'attribut retourne au moins un type de disposition, et si les types de disposition retournés sont parmi l'ensemble pris en charge par le client, le client sait que pNFS est une possibilité pour le système de fichiers. Si, du serveur qui a retourné le nouveau FSID, le client n'a pas un identifiant de client qui vient d'un résultat EXCHANGE_ID qui a retourné EXCHGID4_FLAG_USE_PNFS_MDS, il DOIT envoyer un EXCHANGE_ID au serveur avec le bit EXCHGID4_FLAG_USE_PNFS_MDS établi. Si la réponse du serveur n'a pas EXCHGID4_FLAG_USE_PNFS_MDS, alors contrairement à ce que dit l'attribut fs_layout_type, le serveur ne prend pas en charge pNFS, et le client ne va pas être capable d'utiliser pNFS avec ce serveur ; dans ce cas, le serveur DOIT retourner NFS4ERR_NOTSUPP en réponse à toute opération pNFS.

Le client crée alors une session, demandant une session persistante, afin que des créations exclusives puissent être faites avec un seul aller retour via le createmode4 de GUARDED4. Si la session se révèle non persistante, le client va utiliser EXCLUSIVE4_1 pour des créations exclusives.

Si un fichier est à créer sur un système de fichiers à capacité pNFS, le client utilise l'opération OPEN. Avec l'ensemble normal d'attributs qui peuvent être fournis sur un OPEN utilisé pour une création, il y a un attribut FACULTATIF layout_hint. L'utilisation par le client de layout_hint lui permet d'exprimer sa préférence pour un type de disposition et ses détails de disposition associés. L'utilisation du createmode4 de UNCHECKED4, GUARDED4, ou EXCLUSIVE4_1 va permettre au client de fournir l'attribut layout_hint au moment de la création. Le client NE DOIT PAS utiliser EXCLUSIVE4 (voir le Tableau 18). Il est RECOMMANDÉ au client de combiner une opération GETATTR après le OPEN au sein du même COMPOUND. Le GETATTR peut alors restituer l'attribut layout_type pour le fichier nouvellement créé. Le client va alors savoir quel type de disposition le serveur a choisi pour le fichier et donc quel protocole de mémorisation le client doit utiliser.

Si le client veut ouvrir un fichier existant, il inclut alors aussi un GETATTR pour déterminer quel type de disposition le fichier prend en charge.

Le GETATTR dans la création du fichier ou dans le cas d'ouverture d'un fichier complet peut aussi inclure les attributs layout_blksize et layout_alignment afin que le client puisse déterminer les décalages et longueurs optimaux pour les

entrées/sorties sur le fichier.

En supposant que le client prenne en charge le type de disposition retourné par GETATTR et qu'il choisisse d'utiliser pNFS pour l'accès aux données, il envoie alors LAYOUTGET en utilisant la bride de fichier et l'identifiant d'état retournés par OPEN, spécifiant la gamme sur laquelle il veut faire les entrées/sorties. La réponse est une disposition, qui peut être un sous ensemble de la gamme demandée par le client. Elle inclura aussi les identifiants d'appareil et une description de comment sont organisées les données (ou dans le cas d'écriture, comment les données vont être organisées) à travers les appareils. Les identifiants d'appareil et la description des données sont codés dans un format spécifique du type de disposition, mais que le client est supposé comprendre.

Quand le client veut envoyer une entrée/sortie, il détermine à quel identifiant d'appareil il doit envoyer la commande d'entrée/sortie en examinant la description des données dans la disposition. Il envoie alors un GETDEVICEINFO pour trouver la ou les adresses d'appareil de l'identifiant d'appareil. Le client envoie alors la demande d'entrée/sortie à une des adresses d'appareil de l'identifiant d'appareil, en utilisant le protocole de mémorisation défini pour le type de disposition. Noter que si un client a plusieurs entrées/sorties à envoyer, ces demandes peuvent être faites en parallèle.

Si l'entrée/sortie était un WRITE, alors à un moment le client peut vouloir utiliser LAYOUTCOMMIT pour engager l'instant de modification et la nouvelle taille du fichier (si il croit qu'il étend la taille du fichier) auprès du serveur de métadonnées et les données modifiées au système de fichiers.

12.7 Récupération

La récupération est compliquée par la nature répartie du protocole pNFS. En général, la récupération de défaillances pour les dispositions est similaire à la récupération de défaillances pour les délégations dans le protocole NFSv4.1 de base. Cependant, la capacité du client d'effectuer des entrées/sorties sans contacter le serveur de métadonnées introduit des subtilités qui doivent être traitées correctement si la possibilité d'une corruption du système de fichiers doit être évitée.

12.7.1 Récupération à partir d'un redémarrage du client

La récupération des dispositions par le client est similaire à la récupération par le client des autres états de verrou et délégation. Quand un client pNFS redémarre, il va perdre toutes les informations sur les dispositions qu'il possédait précédemment. Il y a deux méthodes par lesquelles le serveur peut réclamer ces ressources et permettre que des dispositions par ailleurs en conflit soient fournies aux autres clients.

La première est par l'expiration du prêt du client. Si le temps de récupération du client est plus long que la période de prêt, le prêt du client va expirer et le serveur va savoir que l'état peut être libéré. Pour les dispositions, le serveur peut libérer l'état immédiatement à l'expiration du prêt ou il peut permettre que la disposition persiste, attendant une possible résurrection du prêt, pour autant qu'elle ne soit pas en conflit avec une autre disposition.

La seconde est par le redémarrage du client en moins de temps qu'il n'en faut pour que la période de prêt expire. Dans ce cas, le client va contacter le serveur par le protocole EXCHANGE_ID standard. Le serveur va trouver que le co_ownerid du client correspond au co_ownerid de l'invocation précédente du client, mais que le vérificateur est différent. Le serveur utilise cela comme signal pour libérer tout l'état de disposition associé à l'invocation précédente du client. Dans ce scénario, les données écrites par le client mais non couvertes par un LAYOUTCOMMIT réussi sont dans un état indéfini ; elles peuvent avoir été écrites ou peuvent maintenant être perdues. C'est un comportement acceptable et il est de la responsabilité du client d'utiliser LAYOUTCOMMIT pour réaliser le niveau de stabilité désiré.

12.7.2 Traitement de l'expiration du prêt chez le client

Si un client croit que son prêt a expiré, il NE DOIT PAS envoyer d'entrée/sortie à l'appareil de mémorisation avant d'avoir validé son prêt. Le client peut envoyer une opération SEQUENCE au serveur de métadonnées. Si l'opération SEQUENCE réussit, mais si le fanion sr_status_flag a établi SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED, SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED, ou SEQ4_STATUS_ADMIN_STATE_REVOKED, le client NE DOIT PAS utiliser les dispositions actuellement détenues. Le client a un choix pour récupérer de l'expiration de prêt. D'abord, pour toutes les données modifiées mais non engagées, le client les écrit au serveur de métadonnées en utilisant le fanion FILE_SYNC4 pour les WRITE, ou WRITE et COMMIT. Ou bien, le client re-établit un identifiant de client et une session avec le serveur et obtient de nouvelles dispositions et transpositions d'identifiant d'appareil en adresse d'appareil pour les gammes de données modifiées et écrit alors les données aux appareils de mémorisation avec les nouvelles dispositions obtenues.

Si des fanions sr_status_flags provenant du serveur de métadonnées ont SEQ4_STATUS_RESTART_RECLAIM_NEEDED établi (ou si SEQUENCE retourne NFS4ERR_BAD_SESSION et CREATE_SESSION retourne NFS4ERR_STALE_CLIENTID) alors le serveur de métadonnées a redémarré, et le client DEVRAIT récupérer en utilisant les méthodes décrites au paragraphe 12.7.4.

Si sr_status_flags provenant du serveur de métadonnées a SEQ4_STATUS_LEASE_MOVED établi, alors le client récupère en suivant la procédure décrite au paragraphe 11.11.9.2. Après cela, le client peut obtenir une indication que l'état de disposition n'a pas bougé avec le système de fichiers. Le client récupère comme dans les autres situations applicables discutées dans les deux premiers alinéas de ce paragraphe.

Si sr_status_flags ne rapporte pas de perte d'état, alors les prêts pour les dispositions qu'a le client sont valides et renouvelés, et le client peut une fois encore envoyer des demandes d'entrée/sortie aux appareils de mémorisation.

Bien que les clients NE DEVRAIENT PAS envoyer aux appareils de mémorisation des demandes d'entrée/sortie qui peuvent s'étendre au delà de la période d'expiration de prêt, ce n'est pas toujours possible, par exemple, une partition de réseau étendue qui commence après l'envoi de l'entrée/sortie et n'est pas guérie avant que la demande d'entrée/sortie soit reçue par l'appareil de mémorisation. Donc, le serveur de métadonnées et/ou les appareils de mémorisation sont chargés de se protéger des entrées/sorties qui sont envoyées avant que le prêt expire et arrivent après l'expiration du prêt. Voir au paragraphe 12.7.3.

12.7.3 Traitement de la perte d'état de disposition sur le serveur de métadonnées

On décrit le cas où tout ce qui suit est vrai :
* le serveur de métadonnées n'a pas redémarré,
* les dispositions d'un client pNFS ont été éliminées (généralement parce que le prêt du client a expiré) et sont invalides,
* une entrée/sortie du client pNFS arrive à l'appareil de mémorisation.

Le serveur de métadonnées et ses appareils de mémorisation DOIVENT résoudre cela en barricadant le client. En d'autres termes, il DOIT résoudre cela en empêchant l'exécution des opérations d'entrée/sortie du client sur les appareils de mémorisation après la perte de l'état de disposition. Les détails de comment la barricade est faite sont spécifiques du type de disposition. La solution pour les dispositions fondées sur le fichier NFSv4.1 est décrite au paragraphe 13.11, et les solutions pour les autres types de disposition sont dans leurs documents respectifs de spécification externe.

12.7.4 Récupération d'un redémarrage du serveur de métadonnées

Le client pNFS va découvrir que le serveur de métadonnées a redémarré via les méthodes décrites au paragraphe 8.4.2 et discutées dans un contexte spécifique de pNFS au paragraphe 12.7.2, alinéa 2. Le client DOIT cesser d'utiliser les dispositions et supprimer les transpositions d'identifiant d'appareil en adresse d'appareil qu'il avait précédemment reçues du serveur de métadonnées. Cela fait, si le client a écrit des données à l'appareil de mémorisation sans engager les dispositions via LAYOUTCOMMIT, alors le client a un travail supplémentaire à faire pour que le client, le serveur de métadonnées, et le ou les appareils de mémorisation se synchronisent tous sur l'état des données.

* Si le client a encore des données modifiées et non écrites dans la mémoire du client, le client a deux solutions :
  1. Le client peut obtenir une disposition via LAYOUTGET après la période de grâce du serveur et écrire les données sur les appareils de mémorisation.
  2. Le client peut écrire ces données par le serveur de métadonnées en utilisant l'opération WRITE (paragraphe 18.32) et obtenir alors les dispositions désirées.

* Si le client a écrit les données de façon asynchrone à l'appareil de mémorisation, mais a encore une copie des données dans sa mémoire, alors il a disponibles les options de récupération mentionnées au point précédent. Si le serveur de métadonnées est aussi dans sa période de grâce, le client a la disponibilité des options du point suivant.

* Le client n'a pas de copie des données dans sa mémoire et le serveur de métadonnées est encore dans sa période de grâce. Le client ne peut pas utiliser LAYOUTGET (dans ou hors de la période de grâce) pour réclamer une disposition parce que le contenu de la réponse de LAYOUTGET ne peut pas correspondre à ce qu'il était précédemment. La gamme pourrait être différente ou le client pourrait obtenir la même gamme mais le contenu de la disposition pourrait être différent. Même si le contenu de la disposition paraît être le même, les identifiants d'appareil peuvent se transposer en adresses d'appareil différentes, et même si les adresses d'appareil sont les mêmes, les adresses d'appareil pourraient

avoir été allouées à un appareil de mémorisation différent. L'option de restituer les données de l'appareil de mémorisation et de les écrire au serveur de métadonnées selon le scénario de récupération décrit ci-dessus n'est pas disponible parce que, là encore, les transpositions de gamme en identifiant d'appareil, d'identifiant d'appareil en adresse d'appareil, et d'adresse d'appareil en appareil physique sont périmées, et de nouvelles transpositions via un nouveau LAYOUTGET ne résolvent pas le problème.

La seule option de récupération pour ce scénario est d'envoyer un LAYOUTCOMMIT en mode réclamation, que le serveur de métadonnées va accepter tant qu'il est dans sa période de grâce. L'utilisation de LAYOUTCOMMIT en mode réclamation informe le serveur de métadonnées que la disposition a changé. Il est critique que le serveur de métadonnées reçoive cette information avant la fin de sa période de grâce, et donc avant qu'il commence à permettre des mises à jour au système de fichiers.

Pour envoyer LAYOUTCOMMIT en mode réclamation, le client règle le champ loca_reclaim des arguments de l'opération (paragraphe 18.42.1) à VRAI. Durant la période de grâce de récupération du serveur de métadonnées (et seulement durant la période de grâce de récupération) le serveur de métadonnées est prêt à accepter des demandes LAYOUTCOMMIT avec le champ loca_reclaim réglé à VRAI.

Quand loca_reclaim est VRAI, le client tente d'engager des changements à la disposition qui se sont produits avant le redémarrage du serveur de métadonnées. Le serveur de métadonnées applique des vérifications de cohérence sur le champ loca_layoutupdate des arguments pour déterminer si le client peut engager les données écrites sur l'appareil de mémorisation dans le système de fichiers. Le champ loca_layoutupdate est de type de données layoutupdate4 et contient un contenu spécifique du type de disposition (dans le champ lou_body de loca_layoutupdate). Les informations spécifiques du type de disposition que loca_layoutupdate pourrait avoir sont discutées au paragraphe 12.5.4.3. Si les vérifications de cohérence du serveur de métadonnées sur loca_layoutupdate réussissent, alors le serveur de métadonnées DOIT engager les données (comme décrit par les champs loca_offset, loca_length, et loca_layoutupdate des arguments) qui ont été écrites sur l'appareil de mémorisation. Si les vérifications de cohérence du serveur de métadonnées sur loca_layoutupdate échouent, le serveur de métadonnées rejette l'opération LAYOUTCOMMIT et ne fait pas de changement au système de fichiers. Cependant, chaque fois que LAYOUTCOMMIT avec loca_reclaim VRAI échoue, le client pNFS perd toutes les données dans la gamme définie par <loca_offset, loca_length>. Un client peut se défendre contre ce risque en mettant toutes les données en antémémoire, qu'elles soient écrites en synchrone ou asynchrone dans sa mémoire, et en ne libérant pas les données en antémémoire avant un LAYOUTCOMMIT réussi. Cette condition n'est pas vraie pour tous les types de disposition ; par exemple, les appareils de mémorisation fondés sur le fichier ne subissent pas cette limitation.

* Le client n'a pas de copie des données dans sa mémoire et le serveur de métadonnées n'est plus dans sa période de grâce ; c'est-à-dire, le serveur de métadonnées retourne NFS4ERR_NO_GRACE. Comme dans le scénario précédent, l'échec de LAYOUTCOMMIT signifie que les données dans la gamme <loca_offset, loca_length> sont perdues. La défense contre ce risque est la même -- mise en antémémoire de toutes les données écrites chez le client jusqu'à un LAYOUTCOMMIT réussi.

12.7.5 Opérations durant la période de grâce du serveur de métadonnées

Certains des scénarios de récupération ont noté jusqu'à présent que certaines opérations (à savoir WRITE et LAYOUTGET) pourraient être permises durant la période de grâce du serveur de métadonnées. Le serveur de métadonnées peut permettre ces opérations durant sa période de grâce. Pour LAYOUTGET, le serveur de métadonnées doit déterminer fiablement que servir une telle demande ne va pas entrer en conflit avec une demande de réclamation LAYOUTCOMMIT qui surviendrait. Pour WRITE, le serveur de métadonnées doit déterminer fiablement que servir la demande ne va pas entrer en conflit avec la survenue d'un OPEN ou avec un LOCK où le fichier a un verrouillage obligatoire de gamme d'octets activé.

Comme mentionné précédemment, pour aller vite, le serveur de métadonnées pourrait rejeter des opérations (à savoir, WRITE et LAYOUTGET) durant sa période de grâce, parce que l'approche correcte la plus simple est de rejeter toutes les demandes pNFS non de réclamation et les opérations WRITE en retournant l'erreur NFS4ERR_GRACE. Cependant, selon le protocole de mémorisation (qui est spécifique du type de disposition) et la mise en œuvre de serveur de métadonnées, le serveur de métadonnées peut être capable de déterminer qu'une demande particulière est sûre. Par exemple, un serveur de métadonnées peut sauvegarder des transpositions d'allocation provisoires pour chaque fichier en mémorisation stable, ainsi que des informations sur des modes de partage d'ouverture potentiellement conflictuels et des verrous de gamme d'octets obligatoires qui pourraient avoir été en vigueur au moment du redémarrage, et le serveur de métadonnées peut utiliser ces informations durant la période de grâce de récupération pour déterminer qu'une demande WRITE est sûre.

12.7.6 Récupération de l'appareil de mémorisation

La récupération après le redémarrage de l'appareil de mémorisation dépend principalement du type de disposition utilisé. Cependant, il y a peu de techniques générales qu'un client peut utiliser si il découvre qu'un appareil de mémorisation est défaillant alors que des données modifiées détenues non engagées ont été écrites de façon asynchrone. Tout d'abord, il est important de réaliser que le client est le seul a avoir les informations nécessaires pour récupérer les données non engagées car il détient les données modifiées et probablement personne d'autre ne les a. Ensuite, la meilleure solution est que le client penche du côté de la prudence et tente de réécrire les données modifiées par un autre chemin.

Le client DEVRAIT immédiatement écrire les données au serveur de métadonnées, avec le champ stable dans WRITE4args réglé à FILE_SYNC4. Une fois que c'est fait, il n'est pas besoin d'attendre l'appareil de mémorisation original.

12.8 Rôles d'appareil de métadonnées et de mémorisation

Si le même matériel physique est utilisé pour mettre en œuvre un serveur de métadonnées et un appareil de mémorisation, alors la même entité matérielle va mettre en œuvre les deux rôles distincts et il est important qu'il soit clairement compris au nom de quel rôle le matériel agit à un instant donné.

Deux sous cas peuvent être distingués :

1. L'appareil de mémorisation utilise NFSv4.1 comme protocole de mémorisation, c'est-à-dire, le même matériel physique est utilisé pour mettre en œuvre les deux serveurs de métadonnées et de données. Voir au paragraphe 13.1 la description de comment plusieurs rôles sont traités.

2. L'appareil de mémorisation n'utilise pas NFSv4.1 comme protocole de mémorisation, et le même matériel physique est utilisé pour mettre en œuvre les deux appareils de métadonnées et de mémorisation. Si des adresses réseau distinctes sont utilisées pour accéder au serveur de métadonnées et à l'appareil de mémorisation est sans importance. C'est parce que il est toujours clair au client et serveur pNFS, vu du protocole de couche supérieure utilisé (NFSv4.1 ou non) sur quel rôle est dirigée la demande à l'adresse réseau commune de serveur.

12.9 Considérations de sécurité pour pNFS

pNFS sépare les métadonnées et les données de système de fichiers et fournit l'accès aux deux. Il y a des opérations spécifiques de pNFS (mentionnées au paragraphe 12.3) qui fournissent l'accès aux métadonnées ; tous les mécanismes de sécurité et caractéristiques NFSv4.1 conventionnels (non pNFS) existants s'appliquent à l'accès aux métadonnées. La combinaison de composants dans un système pNFS (voir la Figure 1) est exigée pour préserver les propriétés de sécurité de NFSv4.1 à l'égard d'une entité à laquelle accède un appareil de mémorisation à partir d'un client, incluant des contre mesures de sécurité pour défendre contre des menaces pour lesquelles NFSv4.1 fournit des défenses dans des environnements où ces menaces sont considérées comme significatives.

Dans certains cas, les contre mesures de sécurité pour les connexions aux appareils de mémorisation peuvent prendre la forme d'une isolation physique ou d'une recommandation d'éviter l'utilisation de pNFS dans un environnement. Par exemple, il peut être impraticable de fournir la protection de la confidentialité pour certains protocoles de mémorisation pour protéger contre l'espionnage. Dans les environnements où l'espionnage sur de tels protocoles est un souci suffisant pour exiger des contre mesures, l'isolation physique du canal de communication (par exemple, via une connexion directe du ou des clients aux appareils de mémorisations) et/ou une décision de renoncer à l'utilisation de pNFS (par exemple, et revenir au NFSv4.1 conventionnel) peut être appropriée.

Lorsque la communication avec les appareils de mémorisation est soumise aux mêmes menaces que la communication de client à serveur de métadonnées, les protocoles utilisés pour cette communication doivent fournir des mécanismes de sécurité aussi forts ou pas plus faibles que ceux disponibles via RPCSEC_GSS pour NFSv4.1. Sauf pour le protocole de mémorisation utilisé pour la disposition LAYOUT4_NFSV4_1_FILES (voir la Section 13) c'est-à-dire, sauf pour NFSv4.1, il sort du domaine d'application de ce document de spécifier les mécanismes de sécurité pour les protocoles d'accès à la mémorisation.

Les mises en œuvre de pNFS NE DOIVENT PAS supprimer les contrôles d'accès de NFSv4.1. La combinaison de clients, d'appareils de mémorisation, et de serveur de métadonnées est responsable de s'assurer que tout accès de données de fichier de client à appareil de mémorisation respecte les ACL et modes d'ouverture de fichier de NFSv4.1. Cela englobe d'effectuer ces deux vérifications sur chaque accès chez le client, l'appareil de mémorisation, ou les deux (comme applicable ; quand l'appareil de mémorisation est un serveur NFSv4.1, l'appareil de mémorisation est le responsable ultime du contrôle d'accès

comme décrit au paragraphe 13.9.2). Si une configuration pNFS effectue ces vérifications seulement chez le client, le risque qu'un client au mauvais comportement obtienne un accès non autorisé est une considération importante pour déterminer quand il est approprié d'utiliser une telle configuration pNFS. De tels types de disposition NE DEVRAIENT PAS être utilisés quand les vérifications d'accès seulement sur le client ne fournissent pas une assurance suffisante que le contrôle d'accès NFSv4.1 est appliqué correctement. (Ce n'est pas un problème pour le type de disposition de fichier décrit à la Section 13 parce que le protocole d'accès à la mémorisation pour LAYOUT4_NFSV4_1_FILES est NFSv4.1, et donc le modèle de sécurité pour l'accès à l'appareil de mémorisation via LAYOUT4_NFSv4_1_FILES est le même que celui du serveur de métadonnées.) Pour traiter le contrôle d'accès spécifique d'une disposition, le lecteur devrait examiner la spécification de disposition, comme la disposition fondée sur le fichier de NFSv4.1 (Section 13) de ce document, la disposition de blocs [RFC5663], et la disposition d'objets [RFC5664].

13. NFSv4.1 comme protocole de mémorisation dans pNFS : type de disposition de fichier

Cette section décrit la sémantique et le format des dispositions NFSv4.1 fondées sur le fichier pour pNFS. Les dispositions NFSv4.1 fondées sur le fichier utilisent le type de disposition LAYOUT4_NFSV4_1_FILES. Le type LAYOUT4_NFSV4_1_FILES définit un étalement des données sur plusieurs serveurs de données NFSv4.1.

13.1 Identifiant de client et considérations de session

Les sessions sont une caractéristique EXIGÉE de NFSv4.1, et cela s'étend aux deux serveurs de métadonnées et serveurs de données fondés sur le fichier (fondé sur NFSv4.1).

Le rôle que joue un serveur dans pNFS est déterminé par le résultat qu'il retourne de EXCHANGE_ID. Ces rôles sont :

* serveur de métadonnées (EXCHGID4_FLAG_USE_PNFS_MDS est établi dans le eir_flags résultant) ;
* serveur de données (EXCHGID4_FLAG_USE_PNFS_DS) ;
* serveur non de métadonnées (EXCHGID4_FLAG_USE_NON_PNFS). C'est un serveur NFSv4.1 qui ne prend pas en charge d'opérations (par exemple, LAYOUTGET) ou d'attributs qui relèvent de pNFS.

Le client PEUT demander zéro, un ou plusieurs de EXCHGID4_FLAG_USE_NON_PNFS, EXCHGID4_FLAG_USE_PNFS_DS, ou EXCHGID4_FLAG_USE_PNFS_MDS, même si certaines combinaisons (par exemple, EXCHGID4_FLAG_USE_NON_PNFS | EXCHGID4_FLAG_USE_PNFS_MDS) sont contradictoires. Cependant, le serveur DOIT seulement retourner les combinaisons acceptables suivantes :

Résultats acceptables de EXCHANGE_ID
EXCHGID4_FLAG_USE_PNFS_MDS
EXCHGID4_FLAG_USE_PNFS_MDS | EXCHGID4_FLAG_USE_PNFS_DS
EXCHGID4_FLAG_USE_PNFS_DS
EXCHGID4_FLAG_USE_NON_PNFS
EXCHGID4_FLAG_USE_PNFS_DS | EXCHGID4_FLAG_USE_NON_PNFS

Tableau 8

Comme l'implique le tableau ci-dessus, un serveur peut avoir un ou deux rôles. Un serveur peut être à la fois un serveur de métadonnées et un serveur de données, ou il peut être à la fois un serveur de données et serveur non de métadonnées. En plus de retourner deux rôles dans le résultat de EXCHANGE_ID, et donc de servir les deux rôles via un identifiant de client commun, un serveur peut servir deux rôles en retournant un unique identifiant de client et propriétaire de serveur pour chaque rôle dans chacun des deux résultats de EXCHANGE_ID, chaque résultat indiquant chaque rôle.

Dans le cas d'un serveur avec des rôles pNFS concurrents qui sont servis par un identifiant de client commun, si la demande EXCHANGE_ID provenant du client a zéro ou une combinaison de bits établis dans eia_flags, le résultat du serveur devrait établir les bits qui représentent la plus forte combinaison acceptable de rôles du serveur, avec une préférence pour correspondre aux rôles demandés par le client. Donc, si la demande d'un client a les fanions (EXCHGID4_FLAG_USE_NON_PNFS | EXCHGID4_FLAG_USE_PNFS_MDS | EXCHGID4_FLAG_USE_PNFS_DS) établis, et si le serveur est à la fois serveur de métadonnées et serveur de données, servant les deux rôles par un identifiant de client commun, le serveur DEVRAIT retourner avec (EXCHGID4_FLAG_USE_PNFS_MDS | EXCHGID4_FLAG_USE_PNFS_DS) établi.

Dans le cas d'un serveur qui a plusieurs rôles pNFS concurrents, chaque rôle servi par un identifiant de client unique, si le

client spécifie zéro ou une combinaison de rôles dans la demande, le résultat du serveur DEVRAIT retourner seulement un des rôles de la combinaison spécifiée par la demande du client. Si le rôle spécifié par le résultat du serveur ne correspond pas à l'utilisation prévue par le client, il devrait envoyer un EXCHANGE_ID spécifiant juste le rôle pNFS qui l'intéresse.

Si un client de métadonnées pNFS obtient une disposition qui le réfère à un serveur de données NFSv4.1, il a besoin d'un identifiant de client sur ce serveur de données. Si il n'a pas déjà un identifiant de client provenant du serveur qui avait le fanion EXCHGID4_FLAG_USE_PNFS_DS établi dans le résultat de EXCHANGE_ID, le client a alors besoin d'envoyer un EXCHANGE_ID au serveur de données, utilisant le même co_ownerid qu'envoyé au serveur de métadonnées, avec le fanion EXCHGID4_FLAG_USE_PNFS_DS établi dans les arguments. Si le résultat de l'EXCHANGE_ID du serveur a le fanion EXCHGID4_FLAG_USE_PNFS_DS établi, alors le client peut utiliser l'identifiant de client pour créer des sessions qui vont échanger des opérations de données pNFS. L'identifiant de client retourné par le serveur de données n'a pas de relation avec l'identifiant de client retourné par un serveur de métadonnées sauf si les identifiants de client sont égaux, et si les propriétaires de serveur et les portées de serveur du serveur de données et du serveur de métadonnées sont égaux.

Dans NFSv4.1, l'identifiant de session dans l'opération SEQUENCE implique l'identifiant de client, qui à son tour pourrait être utilisé par le serveur pour transposer l'identifiant d'état en la bonne paire de client/serveur. Cependant, quand un serveur de données est présenté avec une opération READ ou WRITE avec un identifiant d'état, parce que l'identifiant d'état est associé à un identifiant de client sur un serveur de métadonnées, et parce que l'identifiant de session dans l'opération SEQUENCE précédente est lié à l'identifiant de client du serveur de données, le serveur de données n'a pas de façon évidente de déterminer le serveur de métadonnées à partir de la procédure COMPOUND, et donc n'a pas de façon de valider l'identifiant d'état. Une approche RECOMMANDÉE est que les serveurs pNFS codent les informations d'acheminement et/ou d'identité de serveur de métadonnées dans les brides de fichier de serveur de données comme retournées dans la disposition.

Si les informations d'acheminement et/ou d'identité de serveur de métadonnées sont codées dans les brides de fichier de serveur de données, quand l'identité ou la localisation du serveur de métadonnées change, les brides de fichier de serveur de données qu'il a données sont devenues invalides (périmées) et donc le serveur de métadonnées DOIT d'abord rappeler les dispositions. Invalider la bride de fichier d'un serveur de données ne rend pas invalide l'antémémoire de données du client NFS. L'antémémoire du client devrait transposer la bride de fichier d'un serveur de données en bride de fichier de serveur de métadonnées, et une bride de fichier de serveur de métadonnées en données en antémémoire.

Si un serveur est à la fois serveur de métadonnées et serveur de données, le serveur pourrait avoir besoin de distinguer les opérations sur les fichiers qui sont dirigées sur le serveur de métadonnées de celles dirigées sur le serveur de données. Il est RECOMMANDÉ que les valeurs des brides de fichier retournées par l'opération LAYOUTGET soient différentes de la valeur de la bride de fichier retournée par le OPEN du même fichier.

Un autre scénario est que le serveur de métadonnées et l'appareil de mémorisation soient distincts du point de vue d'un client, et que les rôles soient inversés du point de vue d'un autre client. Par exemple, dans le modèle de système de fichiers en grappe, un serveur de métadonnées sur un client pourrait être un serveur de données sur un autre client. Si NFSv4.1 est utilisé comme protocole de mémorisation, alors les serveurs pNFS doivent coder les valeurs de brides de fichier en accord avec leurs rôles spécifiques.

13.1.1 Considérations de sessions pour les serveurs de données

Le paragraphe 2.10.11.2 déclare qu'un client doit s'occuper de renouveler son prêt afin d'empêcher qu'une session soit supprimée par le serveur. Si la réponse à EXCHANGE_ID a juste le rôle EXCHGID4_FLAG_USE_PNFS_DS établi, alors (comme noté au paragraphe 13.6) le client ne sera pas capable de déterminer l'attribut lease_time du serveur de données parce que GETATTR ne va pas être permis. À la place, la règle est que chaque fois qu'un client reçoit une disposition qui se réfère à un serveur de données qui retourne juste le rôle EXCHGID4_FLAG_USE_PNFS_DS, le client PEUT supposer que l'attribut lease_time provenant du serveur de métadonnées qui a retourné la disposition s'applique au serveur de données. Donc, le serveur de données DOIT connaître les valeurs de tous les attributs lease_time de tous les serveurs de métadonnées pour lesquels il fournit des entrées/sorties, et il DOIT utiliser le maximum de toutes ces valeurs de lease_time comme intervalle de prêt pour tous les identifiants de client et sessions établis sur lui.

Par exemple, si un serveur de métadonnées a un attribut lease_time de 20 secondes, et si un second serveur de métadonnées a un attribut lease_time de 10 secondes, alors si les deux serveurs retournent des dispositions qui se réfèrent à un serveur de données EXCHGID4_FLAG_USE_PNFS_DS seulement, le serveur de données DOIT renouveler le prêt d'un client si l'intervalle entre deux opérations SEQUENCE sur des demandes COMPOUND différentes est de moins de 20 secondes.

13.2 Définitions de dispositions de fichier

Les définitions suivantes s'appliquent au type de disposition LAYOUT4_NFSV4_1_FILES et peuvent être applicables aux autres types de disposition.

Unité : une unité est une quantité de taille fixe de données écrite à un serveur de données.

Schéma : un schéma est une méthode de distribution d'une ou plusieurs unités de taille égale sur un ensemble de serveurs de données. Un schéma est parcouru une ou plusieurs fois.

Bande : une bande est un ensemble de données distribuées sur un ensemble de serveurs de données dans un schéma avant que ce schéma se répète.

Compte de bandes : un compte de bandes est le nombre d'unités dans un schéma.

Largeur de bande : taille d'une bande en octets. Largeur de bande = compte de bande * taille d'unité de bande.

Ci-après, le présent document se réfère à une unité qui est écrite dans un schéma comme une "unité de bande".

Un schéma peut avoir plus d'unités de bande que les serveurs de données. Si il en est ainsi, certains serveurs de données vont avoir plus d'une unité de bande par bande. Un serveur de données qui a plusieurs unités de bande par bande PEUT mémoriser chaque unité dans un fichier de données différent (et selon la mise en œuvre, va éventuellement allouer une unique bride de fichier de données à chaque fichier de données).

13.3 Types de données de disposition de fichier

Les types de disposition NFSv4.1 généraux sont nfsv4_1_file_layouthint4, nfsv4_1_file_layout_ds_addr4, et nfsv4_1_file_layout4.

L'opération SETATTR prend en charge un attribut d'indication de disposition (paragraphe 5.12.4). Quand le client règle une indication de disposition (type de données layouthint4) avec un type de disposition de LAYOUT4_NFSV4_1_FILES (champ loh_type) le champ loh_body contient une valeur de type de données nfsv4_1_file_layouthint4.

```
const NFL4_UFLG_MASK                = 0x0000003F;
const NFL4_UFLG_DENSE               = 0x00000001;
const NFL4_UFLG_COMMIT_THRU_MDS     = 0x00000002;
const NFL4_UFLG_STRIPE_UNIT_SIZE_MASK = 0xFFFFFFC0;

typedef uint32_t nfl_util4 ;

enum filelayout_hint_care4 {
      NFLH4_CARE_DENSE              = NFL4_UFLG_DENSE,
      NFLH4_CARE_COMMIT_THRU_MDS    = NFL4_UFLG_COMMIT_THRU_MDS,
      NFLH4_CARE_STRIPE_UNIT_SIZE   = 0x00000040,
      NFLH4_CARE_STRIPE_COUNT       = 0x00000080
 };

/* Codé dans le champ loh_body de type de données layouthint4 : */

struct nfsv4_1_file_layouthint4 {
    uint32_t      nflh_care ;
    nfl_util4     nflh_util ;
    count4        nflh_stripe_count ;
};
```

La structure d'indication générique de disposition est décrite au paragraphe 3.3.19. Le client utilise l'indication de disposition dans l'attribut layout_hint (paragraphe 5.12.4) pour indiquer le type préféré de disposition à utiliser pour un fichier nouvellement créé. Le contenu LAYOUT4_NFSV4_1_FILES spécifique du type de disposition pour l'indication de disposition est composé de trois champs. Le premier champ, nflh_care, est un ensemble de fanions indiquant quelles valeurs de l'indication intéressent le client. Si le fanion NFLH4_CARE_DENSE est établi, alors le client indique dans le second champ, nflh_util, une préférence sur la façon dont les fichiers de données sont mis en paquets (paragraphe 13.4.4)

qui est contrôlé par la valeur de l'expression nflh_util & NFL4_UFLG_DENSE ("&" représente l'opérateur ET au bit près). Si le fanion NFLH4_CARE_COMMIT_THRU_MDS est établi, alors le client indique une préférence pour si le client devrait envoyer des opérations COMMIT au serveur de métadonnées ou au serveur de données (paragraphe 13.7) qui est contrôlée par la valeur de nflh_util & NFL4_UFLG_COMMIT_THRU_MDS. Si le fanion NFLH4_CARE_STRIPE_UNIT_SIZE est établi, le client indique sa taille préférée d'unité de bande, qui est indiquée dans nflh_util & NFL4_UFLG_STRIPE_UNIT_SIZE_MASK (donc, la taille d'unité de bande DOIT être un multiple de 64 octets). La taille minimum d'unité de bande est 64 octets. Si le fanion NFLH4_CARE_STRIPE_COUNT est établi, le client indique dans le troisième champ, nflh_stripe_count, le compte de bandes. Le compte de bandes multiplié par la taille de l'unité de bande est la largeur de bande.

Quand LAYOUTGET retourne une disposition LAYOUT4_NFSV4_1_FILES (indiquée dans le champ loc_type du champ lo_content) le champ loc_body du champ lo_content contient une valeur de type de données nfsv4_1_file_layout4. Entre autres contenus, nfsv4_1_file_layout4 a un identifiant d'appareil de mémorisation (champ nfl_deviceid) de type de données deviceid4. L'opération GETDEVICEINFO transpose un identifiant d'appareil en adresse d'appareil de mémorisation (type device_addr4). Quand GETDEVICEINFO retourne une adresse d'appareil avec un type de disposition de LAYOUT4_NFSV4_1_FILES (champ da_layout_type) le champ da_addr_body contient une valeur de type de données nfsv4_1_file_layout_ds_addr4.

```
typedef netaddr4 multipath_list4<>;

/* * Codé dans le champ da_addr_body de type de données device_addr4 : */
struct nfsv4_1_file_layout_ds_addr4 {
    uint32_t          nflda_stripe_indices<>;
    multipath_list4   nflda_multipath_ds_list<>;
};
```

Le type de données nfsv4_1_file_layout_ds_addr4 représente l'adresse de l'appareil. Il est composé de deux champs :

1. nflda_multipath_ds_list : dispositif de listes de serveurs de données, où chaque liste peut être d'un ou plusieurs éléments, et chaque élément représente une adresse de serveur de données qui peut servir également de cible des opérations d'entrée/sortie (voir le paragraphe 13.5). La longueur de ce dispositif pourrait être différente du compte de bandes.
2. nflda_stripe_indices : dispositif d'indices utilisé pour indexer dans nflda_multipath_ds_list. La valeur de chaque élément de nflda_stripe_indices DOIT être inférieure au nombre des éléments dans nflda_multipath_ds_list. Chaque élément de nflda_multipath_ds_list DEVRAIT être référencé par un ou plusieurs éléments de nflda_stripe_indices. Le nombre des éléments dans nflda_stripe_indices est toujours égal au compte de bandes.

```
/* * Codé dans le champ loc_body de type de données layout_content4 : */
struct nfsv4_1_file_layout4 {
    deviceid4     nfl_deviceid ;
    nfl_util4     nfl_util ;
    uint32_t      nfl_first_stripe_index ;
    offset4       nfl_pattern_offset ;
    nfs_fh4       nfl_fh_list<> ;
};
```

Le type de données nfsv4_1_file_layout4 représente la disposition. Il est composé des champs suivants :

1. nfl_deviceid : identifiant d'appareil qui se transpose en une valeur de type nfsv4_1_file_layout_ds_addr4.

2. nfl_util : comme le champ nflh_util de type de données nfsv4_1_file_layouthint4, c'est une représentation compacte de la façon dont sont mises en paquet les données d'un fichier sur chaque serveur de données, si le client devrait envoyer des opérations COMMIT au serveur de métadonnées ou au serveur de données, et la taille de l'unité de bande. Si un serveur retourne deux dispositions en chevauchement ou plus, chaque taille d'unité de bande de chaque disposition en chevauchement DOIT être la même.

3. nfl_first_stripe_index : indice du premier élément du dispositif nflda_stripe_indices à utiliser.

4. nfl_pattern_offset : ce champ est le décalage logique dans le fichier où commence le schéma de mise en bandes. Il est nécessaire pour convertir le décalage logique d'entrée/sortie du client (par exemple, le décalage courant dans un descripteur de fichier POSIX avant que l'appel système read() ou write() soit envoyé) en le nombre d'unités de bande (voir le paragraphe 13.4.1).

Si la mise en paquet dense est utilisée, alors nfl_pattern_offset est aussi nécessaire pour convertir le décalage logique d'entrée/sortie du client en un décalage sur le fichier au serveur de données correspondant au numéro d'unité de bande (voir le paragraphe 13.4.4).

Noter que nfl_pattern_offset n'est pas toujours le même que lo_offset. Par exemple, via l'opération LAYOUTGET, un client pourrait demander une disposition commençant au décalage 1000 d'un fichier qui a son schéma de mise en bandes qui commence au décalage zéro.

5. nfl_fh_list : un dispositif de brides de fichier de serveur de données pour chaque liste de serveurs de données dans chaque élément du dispositif nflda_multipath_ds_list. Le nombre d'éléments dans nfl_fh_list dépend de si la mise en paquets éparse ou dense est utilisée.
   * Si la mise en paquets éparse est utilisée, le nombre d'éléments dans nfl_fh_list DOIT être une des trois valeurs :
   - Zéro : cela signifie que les brides de fichier utilisées pour chaque serveur de données sont les mêmes que la bride de fichier retournée par l'opération OPEN du serveur de métadonnées.
   - Un : cela signifie que chaque serveur de données utilise la même bride de fichier : ce qui est spécifié dans nfl_fh_list[0].
   - Le même nombre d'éléments que dans nflda_multipath_ds_list. Donc, dans ce cas, quand elle envoie une opération d'entrée/sortie à tout serveur de données dans nflda_multipath_ds_list[X], la bride de fichier dans nfl_fh_list[X] DOIT être utilisée.
     Voir la discussion sur la mise en paquets éparse au paragraphe 13.4.4.
   * Si la mise en paquets dense est utilisée, le nombre d'éléments dans nfl_fh_list DOIT être le même que celui dans nflda_stripe_indices. Donc, quand elle envoie une opération d'entrée/sortie à tout serveur de données dans nflda_multipath_ds_list[nflda_stripe_indices[Y]], la bride de fichier dans nfl_fh_list[Y] DOIT être utilisée. De plus, chaque fois qu'il existe i et j, (i != j) tels que l'intersection de nflda_multipath_ds_list[nflda_stripe_indices[i]] et de nflda_multipath_ds_list[nflda_stripe_indices[j]] n'est pas vide, alors nfl_fh_list[i] NE DOIT PAS être égal à nfl_fh_list[j]. En d'autres termes, quand la mise en paquets dense est utilisée, si un serveur de données apparaît dans deux unités d'un schéma de mise en bandes ou plus, chaque référence au serveur de données DOIT utiliser une bride de fichier différente.

     Bien sûr, si il y a plusieurs schémas de mise en bandes, comme indiqué par la présence de plusieurs objets de type de données layout4 (retournés dans une ou plusieurs opérations LAYOUTGET) et si un serveur de données est la cible d'une unité d'un schéma et d'une autre unité d'un autre schéma, alors chaque référence à chaque serveur de données DOIT utiliser une bride de fichier différente.

     Voir la discussion sur la mise en paquets dense au paragraphe 13.4.4.

Les détails sur l'interprétation de la disposition sont au paragraphe 13.4.

13.4 Interprétation de la disposition de fichier

13.4.1 Détermination du nombre d'unités de bande

Pour trouver le nombre d'unités de bande qui correspond au décalage logique de fichier du client, le décalage de schéma va aussi être utilisé. La ième unité de bande (SUi) est :

décalage relatif = décalage de fichier - nfl_décalage de schéma ;
SUi = plancher(décalage relatif / taille d'unité de bande) ;

13.4.2 Interprétation de la disposition de fichier en utilisant la mise en paquets éparse

Quand la mise en paquets éparse est utilisée, l'algorithme pour déterminer la bride de fichier et régler les adresses réseau de serveur de données à écrire à l'unité de bande i (SUi) est :

compte_de_bandes = nombre d'éléments dans nflda_stripe_indices;
j = (SUi + nfl_first_stripe_index) % compte_de_bandes ;
idx = nflda_stripe_indices[j] ;
fh_count = nombre d'éléments dans nfl_fh_list ;
ds_count = nombre d'éléments dans nflda_multipath_ds_list ;

```
switch (fh_count) {
      cas ds_count : fh = nfl_fh_list[idx];
        coupure ;
      cas 1 :        fh = nfl_fh_list[0];
        coupure ;
      cas 0 :        fh = bride de fichier retournée par OPEN;
        coupure ;
      défaut :        lance une exception fatale ;
        coupure ;
    }

    address_list = nflda_multipath_ds_list[idx] ;
```

Le client va alors choisir un serveur de données dans address_list, et envoyer une opération READ ou WRITE en utilisant la bride de fichier spécifiée dans fh.

Considérons l'exemple suivant :
Supposons qu'on ait une adresse d'appareil consistant en sept serveurs de données, arrangés en trois classes d'équivalence (paragraphe 13.5) : { A, B, C, D }, { E }, { F, G } où A à G sont des adresses réseau.
Alors, nflda_multipath_ds_list<> = { A, B, C, D }, { E }, { F, G }
c'est-à-dire,
nflda_multipath_ds_list[0] = { A, B, C, D }
nflda_multipath_ds_list[1] = { E }
nflda_multipath_ds_list[2] = { F, G }

Supposons que le dispositif d'indices de bande soit : nflda_stripe_indices<> = { 2, 0, 1, 0 }
Supposons maintenant que le client obtienne une disposition qui a un identifiant d'appareil se transposant en l'adresse d'appareil ci-dessus. L'indice initial contient nfl_first_stripe_index = 2,
et la liste de brides de fichier est nfl_fh_list = { 0x36, 0x87, 0x67 }.
Si le client veut écrire à SU0, l'ensemble de combinaisons { adresse réseau, bride de fichier } valides pour SUi est déterminé par : nfl_first_stripe_index = 2
Donc,
idx = nflda_stripe_indices[(0 + 2) % 4]
= nflda_stripe_indices[2]
= 1
Donc
nflda_multipath_ds_list[1] = { E }
et
nfl_fh_list[1] = { 0x87 }

Le client peut donc écrire SU0 à { 0x87, { E } }.

Les destinations des 13 premières unités de mémorisation sont :

SUi	Bride de fichier	Serveurs de données |
0	87	E
1	36	A,B,C,D
2	67	F,G
3	36	A,B,C,D
4	87	E
5	36	A,B,C,D
6	67	F,G
7	36	A,B,C,D
8	87	E
9	36	A,B,C,D
11	36	A,B,C,D
12	87	E

Tableau 9

13.4.3 Interprétation de la disposition de fichier en utilisant la mise en paquets dense

Quand la mise en paquets dense est utilisée, l'algorithme pour déterminer la bride de fichier et l'ensemble d'adresses réseau de serveur de données pour écrire l'unité de bande i (SUi) est :

```
compte_de_bandes = nombre d'éléments dans nflda_stripe_indices ;
j = (SUi + nfl_first_stripe_index) % compte_de_bandes ;
idx = nflda_stripe_indices[j] ;
fh_count = nombre d'éléments dans nfl_fh_list ;
ds_count = nombre d'éléments dans nflda_multipath_ds_list ;

switch (fh_count) {
      cas compte_de_bandes : fh = nfl_fh_list[j];
        coupure ;

      défaut : poser une exception fatale ;
        coupure ;
    }

    liste_d'adresses = nflda_multipath_ds_list[idx] ;
```

Le client va alors choisir un serveur de données dans la liste d'adresses, et envoyer une opération READ ou WRITE en utilisant la bride de fichier spécifiée dans fh.

Considérons l'exemple suivant (qui est le même que dans l'exemple de mise en paquets éparse, sauf pour la liste de brides de fichier) :

Supposons qu'on ait une adresse d'appareil consistant en sept serveurs de données, arrangés en trois classes d'équivalence (paragraphe 13.5) : { A, B, C, D }, { E }, { F, G } où A à G sont des adresses réseau.

Alors, nflda_multipath_ds_list<> = { A, B, C, D }, { E }, { F, G }
c'est-à-dire,
nflda_multipath_ds_list[0] = { A, B, C, D }
nflda_multipath_ds_list[1] = { E }
nflda_multipath_ds_list[2] = { F, G }

Supposons que le dispositif d'indice de mise en bandes soit : nflda_stripe_indices<> = { 2, 0, 1, 0 }

Supposons maintenant que le client obtienne une disposition qui a un identifiant d'appareil se transposant en l'adresse d'appareil ci-dessus. L'indice initial contient nfl_first_stripe_index = 2,
et
nfl_fh_list = { 0x67, 0x37, 0x87, 0x36 }.

Les exemples intéressants pour la mise en paquets dense sont SU1 et SU3 parce que chaque unité de bande se réfère à la même liste de serveurs de données, et chaque unité de bande DOIT utiliser une bride de fichier différente. Si le client veut écrire à SU1, l'ensemble de combinaisons { adresse réseau, bride de fichier } valides pour SUi est déterminé par :

nfl_first_stripe_index = 2

Donc
j = (1 + 2) % 4 = 3
idx = nflda_stripe_indices[j]
= nflda_stripe_indices[3]
= 0

Donc
nflda_multipath_ds_list[0] = { A, B, C, D }
et
nfl_fh_list[3] = { 0x36 }

Le client peut donc écrire SU1 à { 0x36, { A, B, C, D } }.

Pour SU3, j = (3 + 2) % 4 = 1, et nflda_stripe_indices[1] = 0. Alors nflda_multipath_ds_list[0] = { A, B, C, D }, et nfl_fh_list[1] = 0x37. Le client peut donc écrire SU3 à { 0x37, { A, B, C, D } }.

Les destinations des 13 premières unités de mémorisation sont :

SUi	bride de fichier	serveurs de données
0	87	E
1	36	A,B,C,D
2	67	F,G
3	37	A,B,C,D
4	87	E
5	36	A,B,C,D
6	67	F,G
7	37	A,B,C,D
8	87	E
9	36	A,B,C,D
10	67	F,G
11	37	A,B,C,D
12	87	E

Tableau 10

13.4.4 Mise en paquets d'unité de surface éparse et dense

Le fanion NFL4_UFLG_DENSE du type de données nfl_util4 (champ nflh_util du type de données nfsv4_1_file_layouthint4 et champ nfl_util de type de données nfsv4_1_file_layout_ds_addr4) spécifie comment les données sont mises en paquet dans les fichiers de données sur un serveur de données. Il permet deux mises en paquet différentes des données : éparse et dense. Le type de mise en paquet détermine le calcul qui va être fait pour transposer le décalage de fichier visible par le client en le décalage dans le fichier de données situé sur le serveur de données.

Si nfl_util & NFL4_UFLG_DENSE est zéro, cela signifie que la mise en paquets éparse est utilisée. Donc, les décalages logiques de fichier vus par un client qui envoie des READ et des WRITE directement au serveur de métadonnées sont les mêmes décalages que chaque serveur de données utilise quand il mémorise une unité de bande. L'effet alors, pour les schémas de mise en bandes consistant en au moins deux unités de bande, est pour chaque fichier de serveur de données d'être épars ou "à trous". Ainsi par exemple, supposons qu'il y ait un schéma avec trois unités de bandes, la taille d'unité de bande est de 4096 octets, et il y a trois serveurs de données dans le schéma. Alors, le fichier dans le serveur de données 1 va avoir les unités de bande 0, 3, 6, 9, ... remplies, le fichier du serveur de données 2 va avoir les unités de bande 1, 4, 7, 10, ... remplies, et le fichier du serveur de données 3 va avoir les unités de bande 2, 5, 8, 11, ... remplies. Les unités de bande non remplies de chaque fichier vont être des trous ; donc, les fichiers dans chaque serveur de données sont épars.

Si la mise en paquets éparse est utilisée et si un client tente une entrée/sortie sur un des trous, alors une erreur DOIT être retournée par le serveur de données. En utilisant l'exemple ci-dessus, si le serveur de données 3 a reçu une opération READ ou WRITE pour le bloc 4, le serveur de données va retourner NFS4ERR_PNFS_IO_HOLE. Donc, les serveurs de données doivent comprendre le schéma de mise en bandes afin de prendre en charge la mise en paquets éparse.

Si nfl_util & NFL4_UFLG_DENSE est un, cela signifie que la mise en paquets dense est utilisée, et que les fichiers du serveur de données n'ont pas de trous. La mise en paquets dense pourrait être choisie parce que le serveur de données ne prend pas (efficacement) en charge les fichiers à trous ou parce que le serveur de données ne peut reconnaître la lecture vers l'avant que si il n'y a pas de trous. Si la mise en paquets dense est indiquée dans la disposition, les fichiers de données vont être mis en paquets. En utilisant le même schéma de mise en bandes et la taille d'unité de bande qui ont été utilisés pour l'exemple de mise en paquets éparse, l'exemple correspondant de mise en paquets dense aurait toutes les unités de bande de tous les fichiers de données remplis comme suit :

* les unités de bande logique 0, 3, 6, ... du fichier vont être sur les unités de bande 0, 1, 2, ... du fichier de serveur de données 1,
* les unités de bande logique 1, 4, 7, ... du fichier vont être sur les unités de bande 0, 1, 2, ... du fichier de serveur de données 2,
* les unités de bande logique 2, 5, 8, ... du fichier vont être sur les unités de bande 0, 1, 2, ... du fichier de serveur de données 3.

Parce que la mise en paquets dense ne laisse pas de trous aux serveurs de données, il est permis au client pNFS d'écrire à tout décalage de tout fichier de données de tout serveur de données dans la bande. Donc, les serveurs de données n'ont pas besoin de connaître le schéma de mise en bandes du fichier.

Le calcul pour déterminer le décalage d'octets au sein des fichiers de données pour les dispositions de serveur de données dense est :

largeur_de_bande = taille_d'unité_de_bande * N ; ; N = nombre d'éléments dans nflda_stripe_indices.
décalage_relatif = décalage_fichier - décalage_schéma_nfl ;
décalage_fichier_données = plancher(décalage_relatif / largeur_de_bande) * taille_d'unité_de_bande + décalage_relatif %
taille_d'unité_de_bande

Si la mise en paquets dense est utilisée, et si un serveur de données apparaît plus d'une fois dans un schéma de mise en bandes, alors pour distinguer une unité de bande d'une autre, le serveur de données DOIT utiliser une bride de fichier différente. Supposons qu'on ait deux serveurs de données. Les unités de bande logiques 0, 3, 6 sont servies par le serveur de données 1, les unités de bande logiques 1, 4, 7 sont servies par le serveur de données 2, et les unités de bande logiques 2, 5, 8 sont aussi servies par le serveur de données 2. Sauf si le serveur de données 2 a deux brides de fichier (chacune se référant à un fichier de données différent) alors, par exemple, une écriture à l'unité de bande logique 1 écrase l'écriture à l'unité de bande logique 2 parce que les deux unités de bande logiques sont situées dans la même unité de bande (0) du serveur de données 2.

13.5 Serveur de données multi chemins

La disposition de fichier NFSv4.1 prend en charge le multi chemins à plusieurs adresses de serveur de données. Le multi chemins de niveau serveur de données est utilisé pour l'adaptation de la bande passante via le multiplexage (paragraphe 2.10.5) et pour une plus haute disponibilité d'utilisation dans le cas de défaillance du serveur de données. Le multi chemins permet au client de passer à une autre adresse de serveur de données qui peut être celle d'un autre serveur de données qui exporte la même unité de bande de données, sans avoir à contacter le serveur de métadonnées pour une nouvelle disposition.

Pour prendre en charge le multi chemins de serveur de données, chaque élément de la nflda_multipath_ds_list contient un dispositif de une adresse réseau de serveur de données de plus. Ce dispositif (type de données multipath_list4) représente une liste de serveurs de données (chacun identifié par une adresse réseau) avec la possibilité que certains serveurs de données apparaissent plusieurs fois dans la liste.

Le client est libre d'utiliser toute adresse réseau comme destination pour envoyer des demandes au serveur de données. Si des adresses réseau sont des chemins moins optimaux que d'autres pour les données, alors le MDS NE DEVRAIT PAS inclure ces adresses réseau dans un élément de nflda_multipath_ds_list. Si des adresses réseau moins optimales existent pour fournir la reprise sur défaillance, la méthode RECOMMANDÉE pour offrir les adresses est de les fournir dans une transposition d'identifiant d'appareil de remplacement à adresse d'appareil, ou un identifiant d'appareil de remplacement. Quand un client trouve qu'aucun serveur de données ne répond dans un élément de nflda_multipath_ds_list, il DEVRAIT envoyer un GETDEVICEINFO pour tenter de remplacer les transpositions existantes d'identifiant d'appareil à adresse d'appareil. Si le MDS détecte que tous les serveurs de données représentés par un élément de nflda_multipath_ds_list sont indisponibles, le MDS DEVRAIT envoyer un CB_NOTIFY_DEVICEID (si le client a indiqué qu'il veut des notifications d'identifiant d'appareil pour les identifiants d'appareil changés) pour changer les transpositions d'identifiant d'appareil à adresse d'appareil pour les serveurs de données disponibles. Si l'identifiant d'appareil lui-même va être remplacé, le MDS DEVRAIT rappeler toutes les dispositions avec l'identifiant d'appareil, et donc forcer le client à obtenir de nouvelles dispositions et transpositions d'identifiant d'appareil via LAYOUTGET et GETDEVICEINFO.

Généralement, si deux adresses réseau apparaissent dans un élément de nflda_multipath_ds_list, elles vont désigner le même serveur de données, et les deux adresses de serveur de données vont prendre en charge la mise en œuvre d'identifiant de client ou de multiplexage de session (cette dernière est RECOMMANDÉE) comme défini au paragraphe 2.10.5. Les deux adresses de serveur de données vont partager le même propriétaire de serveur ou identifiant majeur du propriétaire de serveur. Il n'est pas toujours nécessaire que les deux adresses de serveur de données désignent le même serveur quand le multiplexage est utilisé. Par exemple, les données pourrait être en lecture seule, et les données consister en répliques exactes.

13.6 Opérations envoyées aux serveurs de données NFSv4.1

Les clients qui accèdent aux données sur un serveur de données NFSv4.1 DOIVENT envoyer seulement la procédure NULL et les procédures COMPOUND dont les opérations sont prises seulement de deux sous ensembles restreints des opérations définies comme opérations NFSv4.1 valides. Les clients DOIVENT utiliser la bride de fichier spécifiée par la disposition quand ils accèdent aux données sur les serveurs de données NFSv4.1.

Le premier de ces sous ensembles d'opérations consiste en opérations de gestion. Ce sous ensemble consiste en les opérations BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CREATE_SESSION, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, SECINFO_NO_NAME, SET_SSV, et SEQUENCE. Le client peut utiliser ces opérations afin d'établir et maintenir les identifiants de client, sessions, et contextes de sécurité appropriés impliqués dans la communication avec le serveur de données. À partir d'ici, on va les appeler des opérations de ménage de serveur de données.

Le second sous ensemble consiste en les opérations COMMIT, READ, WRITE, et PUTFH. Ces opérations DOIVENT être utilisées avec une bride de fichier courante spécifiée par la disposition. Dans le cas de PUTFH, la nouvelle bride de fichier courante DOIT être prise dans la disposition. À partir d'ici, on va les appeler des opérations d'entrée/sortie de serveur de données. Comme décrit au paragraphe 12.5.1, un client NE DOIT PAS envoyer une entrée/sortie à un serveur de données pour lequel il ne détient pas une disposition valide ; le serveur de données DOIT rejeter une telle entrée/sortie.

Sauf si le serveur a une personnalité concurrente de serveur non de données -- c'est-à-dire, si le résultat de l'EXCHANGE_ID retourné (EXCHGID4_FLAG_USE_PNFS_DS | EXCHGID4_FLAG_USE_PNFS_MDS) ou (EXCHGID4_FLAG_USE_PNFS_DS | EXCHGID4_FLAG_USE_NON_PNFS) voir le paragraphe 13.1 -- toute tentative d'utilisation d'opérations sur un serveur de données autre que celui spécifié dans les deux sous ensembles ci-dessus DOIT retourner NFS4ERR_NOTSUPP au client.

Quand le serveur a des personnalités concurrentes de serveur de données et de serveur non de données, chaque COMPOUND envoyé par le client DOIT être construit de façon à ce qu'il soit approprié à une des deux personnalités, et il NE DOIT PAS contenir d'opérations dirigées sur un mélange de ces personnalités. Le serveur DOIT appliquer cela. Pour comprendre les contraintes, les opérations dans un COMPOUND sont divisées en les trois classes suivantes :

1. Une opération ambiguë à l'égard de l'allocation de personnalité. Cela inclut tous les opérations de ménage de serveur de données. De plus, si le serveur a des brides de fichier allouées de telle sorte que celles définies par la disposition sont les mêmes que celles utilisées par le serveur de métadonnées, toutes les opérations qui utilisent ces brides de fichier sont dans cette classe, avec l'exception suivante que si l'opération utilise un identifiant d'état incompatible avec la personnalité de serveur de données (par exemple, un identifiant d'état spécial ou l'identifiant d'état a un champ "seqid" non à zéro, voir le paragraphe 13.9.1) l'opération est de classe 3, comme décrit ci-dessous. Un COMPOUND contenant plusieurs opérations de classe 1 (et opérations d'aucune autre classe) PEUT être envoyé à un serveur avec plusieurs personnalités concurrentes de serveur de données et de serveur non de données.

2. Une opération qui se réfère sans ambiguïté à la personnalité de serveur de données. Cela inclut des opérations d'entrée/sortie de serveur de données où la bride de fichier est une qui peut seulement être validement dirigée sur la personnalité de serveur de données.

3. Une opération qui se réfère sans ambiguïté à la personnalité de serveur non de données. Cela inclut toutes les opérations COMPOUND qui ne sont ni des opérations de ménage de serveur de données ni des opérations d'entrée/sortie de serveur de données, plus des opérations d'entrée/sortie de serveur de données où la bride de fichier courante (ou celle qui va devenir courante dans le cas de PUTFH) est seulement valide sur le serveur de métadonnées ou où un identifiant d'état est utilisé qui est incompatible avec le serveur de données, c'est-à-dire, est un identifiant d'état spécial ou a une valeur d'identifiant de séquence non à zéro.

Quand un COMPOUND exécute d'abord une opération de classe 3, il agit comme un COMPOUND normal sur tout autre serveur, et la personnalité de serveur de données cesse d'être pertinente. Il n'y a pas de restriction particulière sur les opérations dans COMPOUND pour les limiter à celles d'un serveur de données. Quand un PUTFH est fait, les brides de fichier déduites de la disposition ne sont pas valides. Si leur format n'est pas normalement acceptable, alors NFS4ERR_BADHANDLE DOIT en résulter. De même, les brides de fichier courantes pour les autres opérations n'acceptent pas les brides de fichier déduites de dispositions et ne sont normalement pas utilisables sur le serveur de métadonnées. Les utiliser résulterait en NFS4ERR_STALE.

Quand un COMPOUND exécute d'abord une opération de classe 2, qui serait un PUTFH où la bride de fichier serait une provenant d'une disposition, le COMPOUND est alors interprété par rapport à la personnalité de serveur de données. Les

opérations en dehors des deux classes discutées ci-dessus DOIVENT résulter en NFS4ERR_NOTSUPP. Les brides de fichier sont validées en utilisant les règles de serveur de données, résultant en NFS4ERR_BADHANDLE et/ou NFS4ERR_STALE même quand il ne le feraient normalement pas quand elles sont adressées à une personnalité de serveur non de données. Les identifiants d'état doivent respecter les règles de serveur de données en ce que toute utilisation d'identifiant d'état spéciaux ou d'identifiants d'état avec des valeurs d'identifiant de séquence non à zéro doit résulter en NFS4ERR_BAD_STATEID.

Tant que le serveur n'a pas d'abord exécuté une opération de classe 2 ou 3, le client NE DOIT PAS dépendre de l'opération exécutée par la personnalité de serveur de données ou de serveur non de données. Le serveur DOIT prendre une personnalité cohérente pour un COMPOUND donné, avec la seule transition possible de quand la première opération de classe 2 ou de classe 3 est exécutée.

À cause de la complexité induite par l'allocation de brides de fichier telle qu'elles puissent être utilisées à la fois sur un serveur de données et un serveur de métadonnées, il est RECOMMANDÉ que lorsque le même serveur peut avoir les deux personnalités, le serveur alloue une bride de fichier séparée unique aux deux personnalités. Cela rend non ambigu à quel serveur une demande donnée est destinée.

GETATTR et SETATTR DOIVENT être dirigés sur le serveur de métadonnées. Dans le cas d'un SETATTR de l'attribut de taille, le protocole de contrôle est responsable de la propagation des mises à jour/troncatures de taille aux serveurs de données. Dans le cas d'extension de WRITE aux serveurs de données, la nouvelle taille doit être visible sur le serveur de métadonnées une fois qu'un LAYOUTCOMMIT s'est achevé (voir le paragraphe 12.5.4.2). Le paragraphe 13.10 décrit le mécanisme par lequel le client va traiter les fichiers du serveur de données qui ne reflètent pas la taille du serveur de métadonnées.

13.7 COMMIT sur serveur de métadonnées

La disposition de fichier fournit deux moyens alternatifs de fournir l'engagement de données écrites à travers des serveurs de données. Le fanion NFL4_UFLG_COMMIT_THRU_MDS dans le champ nfl_util de la disposition de fichier (type de données nfsv4_1_file_layout4) est une indication provenant du serveur de métadonnées au client de la façon EXIGÉE d'effectuer le COMMIT, en envoyant le COMMIT au serveur de données ou au serveur de métadonnées. Ces deux méthodes de traiter la question correspondent aux styles de mise en œuvre pour un serveur pNFS qui prend en charge le type de disposition de fichier.

* Quand le fanion est FAUX, les opérations COMMIT DOIVENT être envoyées au serveur de données auquel les opérations WRITE correspondantes ont été envoyées. Cette approche est parfois utile quand la mise en bande de fichier est mise en œuvre dans le serveur pNFS (au lieu du système de fichiers) avec chaque serveur de données individuel qui met en œuvre son propre système de fichiers.

* Quand le fanion est VRAI, les opérations COMMIT DOIVENT être envoyées au serveur de métadonnées, plutôt que aux serveurs de données individuels. Cette approche est parfois utile quand la mise en bande de fichier est mise en œuvre dans le système de fichiers mis en grappes qui est l'extrémité arrière du serveur pNFS. Dans de telles mises en œuvre, chaque COMMIT à chaque serveur de données pourrait résulter en écritures répétées de blocs de métadonnées au détriment des performances d'écriture. L'envoi d'un seul COMMIT au serveur de métadonnées peut être plus efficace quand il existe un système de fichiers en grappes capable de mettre en œuvre un tel COMMIT coordonné.

Si nfl_util & NFL4_UFLG_COMMIT_THRU_MDS est VRAI, alors afin de maintenir le modèle NFSv4.1 courant d'engagement et de récupération, les serveurs de données DOIVENT retourner un vérificateur writeverf commun dans toutes les réponses WRITE pour une disposition de fichier donnée, et la mise en œuvre de COMMIT du serveur de métadonnées doit retourner le même writeverf. La valeur du vérificateur writeverf DOIT être changée au serveur de métadonnées ou tout serveur de données qui est référencé dans la disposition, chaque fois que il y a un événement de serveur qui peut éventuellement conduire à la perte de données non engagées. La portée du vérificateur peut être pour un fichier ou pour le serveur pNFS entier. Il pourrait être plus difficile au serveur de maintenir le vérificateur au niveau du fichier, mais l'avantage est que seuls les événements qui impactent un certain fichier vont exiger une action de récupération.

Noter que si la disposition spécifie la mise en paquets dense, alors le décalage utilisé pour un COMMIT au MDS peut différer d'un décalage utilisé pour un COMMIT au serveur de données.

Le seul COMMIT au serveur de métadonnées va retourner un vérificateur, et le client devrait le comparer à tous les vérificateurs provenant des WRITE et faire échouer le COMMIT si il y a des vérificateurs discordants. Si le COMMIT au serveur de métadonnées échoue, le client devrait envoyer à nouveau des WRITE pour toutes les données modifiées dans le

fichier. Le client devrait traiter les données modifiées avec un vérificateur discordant comme une défaillance de WRITE et essayer d'en récupérer en envoyant à nouveau les WRITE au serveur de données original ou en utilisant un autre chemin vers ces données si la disposition n'a pas été rappelée. Autrement, le client peut obtenir une nouvelle disposition ou il pourrait réécrire directement les données sur le serveur de métadonnées. Si nfl_util & NFL4_UFLG_COMMIT_THRU_MDS est FAUX, envoyer un COMMIT au serveur de métadonnées pourrait n'avoir pas d'effet. Si nfl_util & NFL4_UFLG_COMMIT_THRU_MDS est FAUX, un COMMIT envoyé au serveur de métadonnées devrait être utilisé seulement pour engager des données qui ont été écrites au serveur de métadonnées. Voir les options de récupération au paragraphe 12.7.6.

13.8 Disposition iomode

La disposition iomode n'a pas besoin d'être utilisée par le serveur de métadonnées quand il sert des dispositions NFSv4.1 fondées sur le fichier, bien que dans certaines circonstances, cela puisse être utile. Par exemple, si la mise en œuvre de serveur prend en charge la lecture provenant de répliques ou miroirs en lecture seule, il serait utile que le serveur retourne une disposition permettant au client de le faire. À ce titre, le client DEVRAIT régler le iomode sur la base de son intention de lire ou écrire les données. Le client peut par défaut avoir un iomode de LAYOUTIOMODE4_RW. Le iomode n'a pas besoin d'être vérifié par les serveurs de données quand les clients effectuent des entrées/sorties. Cependant, les serveurs de données DEVRAIENT quand même valider que le client détient une disposition valide et sinon retourner une erreur.

13.9 Coordination d'état de serveur de métadonnées et de données

13.9.1 Exigences globales pour le stateid

Quand le client envoie une entrée/sortie à un serveur de données, l'identifiant d'état utilisé NE DOIT PAS être un identifiant d'état de disposition retourné par LAYOUTGET ou envoyé par CB_LAYOUTRECALL. Les identifiants d'état permis sont fondés sur un des suivants : un identifiant d'état OPEN (le champ d'identifiant d'état de type de données OPEN4resok comme retourné par OPEN) un identifiant d'état de délégation (le champ d'identifiant d'état des types de données open_read_delegation4 et open_write_delegation4 comme retourné par OPEN ou WANT_DELEGATION, ou comme envoyé par CB_PUSH_DELEG) ou un identifiant d'état retourné par les opérations LOCK ou LOCKU. L'identifiant d'état envoyé au serveur de données DOIT être envoyé avec le seqid réglé à zéro, indiquant la version la plus courante de cet identifiant d'état, plutôt que d'indiquer une valeur de seqid spécifique différente de zéro. En aucun cas l'utilisation de valeurs spéciales d'identifiant d'état n'est permise.

L'identifiant d'état utilisé pour l'entrée/sortie DOIT avoir le même effet et être soumis à la même validation sur le serveur de données que si l'entrée/sortie était effectuée sur le serveur de métadonnées lui-même en l'absence de pNFS. Ceci implique que les identifiants d'état sont globalement valides sur les deux serveurs de métadonnées et de données. Cela exige que le serveur de métadonnées propage les changements dans l'état LOCK et OPEN aux serveurs de données, afin que ceux-ci puissent valider les accès d'entrée/sortie. Ceci est discuté plus en détails au paragraphe 13.9.2. Selon le moment où sont propagés les identifiants d'état, l'existence d'un identifiant d'état valide sur le serveur de données peut agir comme preuve d'une disposition valide.

Les clients qui effectuent des opérations d'entrée/sortie ont besoin de choisir un identifiant d'état approprié à base de verrous (incluant des ouvertures et des délégations) détenus par le client et les divers types de propriétaires d'état qui envoient des demandes d'entrée/sortie. Les règles pour le faire quand ils référencent des serveurs de données sont un peu différentes de celles discutées au paragraphe 8.2.5, qui s'appliquent quand on accède aux serveurs de métadonnées.

Les règles suivantes, appliquées en ordre de priorité décroissante, gouvernent le choix de l'identifiant d'état approprié :

* Si le client détient une délégation pour le fichier en question, l'identifiant d'état de délégation devrait être utilisé.
* Autrement, il doit y avoir un identifiant d'état OPEN pour le propriétaire d'ouverture actuel, et cet identifiant d'état OPEN pour le fichier ouvert en question est utilisé, sauf si le verrouillage obligatoire l'empêche. Voir ci-dessous.
* Si le serveur de données avait précédemment répondu avec NFS4ERR_LOCKED à l'utilisation de l'identifiant d'état OPEN, alors le client devrait utiliser l'identifiant d'état de verrou de gamme d'octets chaque fois qu'il en existe un pour ce fichier ouvert avec le propriétaire de verrou courant.
* Les identifiants d'état spéciaux ne devraient jamais être utilisés. Si ils sont utilisés, le serveur de données DOIT rejeter l'entrée/sortie avec une erreur NFS4ERR_BAD_STATEID.

13.9.2 Propagation de l'état de serveur de données

Comme le serveur de métadonnées, qui traite les changements d'état de verrou de gamme d'octets et de mode d'ouverture

ainsi que des ACL, pourrait ne pas être co-localisé avec les serveurs de données où les accès d'entrée/sortie sont validés, la mise en œuvre de serveur DOIT prendre soin de propager les changements de cet état aux serveurs de données. Une fois que la propagation aux serveurs de données est achevée, le plein effet de ces changements DOIT être en vigueur au serveurs de données. Cependant, certains changements d'état n'ont pas besoin d'être propagés immédiatement, bien que tous les changements DEVRAIENT être propagés rapidement. Ces propagations d'état ont un impact sur la conception du protocole de contrôle, même si celui-ci sort du domaine d'application de la présente spécification. Propagation immédiate se réfère à la propagation synchrone de l'état du serveur de métadonnées au serveur de données ; la propagation doit être achevée avant le retour au client.

13.9.2.1 Propagation d'état de verrou

Si le serveur pNFS prend en charge le verrouillage obligatoire de gamme d'octets, tous les verrous de gamme d'octets obligatoires sur un fichier DOIVENT être rendus effectifs aux serveurs de données avant que la demande qui les établit retourne à l'appelant. L'effet DOIT être le même que si l'état de verrou de gamme d'octets obligatoire était propagé de façon synchrone aux serveurs de données, même si les détails du protocole de contrôle peuvent éviter un transfert réel de l'état dans certaines circonstances.

Par ailleurs, comme l'état indicatif de verrou de gamme d'octets n'est pas utilisé pour vérifier les accès d'entrée/sortie aux serveurs de données, il n'y a pas de raison sémantique pour propager l'état indicatif de verrou de gamme d'octets aux serveurs de données. Comme les mises à jour de verrous indicatifs ne confèrent ni ne suppriment de privilèges, ces changements n'ont pas besoin d'être propagés immédiatement, et peuvent n'avoir pas besoin d'être propagées rapidement. Les mises à jour de verrous indicatifs doivent seulement être propagées quand le serveur de données a besoin de résoudre une question sur un identifiant d'état. En fait, si le verrouillage de gamme d'octets n'est pas obligatoire (c'est-à-dire, est indicatif) il est conseillé aux clients d'éviter d'utiliser des identifiants d'état fondés sur le verrou de gamme d'octets pour l'entrée/sortie. Les identifiants d'état retournés par OPEN sont suffisants et éliminent les frais généraux pour ce type de propagation d'état.

Si un client obtient en retour une erreur NFS4ERR_LOCKED d'un serveur de données, c'est l'indication qu'un verrouillage obligatoire de gamme d'octets est en vigueur. Le client récupère de cela en obtenant un verrou de gamme d'octets qui couvre la gamme affectée et ré envoie l'entrée/sortie avec l'identifiant d'état du verrou de gamme d'octets.

13.9.2.2 Validation de modes ouvert et refus

La validation de modes ouvert et refus DOIT être effectuée à l'égard des modes ouvert et refus détenus par les serveurs de données. Quand l'accès est réduit ou qu'un mode refus est rendu plus restrictif (à cause d'un CLOSE ou OPEN_DOWNGRADE) le serveur de données DOIT empêcher toutes entrées/sorties qui seraient refusées si elles étaient effectuées sur le serveur de métadonnées. Quand l'accès est élargi, le serveur de données DOIT s'assurer qu'aucune demande n'est ensuite rejetée à cause de questions d'ouverture ou refus qui ne s'appliquent plus, étant donnée l'assouplissement précédent.

13.9.2.3 Attributs de fichier

Comme l'opération SETATTR a la capacité de modifier l'état qui est visible sur les serveurs de métadonnées et de données (par exemple, la taille) il faut veiller à s'assurer que l'état résultant à travers l'ensemble de serveurs de données est cohérent, en particulier quand on tronque ou augmente le fichier.

Comme décrit précédemment, l'opération LAYOUTCOMMIT est utilisée pour s'assurer que les métadonnées sont synchronisées avec les changements faits aux serveurs de données. Pour le protocole de mémorisation de données fondé sur NFSv4.1, il est nécessaire de re-synchroniser l'état comme l'attribut de taille, et le réglage de mtime/change/atime. Voir au paragraphe 12.5.4 la description complète de la sémantque de LAYOUTCOMMIT et de la synchronisation d'attribut. On devrait noter qu'en utilisant le type de disposition fondé sur NFSv4.1, il est possible de synchroniser cet état avant que LAYOUTCOMMIT se produise. Par exemple, le protocole de contrôle peut être utilisé pour interroger les attributs présents sur les serveurs de données.

Tout changement des attributs de fichier qui contrôlent l'autorisation ou l'accès comme reflétés par les appels ACCESS ou les READ et WRITE sur le serveur de métadonnées, DOIT être propagé aux serveurs de données pour application sur les appels d'entrée/sortie READ et WRITE. Si les changements faits sur le serveur de métadonnées résultent en des permissions d'accès plus restrictives pour un utilisateur, ces changements DOIVENT être propagés de façon synchrone aux serveurs de données.

L'opération OPEN (paragraphe 18.16.4) n'impose aucune exigence que les opérations d'entrée/sortie sur un fichier ouvert

aient les mêmes accréditifs que le OPEN lui-même (sauf si le fanion EXCHGID4_FLAG_BIND_PRINC_STATEID est établi quand EXCHANGE_ID crée l'identifiant de client) et donc il exige que les opérations READ et WRITE du serveur effectuent les vérifications d'accès appropriées. Les changements aux ACL exigent aussi une nouvelle vérification d'accès par READ et WRITE sur le serveur. La propagation des droits d'accès change parce que les changements dans les ACL ne peuvent être asynchrones que si la mise en œuvre de serveur est capable de déterminer que l'ACL mise à jour n'est pas plus restrictive pour un utilisateur que spécifié dans la vieille ACL. Du fait de la faible fréquence relative des mises à jour d'ACL, il est suggéré que tous les changements soient propagés de façon synchrone.

13.10 Taille de fichier composant au serveur de données

Un problème potentiel existe quand un un fichier de données composant sur un serveur de données particulier a crû au delà de l'EOF ; le problème existe pour les deux dispositions, dense et éparse. Imaginons le scénario suivant : un client crée un nouveau fichier (taille == 0) et écrit à l'octet 131072 ; le client cherche alors le début du fichier et lit l'octet 100. Le client devrait recevoir des zéros en retour comme résultat du READ. Cependant, si le schéma de mise en bandes conduit le client à envoyer le READ à un serveur de données autre que celui qui a reçu le WRITE original du client, le serveur de données qui sert le READ peut croire que la taille du fichier est encore de 0 octet. Dans ce cas, la réponse READ du serveur de données va contenir zéro octet et une indication de EOF. Le serveur de données peut seulement retourner des zéros si il sait que la taille du fichier a été étendue. Cela exigerait la propagation immédiate de la taille du fichier à tous les serveurs de données, ce qui est potentiellement très coûteux. Donc, le client qui a initié l'extension de la taille du fichier DOIT être prêt à traiter ces conditions de EOF. Quand le décalage dans les arguments à READ est moins que la vue du client de la taille du fichier, si la réponse READ indique EOF et/ou contient moins d'octets que demandé, le client va interpréter une telle réponse comme un trou dans le fichier, et le client NFS va substituer des zéros aux données.

Le protocole NFSv4.1 fournit seulement une sémantique d'antémémoire de données de clôture à ouverture de fichier ; cela signifie que quand le fichier est clos, toutes les données modifiées sont écrites au serveur. Quand un OPEN suivant du fichier est fait, l'attribut de changement est inspecté à la recherche d'une différence avec une valeur mise en antémémoire pour l'attribut de changement. Pour le cas ci-dessus, cela signifie que un LAYOUTCOMMIT va être fait à la clôture (avec les données écrites) et va mettre à jour la taille du fichier et l'attribut de changement. L'accès à partir d'un autre client après ce moment va résulter en le retour de la taille appropriée.

13.11 Révocation et barrage de disposition

Comme décrit au paragraphe 12.7, le protocole de mémorisation spécifique du type de disposition est responsable du traitement des effets des entrées/sorties qui ont commencé avant l'expiration du prêt et étendus à travers l'expiration du prêt. Le type de disposition LAYOUT4_NFSV4_1_FILES peut empêcher l'exécution de toutes les entrées/sorties aux serveurs de données après l'expiration de prêt (cette prévention est appelée un "barrage") sans s'appuyer sur une temporisation précise du prêt au client et sans exiger des serveurs de données qu'ils tiennent des temporisateurs de prêt. Le serveur pNFS LAYOUT4_NFSV4_1_FILES a la souplesse de révoquer les dispositions individuelles, et donc de faire barrage aux entrées/sorties sur la base du fichier.

En plus de l'expiration de prêt, les raisons de révocation d'une disposition incluent : que le client échoue à répondre à un CB_LAYOUTRECALL, que le serveur de métadonnées redémarre, ou une intervention administrative. Sans considération de la raison, une fois que la disposition d'un client a été révoquée, le serveur pNFS DOIT empêcher le client d'envoyer des entrées/sorties pour le fichier affecté de et vers tous les serveurs de données ; en d'autres termes, il DOIT faire un barrage entre le client et le fichier affecté sur les serveurs de données.

Le barrage fonctionne comme suit. Comme décrit au paragraphe 13.1, dans les demandes de procédure COMPOUND au serveur de données, la bride de fichier de données fournie par l'opération PUTFH et l'identifiant d'état dans l'opération READ ou WRITE sont utilisés pour assurer que le client a une disposition valide pour les entrées/sorties effectuées ; si il n'en a pas, l'entrée/sortie est rejetée avec NFS4ERR_PNFS_NO_LAYOUT. Le serveur peut simplement vérifier l'identifiant d'état et, de plus, rendre périmée la bride de fichier de données si la disposition spécifiait une bride de fichier de données différente de la bride de fichier du serveur de métadonnées pour le fichier (voir la description de nfl_fh_list au paragraphe 13.3).

Avant que le serveur de métadonnées effectue aucune action pour révoquer l'état de disposition donné par une précédente instance, il doit s'assurer que tout état de disposition provenant de cette précédente instance est invalidé aux serveurs de données. Cela a les implications suivantes :

* Le serveur de métadonnées ne doit pas remettre en bandes un fichier avant qu'il ait contacté tous les serveurs de données pour invalider les dispositions de la précédente instance.

* Le serveur de métadonnées ne doit pas éliminer les verrous obligatoires en conflit avec les dispositions provenant de l'instance précédente sans avoir fait une invalidation de disposition spécifique (qu'il aurait faite de toutes façons) ou une invalidation globale de serveur de données.

13.12 Considérations de sécurité pour le type de disposition de fichier

Le type de disposition de fichier NFSv4.1 DOIT respecter les considérations de sécurité mentionnées au paragraphe 12.9. Les serveurs de données NFSv4.1 DOIVENT faire toutes les vérifications d'accès requises sur chaque entrée/sortie READ ou WRITE comme déterminé par le protocole NFSv4.1. Si le serveur de métadonnées refuse une opération READ ou WRITE sur un fichier à cause de son ACL, son attribut de mode, son mode d'accès d'ouverture, son mode de refus, son état de verrou de gamme d'octets obligatoire, ou tout autre attribut et état, le serveur de données DOIT aussi refuser l'opération READ ou WRITE. Cela impacte le protocole de contrôle et la propagation de l'état du serveur de métadonnées aux serveurs de données ; voir les détails au paragraphe 13.9.2.

Les méthodes pour l'authentification, l'intégrité, et la confidentialité pour les serveurs de données sur la base du type de disposition LAYOUT4_NFSV4_1_FILES sont les mêmes que celles utilisées par les serveurs de métadonnées. Les serveurs de métadonnées et de données utilisent les nuances de sécurité ONC RPC pour authentifier, et SECINFO et SECINFO_NO_NAME pour négocier le mécanisme de sécurité et les services à utiliser. Donc, quand on utilise le type de disposition LAYOUT4_NFSV4_1_FILES, l'impact sur le modèle de sécurité fondé sur RPC dû à pNFS (comme mentionné aux paragraphes 1.8.1 et 1.8.2.2) est nul.

Pour un objet de fichier donné, un serveur de métadonnées PEUT exiger des paramètres de sécurité différents (valeur de secinfo4) de ceux du serveur de données. Pour un objet de fichier donné avec plusieurs serveurs de données, la valeur de secinfo4 DEVRAIT être la même sur tous les serveurs de données. Si les valeurs de secinfo4 entre un serveur de métadonnées et ses serveurs de données diffèrent pour un fichier spécifique, la transposition du principal en l'identifiant d'utilisateur interne du serveur DOIT être la même afin que les vérifications de contrôle d'accès fondées sur l'ACL, le mode, le mode d'ouverture et de refus, et le verrouillage obligatoire soient cohérentes sur le serveur pNFS.

Si une mise en œuvre NFSv4.1 prend en charge pNFS et prend en charge les dispositions de fichier NFSv4.1, alors cette mise en œuvre DOIT prendre en charge l'opération SECINFO_NO_NAME sur les deux serveurs de métadonnées et de données.

14. Internationalisation

La principale question dans laquelle NFSv4.1 est en rapport avec l'internationalisation, ou I18N, est à l'égard des noms de fichiers et autres chaînes utilisées dans le protocole. Le choix d'une représentation de chaîne doit permettre un accès raisonnable aux noms/chaînes par les clients qui utilisent divers langages. Le codage UTF-8 du jeu de caractères universel codé sur plusieurs octets (UCS, *Universal Multiple-Octet Coded Character Set*) comme défini par la norme [ISO10646-1] permet ce type d'accès et suit la politique décrite dans la [RFC2277] "Politique de l'IETF sur les jeux de caractères et les langages".

La [RFC3454], autrement connue sous le nom de "stringprep", documente un cadre pour utiliser Unicode/UTF-8 dans les protocoles de réseautage afin "d'augmenter la probabilité que l'entrée d'une chaîne et la comparaison de chaînes fonctionne d'une façon qui ait un sens pour les utilisateurs normaux à travers le monde". Un protocole doit définir un profil de stringprep "afin de spécifier pleinement les options de traitement". La suite de cette section définit les profils NFSv4.1 stringprep. Beaucoup de la terminologie utilisée dans le reste de cette section vient de stringprep.

Il y a trois types de chaîne UTF-8 définis pour NFSv4.1 : utf8str_cs, utf8str_cis, et utf8str_mixed. Des profils séparés sont définis pour chacun. Chaque profil définit ce qui suit, conformément à stringprep :

* L'applicabilité prévue du profil.
* Le répertoire de caractères en entrée et sortie de stringprep (Unicode 3.2 pour la version référencée de stringprep). Cependant, les mises en œuvre de NFSv4.1 ne sont pas limitées à 3.2.
* Les tableaux de transposition de stringprep utilisés (comme décrit à la Section 3 de stringprep).
* Tous tableaux de transposition supplémentaires spécifiques du profil.
* La normalisation Unicode utilisée, si il en est (comme décrit à la Section 4 de stringprep).
* Les tableaux de stringprep des caractères interdits en sortie (comme décrit à la Section 5 de stringprep).
* L'essai bidirectionnel de chaîne utilisé, si il en est (comme décrit à la Section 6 de stringprep).
* Tous caractères supplémentaires interdits en sortie spécifiques du profil.

Stringprep discute des caractères Unicode, tandis que NFSv4.1 produit des caractères UTF-8. Comme il y a une transposition biunivoque entre UTF-8 et Unicode, quand on se réfère à Unicode dans la suite de ce document, le lecteur devrait supposer UTF-8.

Beaucoup du texte des profils vient de la [RFC3491].

14.1 Profil Stringprep pour le type utf8str_cs

Chaque utilisation de la définition de type utf8str_cs dans la spécification du protocole NFSv4 suit le profil appelé nfs4_cs_prep.

14.1.1 Applicabilité prévue pour le profil nfs4_cs_prep

Le type utf8str_cs est une chaîne de caractères UTF-8 sensible à la casse. Son principal usage dans NFSv4.1 est de nommer les composants et les noms de chemin. Les composants et les noms de chemin sont mémorisés sur le système de fichiers du serveur. Deux chaînes UTF-8 valides distinctes pourraient être les mêmes après traitement via le profil utf8str_cs. Si les chaînes sont deux noms dans un répertoire, le serveur NFSv4.1 va devoir soit :

* interdire la création d'un second nom si sa forme après traitement entre en collision avec celle d'un nom existant,
* permettre la création du second nom, mais s'arranger pour que après le traitement, le second nom soit différent de la forme après traitement du premier nom.

14.1.2 Répertoire de caractères de nfs4_cs_prep

Le profil nfs4_cs_prep utilise Unicode 3.2, comme défini dans l'Appendice A.1 de stringprep. Cependant, les mises en œuvre de NFSv4.1 ne sont pas limitées à 3.2.

14.1.3 Transposition utilisée par nfs4_cs_prep

Le profil nfs4_cs_prep profile spécifie la transposition en utilisant les tableaux suivants provenant de stringprep :

Tableau B.1

Le Tableau B.2 ne fait normalement pas partie du profil nfs4_cs_prep car il est principalement destiné à traiter les comparaisons insensibles à la casse. Cependant, si le serveur de fichier NFSv4.1 prend en charge l'attribut de système de fichiers case_insensitive, et si case_insensitive est VRAI, le serveur NFSv4.1 DOIT utiliser le Tableau B.2 (en plus du Tableau B1) quand il traite des chaînes utf8str_cs, et le client NFSv4.1 DOIT supposer que le Tableau B.2 (en plus du Tableau B.1) est utilisé.

Si l'attribut case_preserving est présent et réglé à FAUX, alors le serveur NFSv4.1 DOIT utiliser le Tableau B.2 pour transposer la casse dans le traitement des chaînes utf8str_cs. Si le serveur transpose de minuscules en majuscules ou l'inverse dépend de la mise en œuvre.

14.1.4 Normalisation utilisée par nfs4_cs_prep

Le profil nfs4_cs_prep ne spécifie pas une forme de normalisation. Une révision ultérieure de la présente spécification pourrait spécifier une forme particulière de normalisation. Donc, le serveur et le client peuvent s'attendre à ce qu'ils puissent recevoir des caractères non normalisés dans des demandes et réponses de protocole. Si l'environnement de fonctionnement exige la normalisation, alors la mise en œuvre doit normaliser les chaînes utf8str_cs au sein du protocole avant de présenter les informations à une application (chez le client) ou au système de fichiers local (chez le serveur).

14.1.5 Résultat interdit pour nfs4_cs_prep

Le profil nfs4_cs_prep RECOMMANDE d'interdire l'utilisation des tableaux suivants de stringprep : Table C.5, Table C.6

14.1.6 Résultat bidirectionnel pour nfs4_cs_prep

Le profil nfs4_cs_prep ne spécifie aucune vérification bidirectionnelle de chaînes.

14.2 Profil stringprep pour le type utf8str_cis

Toute utilisation de la définition de type utf8str_cis dans la spécification de protocole NFSv4.1 suit le profil appelé nfs4_cis_prep.

14.2.1 Applicabilité prévue du profil nfs4_cis_prep

Le type utf8str_cis est une chaîne insensible à la casse de caractères UTF-8. Son usage principal dans NFSv4.1 est pour désigner les serveurs NFS.

14.2.2 Répertoire de caractères de nfs4_cis_prep

Le profil nfs4_cis_prep utilise Unicode 3.2, comme défini à l'Appendice A.1 de stringprep. Cependant, les mises en œuvre de NFSv4.1 ne sont pas limitées à 3.2.

14.2.3 Transposition utilisée par nfs4_cis_prep

Le profil nfs4_cis_prep spécifie une transposition utilisant les tableaux de stringprep suivants : Table B.1, Table B.2

14.2.4 Normalisation utilisée par nfs4_cis_prep

Le profil nfs4_cis_prep spécifie d'utiliser la forme de normalisation KC de Unicode, comme décrit dans stringprep.

14.2.5 Résultat interdit pour nfs4_cis_prep

Le profil nfs4_cis_prep spécifie l'interdiction d'utiliser les tableaux de stringprep suivants :Table C.1.2, Table C.2.2, Table C.3, Table C.4, Table C.5, Table C.6, Table C.7, Table C.8, Table C.9

14.2.6 Résultat bidirectionnel pour nfs4_cis_prep

Le profil nfs4_cis_prep spécifie de vérifier les chaînes bidirectionnelles comme décrit à la Section 6 de stringprep.

14.3 Profil Stringprep pour le type utf8str_mixed

Toute utilisation de la définition de type utf8str_mixed dans la spécification de protocole NFSv4.1 suit le profil appelé nfs4_mixed_prep.

14.3.1 Applicabilité prévue du profil nfs4_mixed_prep

Le type utf8str_mixed est une chaîne de caractères UTF-8, avec un préfixe sensible à la casse, un séparateur égal à "@", et un suffixe qui est un nom de domaine pleinement qualifié. Son utilisation principale dans NFSv4.1 est pour nommer les principaux identifiés dans une entrée de contrôle d'accès.

14.3.2 Répertoire de caractères de nfs4_mixed_prep

Le profil nfs4_mixed_prep utilise Unicode 3.2, comme défini dans l'Appendice A.1 de stringprep. Cependant, les mises en œuvre de NFSv4.1 ne sont pas limitées à 3.2.

14.3.3 Transposition utilisée par nfs4_cis_prep

Pour le préfixe et le séparateur d'une chaîne utf8str_mixed, le profil nfs4_mixed_prep spécifie une transposition utilisant le

tableau suivant de stringprep : Table B.1.

Pour le suffixe d'une chaîne utf8str_mixed, le profil nfs4_mixed_prep spécifie de transposer en utilisant les tableaux suivants de stringprep : Table B.1, Table B.2.

14.3.4 Normalisation utilisée par nfs4_mixed_prep

Le profil nfs4_mixed_prep spécifie d'utiliser la forme KC de normalisation Unicode, comme décrit dans stringprep.

14.3.5 Résultat interdit pour nfs4_mixed_prep

Le profil nfs4_mixed_prep spécifie l'interdiction d'utiliser les tableaux suivants de stringprep : Table C.1.2, Table C.2.2, Table C.3, Table C.4, Table C.5, Table C.6, Table C.7, Table C.8, Table C.9.

14.3.6 Résultat bidirectionnel pour nfs4_mixed_prep

Le profil nfs4_mixed_prep spécifie de vérifier les chaînes bidirectionnelles comme décrit à la Section 6 de stringprep.

14.4 Capacités UTF-8

```
const FSCHARSET_CAP4_CONTAINS_NON_UTF8  = 0x1;
const FSCHARSET_CAP4_ALLOWS_ONLY_UTF8   = 0x2;

typedef uint32_t        fs_charset_cap4;
```

Parce que certains environnements de fonctionnement et systèmes de fichiers n'appliquent pas les codages de jeu de caractères, NFSv4.1 prend en charge l'attribut fs_charset_cap (paragraphe 5.8.2.11) qui indique au client les capacités UTF-8 d'un système de fichiers. L'attribut est un entier contenant une paire de fanions. Le premier fanion est FSCHARSET_CAP4_CONTAINS_NON_UTF8, qui, si il est établi à un, dit au client que le système de fichiers ne contient pas de caractères UTF-8, et le serveur ne va pas convertir les caractères non UTF en UTF-8 si le client lit un lien symbolique ou un répertoire, ni les opérations avec des noms de composant ou des noms de chemin dans les arguments ne vont convertir les chaînes en UTF-8. Le second fanion est FSCHARSET_CAP4_ALLOWS_ONLY_UTF8, qui, si il est établi à un, indique que le serveur va accepter (et générer) seulement des caractères UTF-8 sur le système de fichiers. Si FSCHARSET_CAP4_ALLOWS_ONLY_UTF8 est établi à un, FSCHARSET_CAP4_CONTAINS_NON_UTF8 DOIT être réglé à zéro. FSCHARSET_CAP4_ALLOWS_ONLY_UTF8 DEVRAIT toujours être établi à un.

14.5 Erreurs relatives à UTF-8

Lorsque le client envoie une chaîne UTF-8 invalide, le serveur devrait retourner NFS4ERR_INVAL (voir le Tableau 11). Cela inclut les cas où des préfixes inappropriés sont détectés et où le compte inclut des octets en queue qui ne constituent pas un plein caractère UCS.

Lorsque la chaîne fournie par le client est de l'UTF-8 valide mais contient des caractères qui ne sont pas pris en charge par le serveur comme une valeur pour cette chaîne (par exemple, des noms contenant des caractères en dehors du plan 0 de Unicode sur les systèmes de fichiers qui échouent à prendre en charge de tels caractères en dépit de leur présence dans la norme Unicode) le serveur devrait retourner NFS4ERR_BADCHAR.

Lorsque une chaîne UTF-8 est utilisée comme un nom de fichier, et que le système de fichiers (bien que prenant en charge tous les caractères du nom) ne permet pas que ce nom particulier soit utilisé, le serveur devrait retourner l'erreur NFS4ERR_BADNAME (Tableau 11). Cela inclut des situations où le serveur de système de fichiers impose une contrainte de normalisation aux chaînes de noms, mais va aussi inclure des situations comme des interdictions de système de fichiers de "." et ".." comme noms de fichiers pour certaines opérations, et d'autres contraintes de cette sorte.

15. Valeurs d'erreur

Les numéros d'erreur NFS sont alloués aux opérations défaillantes dans une demande Compound (COMPOUND ou

CB_COMPOUND). Une demande Compound contient un certain nombre d'opérations NFS qui ont leur résultat codé à la suite dans une réponse Compound. Le résultat des opérations réussies va consister en un état NFS4_OK suivi par le résultat codé de l'opération. Si une opération NFS échoue, un état d'erreur va être entré dans la réponse et la demande Compound sera terminée.

15.1 Définitions des erreurs

Erreur	Numéro	Paragraphe de description
NFS4_OK	0	paragraphe 15.1.3.1
NFS4ERR_ACCESS	13	paragraphe 15.1.6.1
NFS4ERR_ATTRNOTSUPP	10032	paragraphe 15.1.15.1
NFS4ERR_ADMIN_REVOKED	10047	paragraphe 15.1.5.1
NFS4ERR_BACK_CHAN_BUSY	10057	paragraphe 15.1.12.1
NFS4ERR_BADCHAR	10040	paragraphe 15.1.7.1
NFS4ERR_BADHANDLE	10001	paragraphe 15.1.2.1
NFS4ERR_BADIOMODE	10049	paragraphe 15.1.10.1
NFS4ERR_BADLAYOUT	10050	paragraphe 15.1.10.2
NFS4ERR_BADNAME	10041	paragraphe 15.1.7.2
NFS4ERR_BADOWNER	10039	paragraphe 15.1.15.2
NFS4ERR_BADSESSION	10052	paragraphe 15.1.11.1
NFS4ERR_BADSLOT	10053	paragraphe 15.1.11.2
NFS4ERR_BADTYPE	10007	paragraphe 15.1.4.1
NFS4ERR_BADXDR	10036	paragraphe 15.1.1.1
NFS4ERR_BAD_COOKIE	10003	paragraphe 15.1.1.2
NFS4ERR_BAD_HIGH_SLOT	10077	paragraphe 15.1.11.3
NFS4ERR_BAD_RANGE	10042	paragraphe 15.1.8.1
NFS4ERR_BAD_SEQID	10026	paragraphe 15.1.16.1
NFS4ERR_BAD_SESSION_DIGEST	10051	paragraphe 15.1.12.2
NFS4ERR_BAD_STATEID	10025	paragraphe 15.1.5.2
NFS4ERR_CB_PATH_DOWN	10048	paragraphe 15.1.11.4
NFS4ERR_CLID_INUSE	10017	paragraphe 15.1.13.2
NFS4ERR_CLIENTID_BUSY	10074	paragraphe 15.1.13.1
NFS4ERR_COMPLETE_ALREADY	10054	paragraphe 15.1.9.1
NFS4ERR_CONN_NOT_BOUND_TO_SESSION	10055	paragraphe 15.1.11.6
NFS4ERR_DEADLOCK	10045	paragraphe 15.1.8.2
NFS4ERR_DEADSESSION	10078	paragraphe 15.1.11.5
NFS4ERR_DELAY	10008	paragraphe 15.1.1.3
NFS4ERR_DELEG_ALREADY_WANTED	10056	paragraphe 15.1.14.1
NFS4ERR_DELEG_REVOKED	10087	paragraphe 15.1.5.3
NFS4ERR_DENIED	10010	paragraphe 15.1.8.3
NFS4ERR_DIRDELEG_UNAVAIL	10084	paragraphe 15.1.14.2
NFS4ERR_DQUOT	69	paragraphe 15.1.4.2
NFS4ERR_ENCR_ALG_UNSUPP	10079	paragraphe 15.1.13.3
NFS4ERR_EXIST	17	paragraphe 15.1.4.3
NFS4ERR_EXPIRED	10011	paragraphe 15.1.5.4
NFS4ERR_FBIG	27	paragraphe 15.1.4.4
NFS4ERR_FHEXPIRED	10014	paragraphe 15.1.2.2
NFS4ERR_FILE_OPEN	10046	paragraphe 15.1.4.5
NFS4ERR_GRACE	10013	paragraphe 15.1.9.2
NFS4ERR_HASH_ALG_UNSUPP	10072	paragraphe 15.1.13.4
NFS4ERR_INVAL	22	paragraphe 15.1.1.4
NFS4ERR_IO	5	paragraphe 15.1.4.6
NFS4ERR_ISDIR	21	paragraphe 15.1.2.3
NFS4ERR_LAYOUTTRYLATER	10058	paragraphe 15.1.10.3
NFS4ERR_LAYOUTUNAVAILABLE	10059	paragraphe 15.1.10.4
NFS4ERR_LEASE_MOVED	10031	paragraphe 15.1.16.2
NFS4ERR_LOCKED	10012	paragraphe 15.1.8.4
NFS4ERR_LOCKS_HELD	10037	paragraphe 15.1.8.5
NFS4ERR_LOCK_NOTSUPP	10043	paragraphe 15.1.8.6
NFS4ERR_LOCK_RANGE	10028	paragraphe 15.1.8.7

NFS4ERR_MINOR_VERS_MISMATCH	10021	paragraphe 15.1.3.2
NFS4ERR_MLINK	31	paragraphe 15.1.4.7
NFS4ERR_MOVED	10019	paragraphe 15.1.2.4
NFS4ERR_NAMETOOLONG	63	paragraphe 15.1.7.3
NFS4ERR_NOENT	2	paragraphe 15.1.4.8
NFS4ERR_NOFILEHANDLE	10020	paragraphe 15.1.2.5
NFS4ERR_NOMATCHING_LAYOUT	10060	paragraphe 15.1.10.5
NFS4ERR_NOSPC	28	paragraphe 15.1.4.9
NFS4ERR_NOTDIR	20	paragraphe 15.1.2.6
NFS4ERR_NOTEMPTY	66	paragraphe 15.1.4.10
NFS4ERR_NOTSUPP	10004	paragraphe 15.1.1.5
NFS4ERR_NOT_ONLY_OP	10081	paragraphe 15.1.3.3
NFS4ERR_NOT_SAME	10027	paragraphe 15.1.15.3
NFS4ERR_NO_GRACE	10033	paragraphe 15.1.9.3
NFS4ERR_NXIO	6	paragraphe 15.1.16.3
NFS4ERR_OLD_STATEID	10024	paragraphe 15.1.5.5
NFS4ERR_OPENMODE	10038	paragraphe 15.1.8.8
NFS4ERR_OP_ILLEGAL	10044	paragraphe 15.1.3.4
NFS4ERR_OP_NOT_IN_SESSION	10071	paragraphe 15.1.3.5
NFS4ERR_PERM	1	paragraphe 15.1.6.2
NFS4ERR_PNFS_IO_HOLE	10075	paragraphe 15.1.10.6
NFS4ERR_PNFS_NO_LAYOUT	10080	paragraphe 15.1.10.7
NFS4ERR_RECALLCONFLICT	10061	paragraphe 15.1.14.3
NFS4ERR_RECLAIM_BAD	10034	paragraphe 15.1.9.4
NFS4ERR_RECLAIM_CONFLICT	10035	paragraphe 15.1.9.5
NFS4ERR_REJECT_DELEG	10085	paragraphe 15.1.14.4
NFS4ERR_REP_TOO_BIG	10066	paragraphe 15.1.3.6
NFS4ERR_REP_TOO_BIG_TO_CACHE	10067	paragraphe 15.1.3.7
NFS4ERR_REQ_TOO_BIG	10065	paragraphe 15.1.3.8
NFS4ERR_RESTOREFH	10030	paragraphe 15.1.16.4
NFS4ERR_RETRY_UNCACHED_REP	10068	paragraphe 15.1.3.9
NFS4ERR_RETURNCONFLICT	10086	paragraphe 15.1.10.8
NFS4ERR_ROFS	30	paragraphe 15.1.4.11
NFS4ERR_SAME	10009	paragraphe 15.1.15.4
NFS4ERR_SHARE_DENIED	10015	paragraphe 15.1.8.9
NFS4ERR_SEQUENCE_POS	10064	paragraphe 15.1.3.10
NFS4ERR_SEQ_FAUX_RETRY	10076	paragraphe 15.1.11.7
NFS4ERR_SEQ_MISORDERED	10063	paragraphe 15.1.11.8
NFS4ERR_SERVERFAULT	10006	paragraphe 15.1.1.6
NFS4ERR_STALE	70	paragraphe 15.1.2.7
NFS4ERR_STALE_CLIENTID	10022	paragraphe 15.1.13.5
NFS4ERR_STALE_STATEID	10023	paragraphe 15.1.16.5
NFS4ERR_SYMLINK	10029	paragraphe 15.1.2.8
NFS4ERR_TOOSMALL	10005	paragraphe 15.1.1.7
NFS4ERR_TOO_MANY_OPS	10070	paragraphe 15.1.3.11
NFS4ERR_UNKNOWN_LAYOUTTYPE	10062	paragraphe 15.1.10.9
NFS4ERR_UNSAFE_COMPOUND	10069	paragraphe 15.1.3.12
NFS4ERR_WRONGSEC	10016	paragraphe 15.1.6.3
NFS4ERR_WRONG_CRED	10082	paragraphe 15.1.6.4
NFS4ERR_WRONG_TYPE	10083	paragraphe 15.1.2.9
NFS4ERR_XDEV	18	paragraphe 15.1.4.12

Tableau 11 : Définitions des erreurs de protocole

15.1.1 Erreurs générales

Ce paragraphe traite des erreurs applicables à un large ensemble d'objets différents.

15.1.1.1 NFS4ERR_BADXDR (code d'erreur 10036)

Les arguments pour cette opération ne correspondent pas à ceux spécifiés dans la définition XDR. Cela inclut des situations où la demande se termine avant que tous les arguments aient été vus. Noter que cette erreur s'applique quand des énumérations fixes (cela inclut des booléens) ont une valeur dans le flux d'entrée qui n'est pas valide pour l'énumération. Un répondeur peut pré-analyser toutes les opérations pour une procédure Compound avant de faire l'exécution d'une opération et retourner des erreur XDR de niveau RPC dans ce cas.

15.1.1.2 NFS4ERR_BAD_COOKIE (code d'erreur 10003)

Utilisé pour des opérations qui fournissent un ensemble d'informations indexées par une certaine quantité fournie par le client ou un mouchard *(cookie)* envoyé par le serveur pour une invocation antérieure. Lorsque la valeur ne peut pas être utilisée pour l'objet prévu, il en résulte cette erreur.

15.1.1.3 NFS4ERR_DELAY (code d'erreur 10008)

Pour toute une série de raisons, le répondeur n'a pas pu traiter cette opération dans ce qui était estimé un délai raisonnable. Le client devrait attendre avant de réessayer la demande avec un nouveau créneau et une nouvelle valeur de séquence.

Exemples de scénarios qui pourrait conduire à cette situation :

* Un serveur qui prend en charge la mémorisation hiérarchique reçoit une demande pour traiter un fichier déjà migré.
* Une opération exige un rappel de délégation pour continuer, mais le besoin d'attendre le rappel de cette délégation et son retour rendent le traitement de cette demande impossible en temps utile.
* Une demande est effectuée sur une session qui migre d'un autre serveur comme décrit au paragraphe 11.14.3, et l'absence d'informations complètes sur l'état de la session à la source rend impossible le traitement immédiat de la demande.

Dans de tels cas, retourner l'erreur NFS4ERR_DELAY permet les nécessaires opérations préparatoires pour poursuivre sans retenir des ressources du demandeur comme un créneau de session. Après un certain délai, le client peut alors envoyer à nouveau l'opération en question, souvent au titre d'une demande presque identique. À cause du besoin d'éviter des reproductions parasites d'opérations non idempotentes et éviter d'agir en réponse aux erreurs NFS4ERR_DELAY retournées sur des réponses revenant de la mémoire tampon de réponse du répondeur, l'intégration avec la mémoire tampon de réponse fournie par la session est nécessaire. Il y a un certain nombre de cas à traiter, dont chacun exige un traitement différent de la part du demandeur et du répondeur :

* Si NFS4ERR_DELAY est retourné sur une opération SEQUENCE, la demande est réessayée en entier avec l'opération SEQUENCE contenant les mêmes valeurs de créneau et séquence. Dans ce cas, le répondeur DOIT éviter de retourner une réponse contenant NFS4ERR_DELAY comme la réponse à SEQUENCE seulement parce que une instance antérieure de la même demande a retourné cette erreur et qu'elle a été mémorisée dans la mémoire tampon de réponse. Si le répondeur a fait cela, le ré-essai ne sera pas efficace car il n'y aura pas d'opportunité pour le répondeur de voir si la condition qui a généré le NFS4ERR_DELAY a été rectifiée entre la demande d'origine et le ré-essai.

* Si NFS4ERR_DELAY est retourné sur une opération autre que SEQUENCE qui apparaît validement comme la première opération d'une demande, le traitement est similaire. La demande peut être ré-essayée en entier sans modification. Dans ce cas aussi, le répondeur DOIT éviter de retourner une réponse contenant NFS4ERR_DELAY comme la réponse à une opération initiale d'une demande seulement sur la base de sa présence dans la mémoire tampon de réponse. Si le répondeur fait cela, le ré-essai ne sera pas efficace car il n'y aura pas d'opportunité pour le répondeur de voir si la condition qui a généré le NFS4ERR_DELAY a été rectifiée entre la demande d'origine et le ré-essai.

* Si NFS4ERR_DELAY est retourné sur une opération autre que la première dans la demande, la demande ré-essayée DOIT contenir une opération SEQUENCE différente de l'originale, avec l'identifiant de créneau ou la valeur de séquence différent de celui de la demande d'origine. Parce que les demandeurs font cela, il n'est pas besoin que le répondeur prenne un soin particulier pour éviter de retourner une erreur NFS4ERR_DELAY obtenue de la mémoire tampon de réponse. Quand aucune opération non idempotente n'a été traitée avant le retour du NFS4ERR_DELAY, le demandeur devrait ré-essayer la demande entièrement, avec pour seule différence avec la demande d'origine la modification de l'identifiant de créneau ou de la valeur de séquence dans la nouvelle opération SEQUENCE.

* Quand NFS4ERR_DELAY est retourné sur une opération autre que la première dans une demande et qu'il y a eu des opérations non idempotentes traitées avant le retour de NFS4ERR_DELAY, reproduire la demande comme elle est normalement faite causerait une ré-exécution incorrecte des opérations non idempotentes.

Pour éviter cette situation, le client devrait reproduire la demande sans les opérations non idempotentes. La demande doit

quand même utiliser une opération SEQUENCE avec un identifiant de créneau ou valeur de séquence différent du SEQUENCE dans la demande d'origine. Parce que ceci est fait, il n'y a pas de moyen par lequel le répondeur pourrait éviter de ré-exécuter de façon parasitaire les opérations non idempotentes car les différents paramètres de SEQUENCE empêchent le demandeur de reconnaître que les opérations non idempotentes sont ré-essayées.

Noter que sans la capacité de retourner NFS4ERR_DELAY et la volonté du demandeur de renvoyer quand il le reçoit, une impasse pourrait résulter. Par exemple, si un rappel est fait, et si le retour de délégation ou si des opérations préparatoires au retour de la délégation sont détenus par d'autres opérations qui ont besoin du retour de la délégation, les créneaux de session pourrait n'être pas disponibles. Le résultat pourrait être une impasse.

15.1.1.4 NFS4ERR_INVAL (code d'erreur 22)

Les arguments pour cette opération ne sont pas valides pour une raison quelconque, même si ils correspondent à ceux spécifiés dans la définition XDR pour la demande.

15.1.1.5 NFS4ERR_NOTSUPP (code d'erreur 10004)

Opération non prise en charge, soit parce que l'opération est FACULTATIVE et n'est pas prise en charge par ce serveur, soit parce que l'opération NE DOIT PAS être mise en œuvre dans la version mineure courante.

15.1.1.6 NFS4ERR_SERVERFAULT (code d'erreur 10006)

Une erreur s'est produite chez le serveur qui ne se transpose pas en une valeur d'erreur légale spécifique du protocole NFSv4.1. Le client devrait traduire cela en une erreur appropriée. Les clients UNIX peuvent choisir de traduire cela en EIO.

15.1.1.7 NFS4ERR_TOOSMALL (code d'erreur 10005)

Utilisé lorsque une opération retourne une quantité variable de données, avec une limite spécifiée par le client. Lorsque les données retournées ne peuvent pas tenir dans la limite spécifiée par le client, il en résulte cette erreur.

15.1.2 Erreurs de bride de fichier

Ces erreurs traitent de la situation où la bride de fichier courante ou sauvegardée, ou la bride de fichier passée à PUTFH destinée à devenir la bride de fichier courante, est invalide d'une certaine façon. Cela inclut des situations dans lesquelles la bride de fichier est valide en général mais n'est pas du type d'objet approprié pour l'opération en cours.

Lorsque la description de l'erreur indique un problème avec la bride de fichier courante ou sauvegardée, il doit être compris que les brides de fichier sont seulement vérifiées quant à la condition qu'elles sont des arguments implicites de l'opération en question.

15.1.2.1 NFS4ERR_BADHANDLE (code d'erreur 10001)

Bride de fichier NFS illégale pour le serveur courant. La bride de fichier courante a échoué aux vérifications de cohérence internes. Une fois acceptée comme valide (par PUTFH) aucun changement d'état suivant ne peut causer la génération de cette erreur par la bride de fichier.

15.1.2.2 NFS4ERR_FHEXPIRED (code d'erreur 10014)

Une bride de fichier courante ou sauvegardée qui est un argument de l'opération courante est volatile et a expiré chez le serveur.

15.1.2.3 NFS4ERR_ISDIR (code d'erreur 21)

La bride de fichier courante ou sauvegardée désigne un répertoire quand l'opération courante ne permet pas qu'un répertoire soit accepté comme cible de cette opération.

15.1.2.4 NFS4ERR_MOVED (code d'erreur 10019)

Le système de fichiers qui contient la bride courante d'objet de fichier n'est pas présent chez le serveur ou n'est pas accessible avec l'adresse réseau utilisée. Il peut avoir été rendu accessible sur un ensemble différent d'adresses réseau, relocalisé ou migré sur un autre serveur, ou il peut n'avoir jamais été présent. Le client peut obtenir la nouvelle localisation

du système de fichiers en obtenant l'attribut fs_locations ou fs_locations_info pour la bride de fichier courante. Pour une discussion plus détaillée, voir au paragraphe 11.3.

Comme avec le cas de NFS4ERR_DELAY, il est possible que une ou plusieurs opérations non idempotentes aient été exécutée avec succès dans un COMPOUND avant que NFS4ERR_MOVED soit retourné. À cause de cela, une fois la nouvelle localisation déterminée, la demande d'origine qui a reçu le NFS4ERR_MOVED ne devrait pas être ré-exécutée en entier. À la place, le client devrait envoyer un nouveau COMPOUND en y retirant toute opération non idempotente exécutée avec succès. Quand le client utilise la même session pour le nouveau COMPOUND, son opération SEQUENCE devrait utiliser un identifiant de créneau ou séquence différent.

15.1.2.5 NFS4ERR_NOFILEHANDLE (code d'erreur 10020)

La valeur de bride de fichier logique courante ou sauvegardée est exigée par l'opération courante et n'est pas établie. Cela peut être le résultat d'une opération COMPOUND mal formée (c'est-à-dire, non PUTFH ou PUTROOTFH avant une opération qui exige que la bride de fichier courante soit établie).

15.1.2.6 NFS4ERR_NOTDIR (code d'erreur 20)

La bride de fichier courante (ou sauvegardée) désigne un objet qui n'est pas un répertoire pour une opération dans laquelle un répertoire est exigé.

15.1.2.7 NFS4ERR_STALE (code d'erreur 70)

La valeur de bride de fichier courante ou sauvegardée désignant un argument à l'opération courante est invalide. Le fichier référencé par cette bride de fichier n'existe plus ou son accès a été révoqué.

15.1.2.8 NFS4ERR_SYMLINK (code d'erreur 10029)

La bride de fichier courante désigne une liaison symbolique alors que l'opération courante ne permet pas de liaison symbolique comme cible.

15.1.2.9 NFS4ERR_WRONG_TYPE (code d'erreur 10083)

La bride de fichier courante (ou sauvegardée) désigne un objet qui est de type invalide pour l'opération courante, et il n'y a pas d'erreur plus spécifique (comme NFS4ERR_ISDIR ou NFS4ERR_SYMLINK) qui s'applique. Noter que dans NFSv4.0, de telles situations résultaient généralement en l'erreur moins spécifique NFS4ERR_INVAL.

15.1.3 Erreurs de structure Compound

Cette section traite des erreurs relatives à la structure globale d'une demande Compound (dans laquelle on inclut les deux COMPOUND et CB_COMPOUND) plutôt que à des opérations particulières.

Il y a un certain nombre de contraintes de base sur les opérations qui peuvent apparaître dans une demande Compound. Les sessions ajoutent à ces contraintes de base en exigeant une opération SEQUENCE (SEQUENCE ou CB_SEQUENCE) au début du Compound.

15.1.3.1 NFS_OK (code d'erreur 0)

Indique que l'opération s'est achevée avec succès, en ce que toutes les opérations constituantes se sont achevées sans erreur.

15.1.3.2 NFS4ERR_MINOR_VERS_MISMATCH (code d'erreur 10021)

La version mineure spécifiée n'est pas une de celles que l'écoutant actuel prend en charge. Cette valeur est retournée dans l'état global pour le Compound mais n'est pas associée à une opération spécifique car le résultat va spécifier un compte de résultat de zéro.

15.1.3.3 NFS4ERR_NOT_ONLY_OP (code d'erreur 10081)

Certaines opérations, dont l'exécution est permise en dehors d'une session, DOIVENT être la seule opération dans un Compound chaque fois que le Compound ne commence pas par une opération SEQUENCE. Cette erreur résulte quand cette contrainte n'est pas respectée.

15.1.3.4 NFS4ERR_OP_ILLEGAL (code d'erreur 10044)

Le code d'opération n'est pas valide pour la procédure Compound courante. Le opcode dans le flux de résultat confronté à cette erreur est la valeur ILLEGAL, bien que la valeur qui apparaît dans le flux de demande puisse être différente. Lorsque une valeur illégale apparaît et que le répondeur pré-analyse toutes les opérations pour une procédure Compound avant d'exécuter aucune opération, une erreur XDR de niveau RPC peut être retournée.

15.1.3.5 NFS4ERR_OP_NOT_IN_SESSION (code d'erreur 10071)

La plupart des opérations vers l'avant et toutes les opérations de rappel sont seulement valides dans le contexte d'une session, de sorte que la demande Compound en question DOIT commencer par une opération SEQUENCE. Si une tentative est faite d'exécuter ces opérations en dehors du contexte de session, il en résulte cette erreur.

15.1.3.6 NFS4ERR_REP_TOO_BIG (code d'erreur 10066)

La réponse à une demande Compound excéderait la taille de réponse maximum négociée du canal.

15.1.3.7 NFS4ERR_REP_TOO_BIG_TO_CACHE (code d'erreur 10067)

La réponse à un Compound excéderait la taille maximum négociée du canal pour les réponses mises dans la mémoire tampon de réponse quand la séquence pour la demande en cours spécifie que cette demande est à mettre en antémémoire.

15.1.3.8 NFS4ERR_REQ_TOO_BIG (code d'erreur 10065)

La demande Compound excède la taille maximum négociée du canal pour les demandes.

15.1.3.9 NFS4ERR_RETRY_UNCACHED_REP (code d'erreur 10068)

Le demandeur a tenté de ré-essayer un Compound qu'il avait été précédemment demandé de n'être pas placé dans la mémoire tampon de réponse.

15.1.3.10 NFS4ERR_SEQUENCE_POS (code d'erreur 10064)

Une opération SEQUENCE est apparue dans une position autre que la première opération d'une demande Compound.

15.1.3.11 NFS4ERR_TOO_MANY_OPS (code d'erreur 10070)

La demande Compound a trop d'opérations, excédant le compte négocié quand la session a été créée.

15.1.3.12 NFS4ERR_UNSAFE_COMPOUND (code d'erreur 10068)

Le client a envoyé une demande COMPOUND avec un mélange non sûr d'opérations -- spécifiquement, avec des opérations non idempotentes qui changent la bride de fichier courante et ne sont pas suivies d'un GETFH.

15.1.4 Erreurs de système de fichier

Ces erreurs décrivent des situations qui se sont produites dans la mise en œuvre sous-jacente de système de fichiers plutôt que dans le protocole ou une caractéristique de NFSv4.x.

15.1.4.1 NFS4ERR_BADTYPE (code d'erreur 10007)

Une tentative a été faite de créer un objet avec un type inapproprié spécifié à CREATE. Cela peut être parce que le type est indéfini, parce que le type n'est pas pris en charge par le serveur, ou parce que le type n'est pas destiné à être créé par CREATE (comme un fichier régulier ou attribut nommé, pour lequel OPEN est utilisé pour faire la création de fichier).

15.1.4.2 NFS4ERR_DQUOT (code d'erreur 69)

La limite physique de ressource (quota) est dépassée. La limite de ressource de l'utilisateur sur le serveur a été excédée.

15.1.4.3 NFS4ERR_EXIST (code d'erreur 17)

Un fichier du nom cible spécifié (lors de la création, d'un renommage, ou d'une liaison) existe déjà.

15.1.4.4 NFS4ERR_FBIG (code d'erreur 27)

Le fichier est trop grand. L'opération aurait causé la croissance du fichier au delà de la limite du serveur.

15.1.4.5 NFS4ERR_FILE_OPEN (code d'erreur 10046)

L'opération n'est pas permise parce que un fichier impliqué dans l'opération est actuellement ouvert. Les serveurs peuvent, mais ne sont pas obligés, interdire, relier, renommer, ou supprimer des fichiers ouverts.

15.1.4.6 NFS4ERR_IO (code d'erreur 5)

Indique qu'une erreur d'entrée/sortie s'est produite pour laquelle le système de fichiers a été incapable de fournir la récupération.

15.1.4.7 NFS4ERR_MLINK (code d'erreur 31)

La demande aurait causé le dépassement de la limite du serveur sur le nombre de liaisons physiques qu'un fichier peut avoir.

15.1.4.8 NFS4ERR_NOENT (code d'erreur 2)

Indique qu'il n'y a pas de tel fichier ou répertoire. Le nom de fichier ou de répertoire spécifié n'existe pas.

15.1.4.9 NFS4ERR_NOSPC (code d'erreur 28)

Indique qu'il n'y a plus d'espace sur l'appareil. L'opération aurait causé le dépassement par le système de fichiers de la limite du serveur.

15.1.4.10 NFS4ERR_NOTEMPTY (code d'erreur 66)

Tentative de suppression d'un répertoire qui n'est pas vide.

15.1.4.11 NFS4ERR_ROFS (code d'erreur 30)

Indique un système de fichiers en lecture seule. Une opération de modification a été tentée sur un système de fichiers en lecture seule.

15.1.4.12 NFS4ERR_XDEV (code d'erreur 18)

Indique une tentative de faire une opération, comme une liaison, qui franchit une frontière inappropriée. Cela peut être dû à des frontières comme :

* celles entre systèmes de fichiers (où les fsid sont différents) ;
* celles entre différents répertoires d'attributs nommés ou entre un répertoire des attributs nommés et un répertoire ordinaire ;
* celles entre des gammes d'octets d'un système de fichiers que la mise en œuvre de système de fichiers traite comme séparées (par exemple, pour les besoins de comptabilité de l'espace) et où l'interconnexion entre les gammes d'octets n'est pas permise.

15.1.5 Erreurs de gestion d'état

Ces erreurs indiquent des problèmes avec l'identifiant d'état (ou un des identifiants d'état) passé à une opération donnée. Cela inclut des situations dans lesquelles l'identifiant d'état est invalide ainsi que des situations dans lesquelles l'identifiant d'état est valide mais désigne l'état de verrouillage qui a été révoqué. Selon l'opération, l'identifiant d'état quand il est valide peut désigner des ouvertures, des verrous de gamme d'octets, des délégations de fichier ou de répertoire, des dispositions, ou des transpositions d'appareil.

15.1.5.1 NFS4ERR_ADMIN_REVOKED (code d'erreur 10047)

Un identifiant d'état désigne un état de verrouillage d'un type qui a été révoqué du fait d'une interaction administrative, éventuellement alors que le prêt est valide.

15.1.5.2 NFS4ERR_BAD_STATEID (code d'erreur 10026)

Un identifiant d'état ne désigne pas de façon appropriée un état valide. Voir aux paragraphes 8.2.4 et 8.2.3 la discussion sur la validation des identifiants d'état.

15.1.5.3 NFS4ERR_DELEG_REVOKED (code d'erreur 10087)

Un identifiant d'état désigne un état de verrouillage rappelable de n'importe quel type (délégation ou disposition) qui a été révoqué suite à la défaillance du client de retourner le verrou quand il a été rappelé.

15.1.5.4 NFS4ERR_EXPIRED (code d'erreur 10011)

Un identifiant d'état désigne un état de verrouillage d'un type quelconque révoqué suite à l'expiration du prêt du client, soit immédiatement à l'expiration du prêt, soit suite à une demande ultérieure pour un verrou en conflit.

15.1.5.5 NFS4ERR_OLD_STATEID (code d'erreur 10024)

Un identifiant d'état avec une valeur de seqid non zéro correspond au seqid courant pour l'état désigné par l'utilisateur.

15.1.6 Erreurs de sécurité

Il y a diverses erreurs en relation avec des permissions dans NFSv4.1.

15.1.6.1 NFS4ERR_ACCESS (code d'erreur 13)

Indique une permission refusée. L'appelant n'a pas la permission correcte d'effectuer l'opération demandée. À comparer avec NFS4ERR_PERM (paragraphe 15.1.6.2) qui se restreint aux défaillances de permission de propriétaire ou d'utilisateur privilégié, et avec NFS4ERR_WRONG_CRED (paragraphe 15.1.6.4) qui traite de la permission appropriée de supprimer ou modifier des objets transitoires sur la base des accréditifs de l'utilisateur qui les a créés.

15.1.6.2 NFS4ERR_PERM (code d'erreur 1)

Indique que le demandeur n'est pas le propriétaire. L'opération n'est pas permise parce que l'appelant n'est ni un utilisateur privilégié (racine) ni le propriétaire de la cible de l'opération.

15.1.6.3 NFS4ERR_WRONGSEC (code d'erreur 10016)

Indique que le mécanisme de sécurité utilisé par le client pour l'opération ne correspond pas à la politique de sécurité du serveur. Le client devrait changer le mécanisme de sécurité utilisé et envoyer à nouveau l'opération (mais pas avec le même identifiant de créneau et identifiant de séquence ; un ou les deux DOIT être différent sur le nouvel envoi). SECINFO et SECINFO_NO_NAME peut être utilisé pour déterminer le mécanisme approprié.

15.1.6.4 NFS4ERR_WRONG_CRED (code d'erreur 10082)

Une opération qui manipule l'état a été tentée par un principal à qui il n'est pas permis de modifier cet élément d'état.

15.1.7 Erreurs de noms

Les noms dans NFSv4 sont des chaînes UTF-8. Quand les chaînes ne sont pas de l'UTF-8 valide ou sont de longueur zéro, l'erreur NFS4ERR_INVAL en résulte. À côté de cela, il y a un certain nombre d'autres erreurs pour indiquer des problèmes spécifiques avec les noms.

15.1.7.1 NFS4ERR_BADCHAR (code d'erreur 10040)

Une chaîne UTF-8 contient un caractère non pris en charge par le serveur dans le contexte utilisé.

15.1.7.2 NFS4ERR_BADNAME (code d'erreur 10041)

Une chaîne de nom dans une demande consiste en des caractères UTF-8 valides pris en charge par le serveur, mais le nom n'est pas pris en charge par le serveur comme nom valide pour l'opération en cours. Un exemple pourrait être de créer un fichier ou répertoire nommé ".." sur un serveur dont le système de fichiers utilise ce nom pour les liaisons aux répertoires parents.

15.1.7.3 NFS4ERR_NAMETOOLONG (code d'erreur 63)

Retourné quand le nom de fichier dans une opération excède la limite de mise en œuvre du serveur.

15.1.8 Erreurs de verrouillage

Cette section traite des erreurs relatives au verrouillage, aussi bien de partage de réservation que de gamme d'octets. Il ne traite pas des erreurs spécifiques du processus de réclamation de verrous. Celles-ci sont traitées au paragraphe 15.1.9.

15.1.8.1 NFS4ERR_BAD_RANGE (code d'erreur 10042)

La gamme d'octets d'une opération LOCK, LOCKT, ou LOCKU n'est pas permise par le serveur. Par exemple, cette erreur résulte quand un serveur qui prend seulement en charge de gammes de 32 bits reçoit une gamme qui ne peut pas être traitée par ce serveur. (Voir au paragraphe 18.10.3.)

15.1.8.2 NFS4ERR_DEADLOCK (code d'erreur 10045)

Le serveur a été capable de déterminer une condition d'impasse de verrouillage de gamme d'octets pour une opération LOCK READW_LT ou WRITEW_LT.

15.1.8.3 NFS4ERR_DENIED (code d'erreur 10010)

Une tentative de verrouillage d'un fichier est refusée. Comme ce peut être une condition temporaire, le client est encouragé à envoyer à nouveau la demande de verrou (mais pas avec le même identifiant de créneau et identifiant de séquence ; un ou les deux DOIT être différent sur le nouvel envoi) jusqu'à ce que le verrou soit accepté. Voir au paragraphe 9.6 la discussion du nouvel envoi.

15.1.8.4 NFS4ERR_LOCKED (code d'erreur 10012)

Une opération READ ou WRITE a été tentée sur un fichier où il y a un conflit entre l'entrée/sortie et un verrou existant :

* il y a un partage de réservation non cohérent avec l'entrée/sortie ;
* la gamme à lire ou écrire recoupe un verrou de gamme d'octets obligatoire existant.

15.1.8.5 NFS4ERR_LOCKS_HELD (code d'erreur 10037)

Une opération a été empêchée par la présence inattendue de verrous.

15.1.8.6 NFS4ERR_LOCK_NOTSUPP (code d'erreur 10043)

Une opération LOCK a été tentée qui exigerait une dégradation ou mise à niveau d'une gamme de verrous de gamme d'octets déjà détenue par le propriétaire, et le serveur ne prend pas en charge la dégradation ou mise à niveau atomique de verrous.

15.1.8.7 NFS4ERR_LOCK_RANGE (code d'erreur 10028)

Une opération LOCK opère sur une gamme qui se chevauche partiellement avec un verrou de gamme d'octets détenu actuellement par le propriétaire de verrou et ne correspond pas précisément à un seul verrou de gamme d'octets lorsque le serveur ne prend pas en charge ce type de demande, et donc ne met pas en œuvre la sémantique de verrouillage POSIX [IEEE-1003.1]. Voir respectivement aux paragraphes 18.10.4, 18.11.4, et 18.12.4 la discussion sur comment cela s'applique à LOCK, LOCKT, et LOCKU.

15.1.8.8 NFS4ERR_OPENMODE (code d'erreur 10038)

Le client a tenté un READ, WRITE, LOCK, ou autre opération non sanctionnée par l'identifiant d'état passé (par exemple, écrire sur un fichier ouvert pour un accès en lecture seule).

15.1.8.9 NFS4ERR_SHARE_DENIED (code d'erreur 10015)

Une tentative de OPEN sur un fichier avec un partage de réservation qui a échoué à cause d'un conflit de partage.

15.1.9 Erreurs de réclamation

Ces erreurs se rapportent au processus de réclamation de verrous après un redémarrage de serveur.

15.1.9.1 NFS4ERR_COMPLETE_ALREADY (code d'erreur 10054)

Le client a précédemment envoyé une opération réussie RECLAIM_COMPLETE spécifiant la même portée, si cette portée est globale ou pour le même système de fichiers dans le cas d'un RECLAIM_COMPLETE par système de fichiers. Une opération RECLAIM_COMPLETE supplémentaire n'est pas nécessaire et résulte en cette erreur.

15.1.9.2 NFS4ERR_GRACE (code d'erreur 10013)

Cette erreur est retournée quand le serveur est dans sa période de grâce à l'égard de l'objet de système de fichiers pour lequel le verrou était demandé. Dans cette situation, une demande de verrouillage non de réclamation ne peut pas être accordée. Cela peut survenir parce que :

* le serveur n'a pas des informations suffisantes sur les verrous qui pourraient être éventuellement réclamés pour déterminer si le verrou pourrait être accordé ;
* la demande est faite par un client chargé de réclamer ses verrous qui n'a pas encore fait l'opération RECLAIM_COMPLETE appropriée, lui permettant de procéder à l'obtention de nouveaux verrous.

Dans le cas de période de grâce par système de fichiers, il peut y avoir des clients (c'est-à-dire, ceux qui utilisent actuellement le système de fichiers de destination) qui pourraient ne pas connaître les circonstances résultant en l'initiation de la période de grâce. Ces clients doivent périodiquement réessayer la demande jusqu'à ce que la période de grâce soit finie, juste comme le font les autres clients.

15.1.9.3 NFS4ERR_NO_GRACE (code d'erreur 10033)

Une réclamation de l'état de client a été tentée dans des circonstances où le serveur ne peut pas garantir qu'un état en conflit n'a pas été fourni à un autre client. Cela se produit dans une des situations suivantes :

* il n'y a pas de période de grâce active qui s'applique à l'objet de système de fichiers pour lequel la demande a été faite ;
* le client qui fait la demande n'a pas de rôle actuel dans la réclamation de verrous ;
* les opérations précédentes ont créé une situation dans laquelle le serveur n'est pas capable de déterminer qu'une condition à la bordure d'une réclamation interférente n'existe pas.

15.1.9.4 NFS4ERR_RECLAIM_BAD (code d'erreur 10034)

Le serveur a déterminé qu'une réclamation tentée par le client n'est pas valide, c'est-à-dire, le verrou spécifié comme étant réclamé ne pourrait pas avoir existé avant le redémarrage de serveur ou l'événement de migration du système de fichiers. Un serveur n'est pas obligé de faire cette détermination et va normalement s'appuyer sur le client pour seulement réclamer les verrous qui étaient accordés au client avant le redémarrage. Cependant, quand un serveur a bien des informations fiables pour lui permettre de faire cette détermination, cette erreur indique que la réclamation a été rejetée comme invalide. C'est opposé à l'erreur NFS4ERR_RECLAIM_CONFLICT (voir le paragraphe 15.1.9.5) où le serveur peut seulement déterminer qu'il y a eu une réclamation invalide, mais ne peut pas déterminer quelle demande est invalide.

15.1.9.5 NFS4ERR_RECLAIM_CONFLICT (code d'erreur 10035)

La réclamation tentée par le client a rencontré un conflit et ne peut pas être satisfaite. Cela indique potentiellement un client qui se comporte mal, bien que pas nécessairement celui recevant l'erreur. Le mauvais comportement pourrait être de la part du client qui a établi le verrou avec lequel ce client est en conflit. Voir aussi au paragraphe 15.1.9.4 l'erreur en relation, NFS4ERR_RECLAIM_BAD.

15.1.10 Erreurs de pNFS

Cette section traite des erreurs relatives à pNFS, incluant celles associées à l'utilisation de NFSv4.1 pour communiquer avec un serveur de données.

15.1.10.1 NFS4ERR_BADIOMODE (code d'erreur 10049)

Une disposition iomode invalide ou inappropriée a été spécifiée. Pour un exemple d'une disposition iomode inappropriée, supposons l'opération LAYOUTGET d'un client qui spécifie un iomode de LAYOUTIOMODE4_RW, et le serveur n'est ni capable ni désireux de laisser le client envoyer des demandes d'écriture aux serveurs de données ; le serveur peut répondre avec NFS4ERR_BADIOMODE. Le client va alors envoyer un autre LAYOUTGET avec un iomode de

LAYOUTIOMODE4_READ.

15.1.10.2 NFS4ERR_BADLAYOUT (code d'erreur 10050)

La disposition spécifiée est invalide d'une certaine façon. Pour LAYOUTCOMMIT, cela indique que la disposition spécifiée n'est pas détenue par le client ou n'est pas de mode LAYOUTIOMODE4_RW. Pour LAYOUTGET, il indique qu'une disposition correspondant à la spécification du client quant à la longueur minimum ne peut pas être accordée.

15.1.10.3 NFS4ERR_LAYOUTTRYLATER (code d'erreur 10058)

Les dispositions sont temporairement indisponibles pour le fichier. Le client devrait envoyer à nouveau plus tard (mais pas avec le même identifiant de créneau et identifiant de séquence ; un ou les deux DOIT être différent sur le nouvel envoi).

15.1.10.4 NFS4ERR_LAYOUTUNAVAILABLE (code d'erreur 10059)

Retourné quand les dispositions ne sont pas disponibles pour le système de fichiers courant ou le fichier particulier spécifié.

15.1.10.5 NFS4ERR_NOMATCHING_LAYOUT (code d'erreur 10060)

Retourné quand les dispositions sont rappelées et que le client n'a pas de dispositions correspondant à la spécification des dispositions rappelées.

15.1.10.6 NFS4ERR_PNFS_IO_HOLE (code d'erreur 10075)

Le client pNFS a tenté de lire ou écrire sur un trou illégal d'un fichier d'un serveur de données qui utilise une mise en paquets éparse. Voir au paragraphe 13.4.4.

15.1.10.7 NFS4ERR_PNFS_NO_LAYOUT (code d'erreur 10080)

Le client pNFS a tenté de lire ou écrire sur un fichier (en utilisant une demande à un serveur de données) sans détenir une disposition valide. Cela inclut le cas où le client a une disposition, mais le iomode ne permet pas de WRITE.

15.1.10.8 NFS4ERR_RETURNCONFLICT (code d'erreur 10086)

Une disposition est indisponible du fait d'une tentative d'effectuer le LAYOUTGET avant qu'ait été reçu un LAYOUTRETURN en instance sur le fichier. Voir au paragraphe 12.5.5.2.1.3.

15.1.10.9 NFS4ERR_UNKNOWN_LAYOUTTYPE (code d'erreur 10062)

Le client a spécifié un type de disposition qui n'est pas pris en charge par le serveur.

15.1.11 Erreurs d'utilisation de session

Cette section traite des erreurs rencontrées quand on utilise des sessions, c'est-à-dire, des erreurs rencontrées quand une demande utilise une opération Sequence (c'est-à-dire, SEQUENCE ou CB_SEQUENCE).

15.1.11.1 NFS4ERR_BADSESSION (code d'erreur 10052)

L'identifiant de session spécifié est inconnu du serveur auquel l'opération est adressée.

15.1.11.2 NFS4ERR_BADSLOT (code d'erreur 10053)

Le demandeur a envoyé une opération SEQUENCE qui a tenté d'utiliser un créneau que le répondeur n'a pas dans son tableau de créneaux. Il est possible que le créneau ait été retiré.

15.1.11.3 NFS4ERR_BAD_HIGH_SLOT (code d'erreur 10077)

L'argument highest_slot dans une opération SEQUENCE excède le highest_slotid appliqué par le répondeur.

15.1.11.4 NFS4ERR_CB_PATH_DOWN (code d'erreur 10048)

Il y a un problème pour contacter le client via le chemin de rappel. La fonction de cette erreur a été largement supplantée par l'utilisation des fanions d'état dans la réponse à l'opération SEQUENCE (voir le paragraphe 18.46).

15.1.11.5 NFS4ERR_DEADSESSION (code d'erreur 10078)

La session spécifiée est une session persistante qui est morte et n'accepte pas de nouvelles demandes ni n'effectue de nouvelles opérations sur les demandes existantes (dans le cas où une demande est exécutée partiellement avant le redémarrage du serveur).

15.1.11.6 NFS4ERR_CONN_NOT_BOUND_TO_SESSION (code d'erreur 10055)

Une opération SEQUENCE a été envoyée sur une connexion qui n'était pas associée à la session spécifiée, alors que le client avait spécifié que l'association de connexion devait être appliquée avec la protection d'état SP4_MACH_CRED ou SP4_SSV.

15.1.11.7 NFS4ERR_SEQ_FAUX_RETRY (code d'erreur 10076)

Le demandeur a envoyé une opération SEQUENCE avec un identifiant de créneau et un identifiant de séquence qui sont dans la mémoire tampon de réponse, mais le répondeur a détecté que la demande réessayée n'est pas la même que la demande d'origine. Voir au paragraphe 2.10.6.1.3.1.

15.1.11.8 NFS4ERR_SEQ_MISORDERED (code d'erreur 10063)

Le demandeur a envoyé une opération SEQUENCE avec un identifiant de séquence invalide.

15.1.12 Session Management Errors

Cette section traite des erreurs associées aux demandes utilisées dans la gestion de session.

15.1.12.1 NFS4ERR_BACK_CHAN_BUSY (code d'erreur 10057)

Une tentative a été faite de détruire une session alors que la session ne peut pas être détruite parce que le serveur a des demandes de rappel en instance.

15.1.12.2 NFS4ERR_BAD_SESSION_DIGEST (code d'erreur 10051)

Le résumé utilisé dans une demande SET_SSV n'est pas valide.

15.1.13 Erreurs de gestion de client

Cette section traite des erreurs associées aux demandes utilisées pour créer et gérer les identifiants de client.

15.1.13.1 NFS4ERR_CLIENTID_BUSY (code d'erreur 10074)

L'opération DESTROY_CLIENTID a trouvé qu'il y a des sessions et/ou de l'état non expirés associés à l'identifiant de client à détruire.

15.1.13.2 NFS4ERR_CLID_INUSE (code d'erreur 10017)

Pendant le traitement d'une opération EXCHANGE_ID, le serveur a été présenté avec un champ co_ownerid qui correspond à un client existant avec un état de prêt valide, mais le principal qui envoie l'opération EXCHANGE_ID diffère du principal qui a établi le client existant. Cela indique une collision (très probablement par hasard) entre les clients. Le client devrait récupérer en changeant le co_ownerid et en envoyant à nouveau l'EXCHANGE_ID (mais pas avec le même identifiant de créneau et identifiant de séquence ; un ou les deux DOIT être différent sur le nouvel envoi).

15.1.13.3 NFS4ERR_ENCR_ALG_UNSUPP (code d'erreur 10079)

Un EXCHANGE_ID a été envoyé qui spécifie la protection d'état via SSV, et où l'ensemble des algorithmes de chiffrement présentés par le client n'incluait pas de prise en charge par le serveur.

15.1.13.4 NFS4ERR_HASH_ALG_UNSUPP (code d'erreur 10072)

Un EXCHANGE_ID a été envoyé qui spécifiait la protection d'état via SSV, et où l'ensemble des algorithmes de hachage présenté par le client n'incluait aucune prise en charge par le serveur.

15.1.13.5 NFS4ERR_STALE_CLIENTID (code d'erreur 10022)

Un identifiant de client non reconnu par le serveur a été passé à une opération. Noter qu'à la différence du cas de NFSv4.0, les identifiants de client ne sont pas passés explicitement au serveur dans les opérations ordinaires de verrouillage et ne peuvent pas avoir cette erreur pour résultat. À la place, quand il y a un redémarrage de serveur, il est d'abord manifesté par une erreur sur la session associée, et la péremption de l'identifiant de client est détectée quand on essaye d'associer un identifiant de client à une nouvelle session.

15.1.14 Erreurs de délégation

Cette section traite des erreurs associées aux demandes et retours de délégations.

15.1.14.1 NFS4ERR_DELEG_ALREADY_WANTED (code d'erreur 10056)

Le client a demandé une délégation quand il avait déjà enregistré qu'il veut cette même délégation.

15.1.14.2 NFS4ERR_DIRDELEG_UNAVAIL (code d'erreur 10084)

Cette erreur est retournée quand le serveur est incapable ou ne veut pas fournir une délégation de répertoire demandée.

15.1.14.3 NFS4ERR_RECALLCONFLICT (code d'erreur 10061)

Un objet rappelable (c'est-à-dire, une disposition ou délégation) est indisponible du fait d'une opération de rappel en conflit qui est actuellement en cours pour cet objet.

15.1.14.4 NFS4ERR_REJECT_DELEG (code d'erreur 10085)

L'opération de rappel invoquée pour traiter une nouvelle délégation l'a rejetée.

15.1.15 Erreurs de traitement d'attribut

Cette section traite des erreurs spécifiques de l'attribut traitant de NFSv4.

15.1.15.1 NFS4ERR_ATTRNOTSUPP (code d'erreur 10032)

Un attribut spécifié n'est pas pris en charge par le serveur. Cette erreur NE DOIT PAS être retournée par l'opération GETATTR.

15.1.15.2 NFS4ERR_BADOWNER (code d'erreur 10039)

Cette erreur est retournée quand un propriétaire ou une valeur d'attribut owner_group ou le champ who d'un ACE dans une valeur d'attribut d'ACL ne peut pas être traduit en une représentation locale.

15.1.15.3 NFS4ERR_NOT_SAME (code d'erreur 10027)

Cette erreur est retournée par l'opération VERIFY pour signifier que les attributs comparés n'étaient pas les mêmes que ceux fournis dans la demande du client.

15.1.15.4 NFS4ERR_SAME (code d'erreur 10009)

Cette erreur est retournée par l'opération NVERIFY pour signifier que les attributs comparés étaient les mêmes que ceux fournis dans la demande du client

15.1.16 Erreurs obsolètes

Ces erreurs NE DOIVENT PAS être générées par une opération NFSv4.1. Ce peut être pour un certain nombre de raisons.

* La fonction fournie par l'erreur a été supplantée par un des bits d'état retourné par l'opération SEQUENCE.

* La nouvelle structure de session et le changement associé dans le verrouillage ont rendu l'erreur inutile.
* Il y a eu une restructuration de certaines erreurs pour NFSv4.1 qui a résulté en l'élimination de certaines erreurs.

15.1.16.1 NFS4ERR_BAD_SEQID (code d'erreur 10026)

Le numéro de séquence (seqid) dans une demande de verrouillage n'est ni le prochain numéro attendu ni le dernier numéro traité. Ces numéros de séquence sont ignorés dans NFSv4.1.

15.1.16.2 NFS4ERR_LEASE_MOVED (code d'erreur 10031)

Un prêt à renouveler est associé à un système de fichiers qui a migré sur un nouveau serveur. L'erreur a été supplantée par le bit d'état SEQ4_STATUS_LEASE_MOVED (voir le paragraphe 18.46).

15.1.16.3 NFS4ERR_NXIO (code d'erreur 5)

Erreur d'entrée/sortie. Pas de tel appareil ou adresse. Cette erreur est pour des erreurs qui impliquent l'accès à l'appareil de bloc et caractère, mais parce que NFSv4.1 n'est pas un protocole d'accès d'appareil, cette erreur n'est pas applicable.

15.1.16.4 NFS4ERR_RESTOREFH (code d'erreur 10030)

L'opération RESTOREFH n'a pas de bride de fichier sauvegardée (identifiée par SAVEFH) sur laquelle opérer. Dans NFSv4.1, cette erreur a été remplacée par NFS4ERR_NOFILEHANDLE.

15.1.16.5 NFS4ERR_STALE_STATEID (code d'erreur 10023)

Un identifiant d'état généré par une instance de serveur antérieure a été utilisée. Cette erreur est discutable dans NFSv4.1 parce que toutes les opérations qui prennent un identifiant d'état DOIVENT être précédées de l'opération SEQUENCE, et l'instance de serveur précédente est détectée par l'infrastructure de session qui prend en charge SEQUENCE.

15.2 Opérations et leurs erreurs valides

Ce paragraphe contient un tableau qui donne l'erreur valide retournée pour chaque opération du protocole. Le code d'erreur NFS4_OK (indiquant qu'il n'y a pas d'erreur) n'est pas mentionné mais devrait être compris comme étant retournable par toutes les opérations avec deux importantes exceptions :

* les opérations qui NE DOIVENT PAS être mises en œuvre : OPEN_CONFIRM, RELEASE_LOCKOWNER, RENEW, SETCLIENTID, et SETCLIENTID_CONFIRM,
* l'opération invalide ILLEGAL.

Opération	Erreurs
ACCESS	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS
BACKCHANNEL_CTL	NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOENT, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_TOO_MANY_OPS
BIND_CONN_TO_ SESSION	NFS4ERR_BADSESSION, NFS4ERR_BADXDR, NFS4ERR_BAD_SESSION_DIGEST, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOT_ONLY_OP, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CLOSE	NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_LOCKS_HELD, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG,

	NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
COMMIT	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_IO, NFS4ERR_ISDIR, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
CREATE	NFS4ERR_ACCESS, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADOWNER, NFS4ERR_BADTYPE, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DQUOT, NFS4ERR_EXIST, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MLINK, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTDIR, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PERM, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNSAFE_COMPOUND
CREATE_SESSION	NFS4ERR_BADXDR, NFS4ERR_CLID_INUSE, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOENT, NFS4ERR_NOT_ONLY_OP, NFS4ERR_NOSPC, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SEQ_MISORDERED, NFS4ERR_SERVERFAULT, NFS4ERR_STALE_CLIENTID, NFS4ERR_TOOSMALL, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
DELEGPURGE	NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
DELEGRETURN	NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
DESTROY_CLIENTID	NFS4ERR_BADXDR, NFS4ERR_CLIENTID_BUSY, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_NOT_ONLY_OP, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE_CLIENTID, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
DESTROY_SESSION	NFS4ERR_BACK_CHAN_BUSY, NFS4ERR_BADSESSION, NFS4ERR_BADXDR, NFS4ERR_CB_PATH_DOWN, NFS4ERR_CONN_NOT_BOUND_TO_SESSION, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_NOT_ONLY_OP, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE_CLIENTID, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
EXCHANGE_ID	NFS4ERR_BADCHAR, NFS4ERR_BADXDR, NFS4ERR_CLID_INUSE, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_ENCR_ALG_UNSUPP, NFS4ERR_HASH_ALG_UNSUPP, NFS4ERR_INVAL, NFS4ERR_NOENT, NFS4ERR_NOT_ONLY_OP, NFS4ERR_NOT_SAME, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
FREE_STATEID	NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION,

	NFS4ERR_DELAY, NFS4ERR_LOCKS_HELD, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
GET_DIR_DELEGATION	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DIRDELEG_UNAVAIL, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS
GETATTR	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
GETDEVICEINFO	NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOENT, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOOSMALL, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE
GETDEVICELIST	NFS4ERR_BADXDR, NFS4ERR_BAD_COOKIE, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_NOT_SAME, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE
GETFH	NFS4ERR_FHEXPIRED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_STALE
ILLEGAL	NFS4ERR_BADXDR, NFS4ERR_OP_ILLEGAL
LAYOUTCOMMIT	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADIOMODE, NFS4ERR_BADLAYOUT, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_EXPIRED, NFS4ERR_FBIG, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_ISDIR NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_NO_GRACE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_RECLAIM_BAD, NFS4ERR_RECLAIM_CONFLICT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_CRED
LAYOUTGET	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADIOMODE, NFS4ERR_BADLAYOUT, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_LAYOUTTRYLATER, NFS4ERR_LAYOUTUNAVAILABLE, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTSUPP, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_RECALLCONFLICT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOOSMALL, NFS4ERR_TOO_MANY_OPS,

	NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_TYPE
LAYOUTRETURN	NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_NO_GRACE, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_CRED, NFS4ERR_WRONG_TYPE
LINK	NFS4ERR_ACCESS, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DQUOT, NFS4ERR_EXIST, NFS4ERR_FHEXPIRED, NFS4ERR_FILE_OPEN, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_IO, NFS4ERR_MLINK, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTDIR, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC, NFS4ERR_WRONG_TYPE, NFS4ERR_XDEV
LOCK	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_RANGE, NFS4ERR_BAD_STATEID, NFS4ERR_DEADLOCK, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DENIED, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_LOCK_NOTSUPP, NFS4ERR_LOCK_RANGE, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NO_GRACE, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_RECLAIM_BAD, NFS4ERR_RECLAIM_CONFLICT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED, NFS4ERR_WRONG_TYPE
LOCKT	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_BAD_RANGE, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DENIED, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_LOCK_RANGE, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED, NFS4ERR_WRONG_TYPE
LOCKU	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_RANGE, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_LOCK_RANGE, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
LOOKUP	NFS4ERR_ACCESS, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC

LOOKUPP	NFS4ERR_ACCESS, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC
NVERIFY	NFS4ERR_ACCESS, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADCHAR, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SAME, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_TYPE
OPEN	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADOWNER, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_ALREADY_WANTED, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_EXIST, NFS4ERR_EXPIRED, NFS4ERR_FBIG, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTDIR, NFS4ERR_NO_GRACE, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PERM, NFS4ERR_RECLAIM_BAD, NFS4ERR_RECLAIM_CONFLICT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_SHARE_DENIED, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNSAFE_COMPOUND, NFS4ERR_WRONGSEC, NFS4ERR_WRONG_TYPE
OPEN_CONFIRM	NFS4ERR_NOTSUPP
OPEN_DOWNGRADE	NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED
OPENATTR	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DQUOT, NFS4ERR_FHEXPIRED, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNSAFE_COMPOUND, NFS4ERR_WRONG_TYPE
PUTFH	NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_MOVED, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC
PUTPUBFH	NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC
PUTROOTFH	NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE,

	NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC|
READ	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_IO, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PNFS_IO_HOLE, NFS4ERR_PNFS_NO_LAYOUT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
READDIR	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_BAD_COOKIE, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_NOT_SAME, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOOSMALL, NFS4ERR_TOO_MANY_OPS
READLINK	NFS4ERR_ACCESS, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
RECLAIM_COMPLETE	NFS4ERR_BADXDR, NFS4ERR_COMPLETE_ALREADY, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_CRED, NFS4ERR_WRONG_TYPE
RELEASE_LOCKOWNER	NFS4ERR_NOTSUPP
REMOVE	NFS4ERR_ACCESS, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_FILE_OPEN, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_NOTEMPTY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS
RENAME	NFS4ERR_ACCESS, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DQUOT, NFS4ERR_EXIST, NFS4ERR_FHEXPIRED, NFS4ERR_FILE_OPEN, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MLINK, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTDIR, NFS4ERR_NOTEMPTY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC, NFS4ERR_XDEV
RENEW	NFS4ERR_NOTSUPP
RESTOREFH	NFS4ERR_DEADSESSION, NFS4ERR_FHEXPIRED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP,

	NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC
SAVEFH	NFS4ERR_DEADSESSION, NFS4ERR_FHEXPIRED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS
SECINFO	NFS4ERR_ACCESS, NFS4ERR_BADCHAR, NFS4ERR_BADNAME, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS
SECINFO_NO_NAME	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_MOVED, NFS4ERR_NOENT, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTDIR, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS
SEQUENCE	NFS4ERR_BADSESSION, NFS4ERR_BADSLOT, NFS4ERR_BADXDR, NFS4ERR_BAD_HIGH_SLOT, NFS4ERR_CONN_NOT_BOUND_TO_SESSION, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SEQUENCE_POS, NFS4ERR_SEQ_FAUX_RETRY, NFS4ERR_SEQ_MISORDERED, NFS4ERR_TOO_MANY_OPS
SET_SSV	NFS4ERR_BADXDR, NFS4ERR_BAD_SESSION_DIGEST, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_TOO_MANY_OPS
SETATTR	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADCHAR, NFS4ERR_BADOWNER, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_EXPIRED, NFS4ERR_FBIG, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PERM, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_TYPE
SETCLIENTID	NFS4ERR_NOTSUPP
SETCLIENTID_CONFIRM	NFS4ERR_NOTSUPP
TEST_STATEID	NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
VERIFY	NFS4ERR_ACCESS, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADCHAR, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOT_SAME, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE,

	NFS4ERR_WRONG_TYPE
WANT_DELEGATION	NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_ALREADY_WANTED, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_NO_GRACE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_RECALLCONFLICT, NFS4ERR_RECLAIM_BAD, NFS4ERR_RECLAIM_CONFLICT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
WRITE	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_EXPIRED, NFS4ERR_FBIG, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_ISDIR, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PNFS_IO_HOLE, NFS4ERR_PNFS_NO_LAYOUT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE

Tableau 12 : Retours d'erreur valides pour chaque opération de protocole

15.3 Opérations de rappel et leurs erreurs valides

Ce paragraphe contient un tableau qui donne l'erreur valide retournée pour chaque opération de rappel. Le code d'erreur NFS4_OK (indiquant pas d'erreur) n'est pas mentionné mais devrait être compris comme étant retournable par toutes les opérations de rappel à l'exception de CB_ILLEGAL.

Opération de rappel	**Erreurs**
CB_GETATTR	NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CB_ILLEGAL	NFS4ERR_BADXDR, NFS4ERR_OP_ILLEGAL
CB_LAYOUTRECALL	NFS4ERR_BADHANDLE, NFS4ERR_BADIOMODE, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOMATCHING_LAYOUT, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_TYPE
CB_NOTIFY	NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CB_NOTIFY_DEVICEID	NFS4ERR_BADXDR, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CB_NOTIFY_LOCK	NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DELAY, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE,

	NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CB_PUSH_DELEG	NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REJECT_DELEG, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
CB_RECALL	NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DELAY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CB_RECALL_ANY	NFS4ERR_BADXDR, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_TOO_MANY_OPS
CB_RECALLABLE_OBJ_AVAIL	NFS4ERR_BADXDR, NFS4ERR_DELAY,NFS4ERR_INVAL, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS
CB_RECALL_SLOT	NFS4ERR_BADXDR, NFS4ERR_BAD_HIGH_SLOT, NFS4ERR_DELAY, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_TOO_MANY_OPS
CB_SEQUENCE	NFS4ERR_BADSESSION, NFS4ERR_BADSLOT, NFS4ERR_BADXDR, NFS4ERR_BAD_HIGH_SLOT, NFS4ERR_CONN_NOT_BOUND_TO_SESSION, NFS4ERR_DELAY, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SEQUENCE_POS, NFS4ERR_SEQ_FAUX_RETRY, NFS4ERR_SEQ_MISORDERED, NFS4ERR_TOO_MANY_OPS
CB_WANTS_CANCELLED	NFS4ERR_BADXDR, NFS4ERR_DELAY, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_TOO_MANY_OPS

Tableau 13 : Retours d'erreur valides pour chaque opération de rappel du protocole

15.4 Erreurs et opérations qui les utilisent

Erreur	Opérations
NFS4ERR_ACCESS	ACCESS, COMMIT, CREATE, GETATTR, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, READ, READDIR, READLINK, REMOVE, RENAME, SECINFO, SECINFO_NO_NAME, SETATTR, VERIFY, WRITE
NFS4ERR_ADMIN_REVOKED	CLOSE, DELEGRETURN, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LOCK, LOCKU, OPEN, OPEN_DOWNGRADE, READ, SETATTR, WRITE
NFS4ERR_ATTRNOTSUPP	CREATE, LAYOUTCOMMIT, NVERIFY, OPEN, SETATTR, VERIFY
NFS4ERR_BACK_CHAN_BUSY	DESTROY_SESSION
NFS4ERR_BADCHAR	CREATE, EXCHANGE_ID, LINK, LOOKUP, NVERIFY, OPEN, REMOVE, RENAME, SECINFO, SETATTR, VERIFY

NFS4ERR_BADHANDLE	CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, PUTFH
NFS4ERR_BADIOMODE	CB_LAYOUTRECALL, LAYOUTCOMMIT, LAYOUTGET
NFS4ERR_BADLAYOUT	LAYOUTCOMMIT, LAYOUTGET
NFS4ERR_BADNAME	CREATE, LINK, LOOKUP, OPEN, REMOVE, RENAME, SECINFO
NFS4ERR_BADOWNER	CREATE, OPEN, SETATTR
NFS4ERR_BADSESSION	BIND_CONN_TO_SESSION, CB_SEQUENCE, DESTROY_SESSION, SEQUENCE
NFS4ERR_BADSLOT	CB_SEQUENCE, SEQUENCE
NFS4ERR_BADTYPE	CREATE
NFS4ERR_BADXDR	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_ILLEGAL, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, ILLEGAL, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, READ, READDIR, RECLAIM_COMPLETE, REMOVE, RENAME, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_BAD_COOKIE	GETDEVICELIST, READDIR
NFS4ERR_BAD_HIGH_SLOT	CB_RECALL_SLOT, CB_SEQUENCE, SEQUENCE
NFS4ERR_BAD_RANGE	LOCK, LOCKT, LOCKU
NFS4ERR_BAD_SESSION_ DIGEST	BIND_CONN_TO_SESSION, SET_SSV
NFS4ERR_BAD_STATEID	CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_LOCK, CB_RECALL, CLOSE, DELEGRETURN, FREE_STATEID, LAYOUTGET, LAYOUTRETURN, LOCK, LOCKU, OPEN, OPEN_DOWNGRADE, READ, SETATTR, WRITE
NFS4ERR_CB_PATH_DOWN	DESTROY_SESSION
NFS4ERR_CLID_INUSE	CREATE_SESSION, EXCHANGE_ID
NFS4ERR_CLIENTID_BUSY	DESTROY_CLIENTID
NFS4ERR_COMPLETE_ ALREADY	RECLAIM_COMPLETE
NFS4ERR_CONN_NOT_ BOUND_TO_SESSION	CB_SEQUENCE, DESTROY_SESSION, SEQUENCE
NFS4ERR_DEADLOCK	LOCK
NFS4ERR_DEADSESSION	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE

NFS4ERR_DELAY	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_DELEG_ALREADY_WANTED	OPEN, WANT_DELEGATION
NFS4ERR_DELEG_REVOKED	DELEGRETURN, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, OPEN, READ, SETATTR, WRITE
NFS4ERR_DENIED	LOCK, LOCKT
NFS4ERR_DIRDELEG_UNAVAIL	GET_DIR_DELEGATION
NFS4ERR_DQUOT	CREATE, LAYOUTGET, LINK, OPEN, OPENATTR, RENAME, SETATTR, WRITE
NFS4ERR_ENCR_ALG_UNSUPP	EXCHANGE_ID
NFS4ERR_EXIST	CREATE, LINK, OPEN, RENAME
NFS4ERR_EXPIRED	CLOSE, DELEGRETURN, LAYOUTCOMMIT, LAYOUTRETURN, LOCK, LOCKU, OPEN, OPEN_DOWNGRADE, READ, SETATTR, WRITE
NFS4ERR_FBIG	LAYOUTCOMMIT, OPEN, SETATTR, WRITE
NFS4ERR_FHEXPIRED	ACCESS, CLOSE, COMMIT, CREATE, DELEGRETURN, GETATTR, GETDEVICELIST, GETFH, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_FILE_OPEN	LINK, REMOVE, RENAME
NFS4ERR_GRACE	GETATTR, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, NVERIFY, OPEN, READ, REMOVE, RENAME, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_HASH_ALG_UNSUPP	EXCHANGE_ID
NFS4ERR_INVAL	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_PUSH_DELEG, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CREATE, CREATE_SESSION, DELEGRETURN, EXCHANGE_ID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, NVERIFY, OPEN, OPEN_DOWNGRADE, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, SECINFO, SECINFO_NO_NAME, SETATTR, SET_SSV, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_IO	ACCESS, COMMIT, CREATE, GETATTR, GETDEVICELIST,

	GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LINK, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, READ, READDIR, READLINK, REMOVE, RENAME, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_ISDIR	COMMIT, LAYOUTCOMMIT, LAYOUTRETURN, LINK, LOCK, LOCKT, OPEN, READ, WRITE
NFS4ERR_ LAYOUTTRYLATER	LAYOUTGET
NFS4ERR_ LAYOUTUNAVAILABLE	LAYOUTGET
NFS4ERR_LOCKED	LAYOUTGET, READ, SETATTR, WRITE
NFS4ERR_LOCKS_HELD	CLOSE, FREE_STATEID
NFS4ERR_LOCK_NOTSUPP	LOCK
NFS4ERR_LOCK_RANGE	LOCK, LOCKT, LOCKU
NFS4ERR_MLINK	CREATE, LINK, RENAME
NFS4ERR_MOVED	ACCESS, CLOSE, COMMIT, CREATE, DELEGRETURN, GETATTR, GETFH, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_NAMETOOLONG	CREATE, LINK, LOOKUP, OPEN, REMOVE, RENAME, SECINFO
NFS4ERR_NOENT	BACKCHANNEL_CTL, CREATE_SESSION, EXCHANGE_ID, GETDEVICEINFO, LOOKUP, LOOKUPP, OPEN, OPENATTR, REMOVE, RENAME, SECINFO, SECINFO_NO_NAME
NFS4ERR_NOFILEHANDLE	ACCESS, CLOSE, COMMIT, CREATE, DELEGRETURN, GETATTR, GETDEVICELIST, GETFH, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_NOMATCHING_ LAYOUT	CB_LAYOUTRECALL
NFS4ERR_NOSPC	CREATE, CREATE_SESSION, LAYOUTGET, LINK, OPEN, OPENATTR, RENAME, SETATTR, WRITE
NFS4ERR_NOTDIR	CREATE, GET_DIR_DELEGATION, LINK, LOOKUP, LOOKUPP, OPEN, READDIR, REMOVE, RENAME, SECINFO, SECINFO_NO_NAME
NFS4ERR_NOTEMPTY	REMOVE, RENAME
NFS4ERR_NOTSUPP	CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALLABLE_OBJ_AVAIL, CB_WANTS_CANCELLED, DELEGPURGE, DELEGRETURN, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, OPENATTR, OPEN_CONFIRM, RELEASE_LOCKOWNER, RENEW, SECINFO_NO_NAME, SETCLIENTID, SETCLIENTID_CONFIRM, WANT_DELEGATION
NFS4ERR_NOT_ONLY_OP	BIND_CONN_TO_SESSION, CREATE_SESSION, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID
NFS4ERR_NOT_SAME	EXCHANGE_ID, GETDEVICELIST, READDIR, VERIFY
NFS4ERR_NO_GRACE	LAYOUTCOMMIT, LAYOUTRETURN, LOCK, OPEN, WANT_DELEGATION

NFS4ERR_OLD_STATEID	CLOSE, DELEGRETURN, FREE_STATEID, LAYOUTGET, LAYOUTRETURN, LOCK, LOCKU, OPEN, OPEN_DOWNGRADE, READ, SETATTR, WRITE
NFS4ERR_OPENMODE	LAYOUTGET, LOCK, READ, SETATTR, WRITE
NFS4ERR_OP_ILLEGAL	CB_ILLEGAL, ILLEGAL
NFS4ERR_OP_NOT_IN_SESSION	ACCESS, BACKCHANNEL_CTL, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, DELEGPURGE, DELEGRETURN, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GETFH, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_PERM	CREATE, OPEN, SETATTR
NFS4ERR_PNFS_IO_HOLE	READ, WRITE
NFS4ERR_PNFS_NO_LAYOUT	READ, WRITE
NFS4ERR_RECALLCONFLICT	LAYOUTGET, WANT_DELEGATION
NFS4ERR_RECLAIM_BAD	LAYOUTCOMMIT, LOCK, OPEN, WANT_DELEGATION
NFS4ERR_RECLAIM_CONFLICT	LAYOUTCOMMIT, LOCK, OPEN, WANT_DELEGATION
NFS4ERR_REJECT_DELEG	CB_PUSH_DELEG
NFS4ERR_REP_TOO_BIG	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_REP_TOO_BIG_TO_CACHE	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_REQ_TOO_BIG	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR,

	CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_RETRY_UNCACHED_REP	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_ROFS	CREATE, LINK, LOCK, LOCKT, OPEN, OPENATTR, OPEN_DOWNGRADE, REMOVE, RENAME, SETATTR, WRITE
NFS4ERR_SAME	NVERIFY
NFS4ERR_SEQUENCE_POS	CB_SEQUENCE, SEQUENCE
NFS4ERR_SEQ_FAUX_RETRY	CB_SEQUENCE, SEQUENCE
NFS4ERR_SEQ_MISORDERED	CB_SEQUENCE, CREATE_SESSION, SEQUENCE
NFS4ERR_SERVERFAULT	ACCESS, BIND_CONN_TO_SESSION, CB_GETATTR, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SETATTR, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_SHARE_DENIED	OPEN
NFS4ERR_STALE	ACCESS, CLOSE, COMMIT, CREATE, DELEGRETURN, GETATTR, GETFH, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_STALE_CLIENTID	CREATE_SESSION, DESTROY_CLIENTID, DESTROY_SESSION
NFS4ERR_SYMLINK	COMMIT, LAYOUTCOMMIT, LINK, LOCK, LOCKT, LOOKUP, LOOKUPP,

	OPEN, READ, WRITE
NFS4ERR_TOOSMALL	CREATE_SESSION, GETDEVICEINFO, LAYOUTGET, READDIR
NFS4ERR_TOO_MANY_OPS	ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CB_GETATTR, CB_LAYOUTRECALL, CB_NOTIFY, CB_NOTIFY_DEVICEID, CB_NOTIFY_LOCK, CB_PUSH_DELEG, CB_RECALL, CB_RECALLABLE_OBJ_AVAIL, CB_RECALL_ANY, CB_RECALL_SLOT, CB_SEQUENCE, CB_WANTS_CANCELLED, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH, READ, READDIR, READLINK, RECLAIM_COMPLETE, REMOVE, RENAME, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE, SETATTR, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_UNKNOWN_ LAYOUTTYPE	CB_LAYOUTRECALL, GETDEVICEINFO, GETDEVICELIST, LAYOUTCOMMIT, LAYOUTGET, LAYOUTRETURN, NVERIFY, SETATTR, VERIFY
NFS4ERR_UNSAFE_ COMPOUND	CREATE, OPEN, OPENATTR
NFS4ERR_WRONGSEC	LINK, LOOKUP, LOOKUPP, OPEN, PUTFH, PUTPUBFH, PUTROOTFH, RENAME, RESTOREFH
NFS4ERR_WRONG_CRED	CLOSE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID, DESTROY_SESSION, FREE_STATEID, LAYOUTCOMMIT, LAYOUTRETURN, LOCK, LOCKT, LOCKU, OPEN_DOWNGRADE, RECLAIM_COMPLETE
NFS4ERR_WRONG_TYPE	CB_LAYOUTRECALL, CB_PUSH_DELEG, COMMIT, GETATTR, LAYOUTGET, LAYOUTRETURN, LINK, LOCK, LOCKT, NVERIFY, OPEN, OPENATTR, READ, READLINK, RECLAIM_COMPLETE, SETATTR, VERIFY, WANT_DELEGATION, WRITE
NFS4ERR_XDEV	LINK, RENAME

Tableau 14 : Erreurs et les opérations qui les utilisent

16. Procédures de NFSv4.1

Les deux procédures NULL et COMPOUND DOIVENT être mises en œuvre.

16.1 Procédure 0 : NULL - pas d'opération

16.1.1 Arguments

Vide

16.1.2 Résultats

Vide

16.1.3 Description

C'est la procédure standard NULL avec l'argument standard vide et la réponse vide. Cette procédure n'a pas de fonction associée. À cause de cela, elle est parfois utilisée pour mesurer les frais généraux de traitement d'une demande de service. Donc, le serveur DEVRAIT s'assurer qu'aucun travail inutile n'est fait en traitant cette procédure.

16.1.4 Erreurs

Aucune.

16.2 Procédure 1 : COMPOUND - opérations Compound

16.2.1 Arguments

```
enum nfs_opnum4 {
 OP_ACCESS               = 3,
 OP_CLOSE                = 4,
 OP_COMMIT               = 5,
 OP_CREATE               = 6,
 OP_DELEGPURGE           = 7,
 OP_DELEGRETURN          = 8,
 OP_GETATTR              = 9,
 OP_GETFH                = 10,
 OP_LINK                 = 11,
 OP_LOCK                 = 12,
 OP_LOCKT                = 13,
 OP_LOCKU                = 14,
 OP_LOOKUP               = 15,
 OP_LOOKUPP              = 16,
 OP_NVERIFY              = 17,
 OP_OPEN                 = 18,
 OP_OPENATTR             = 19,
 OP_OPEN_CONFIRM    = 20,                    /* Obligatoire de ne pas mettre en œuvre */
 OP_OPEN_DOWNGRADE = 21,
 OP_PUTFH                = 22,
 OP_PUTPUBFH             = 23,
 OP_PUTROOTFH            = 24,
 OP_READ                 = 25,
 OP_READDIR              = 26,
 OP_READLINK             = 27,
 OP_REMOVE               = 28,
 OP_RENAME               = 29,
 OP_RENEW                = 30,                    /* Obligatoire de ne pas mettre en œuvre */
 OP_RESTOREFH            = 31,
 OP_SAVEFH               = 32,
 OP_SECINFO              = 33,
 OP_SETATTR              = 34,
 OP_SETCLIENTID          = 35,                    /* Obligatoire de ne pas mettre en œuvre */
 OP_SETCLIENTID_CONFIRM = 36,                     /* Obligatoire de ne pas mettre en œuvre */
 OP_VERIFY               = 37,
 OP_WRITE                = 38,
 OP_RELEASE_LOCKOWNER   = 39,                     /* Obligatoire de ne pas mettre en œuvre */

/* nouvelles opérations pour NFSv4.1 */

 OP_BACKCHANNEL_CTL      = 40,
 OP_BIND_CONN_TO_SESSION = 41,
 OP_EXCHANGE_ID          = 42,
 OP_CREATE_SESSION       = 43,
 OP_DESTROY_SESSION      = 44,
 OP_FREE_STATEID         = 45,
 OP_GET_DIR_DELEGATION   = 46,
 OP_GETDEVICEINFO        = 47,
 OP_GETDEVICELIST        = 48,
```

```
 OP_LAYOUTCOMMIT             = 49,
 OP_LAYOUTGET                = 50,
 OP_LAYOUTRETURN             = 51,
 OP_SECINFO_NO_NAME          = 52,
 OP_SEQUENCE                 = 53,
 OP_SET_SSV                  = 54,
 OP_TEST_STATEID             = 55,
 OP_WANT_DELEGATION          = 56,
 OP_DESTROY_CLIENTID         = 57,
 OP_RECLAIM_COMPLETE         = 58,
 OP_ILLEGAL                  = 10044
};

union nfs_argop4 switch (nfs_opnum4 argop) {
cas OP_ACCESS :       ACCESS4args opaccess;
cas OP_CLOSE :        CLOSE4args opclose;
cas OP_COMMIT :       COMMIT4args opcommit;
cas OP_CREATE :       CREATE4args opcreate;
cas OP_DELEGPURGE :   DELEGPURGE4args opdelegpurge;
cas OP_DELEGRETURN :  DELEGRETURN4args opdelegreturn;
cas OP_GETATTR :      GETATTR4args opgetattr;
cas OP_GETFH :        Vide
cas OP_LINK :         LINK4args oplink;
cas OP_LOCK :         LOCK4args oplock;
cas OP_LOCKT :        LOCKT4args oplockt;
cas OP_LOCKU :        LOCKU4args oplocku;
cas OP_LOOKUP :       LOOKUP4args oplookup;
cas OP_LOOKUPP :      Vide
cas OP_NVERIFY :      NVERIFY4args opnverify;
cas OP_OPEN :         OPEN4args opopen;
cas OP_OPENATTR :     OPENATTR4args opopenattr;

/* Pas pour NFSv4.1 */
cas OP_OPEN_CONFIRM :  OPEN_CONFIRM4args opopen_confirm;
cas OP_OPEN_DOWNGRADE : OPEN_DOWNGRADE4args opopen_downgrade;
cas OP_PUTFH:         PUTFH4args opputfh;
cas OP_PUTPUBFH:      Vide
cas OP_PUTROOTFH:     Vide
cas OP_READ:          READ4args opread;
cas OP_READDIR:       READDIR4args opreaddir;
cas OP_READLINK:      Vide
cas OP_REMOVE:        REMOVE4args opremove;
cas OP_RENAME:        RENAME4args oprename;

/* Pas pour NFSv4.1 */
cas OP_RENEW:         RENEW4args oprenew;
cas OP_RESTOREFH:     Vide
cas OP_SAVEFH:        Vide
cas OP_SECINFO:       SECINFO4args opsecinfo;
cas OP_SETATTR:       SETATTR4args opsetattr;

/* Pas pour NFSv4.1 */
cas OP_SETCLIENTID: SETCLIENTID4args opsetclientid;

/* Pas pour NFSv4.1 */
cas OP_SETCLIENTID_CONFIRM: SETCLIENTID_CONFIRM4args opsetclientid_confirm;
cas OP_VERIFY:        VERIFY4args opverify;
cas OP_WRITE:         WRITE4args opwrite;

/* Pas pour NFSv4.1 */
cas OP_RELEASE_LOCKOWNER: RELEASE_LOCKOWNER4args oprelease_lockowner;
```

```
/* Opérations nouvelles pour NFSv4.1 */
cas OP_BACKCHANNEL_CTL: BACKCHANNEL_CTL4args opbackchannel_ctl;
cas OP_BIND_CONN_TO_SESSION: BIND_CONN_TO_SESSION4args opbind_conn_to_session;
cas OP_EXCHANGE_ID:   EXCHANGE_ID4args opexchange_id;
cas OP_CREATE_SESSION: CREATE_SESSION4args opcreate_session;
cas OP_DESTROY_SESSION: DESTROY_SESSION4args opdestroy_session;
cas OP_FREE_STATEID:  FREE_STATEID4args opfree_stateid;
cas OP_GET_DIR_DELEGATION: GET_DIR_DELEGATION4args opget_dir_delegation;
cas OP_GETDEVICEINFO: GETDEVICEINFO4args opgetdeviceinfo;
cas OP_GETDEVICELIST: GETDEVICELIST4args opgetdevicelist;
cas OP_LAYOUTCOMMIT:  LAYOUTCOMMIT4args oplayoutcommit;
cas OP_LAYOUTGET:     LAYOUTGET4args oplayoutget;
cas OP_LAYOUTRETURN:  LAYOUTRETURN4args oplayoutreturn;
cas OP_SECINFO_NO_NAME: SECINFO_NO_NAME4args opsecinfo_no_name;
cas OP_SEQUENCE :     SEQUENCE4args opsequence;
cas OP_SET_SSV :      SET_SSV4args opset_ssv;
cas OP_TEST_STATEID :  TEST_STATEID4args optest_stateid;
cas OP_WANT_DELEGATION : WANT_DELEGATION4args opwant_delegation;
cas OP_DESTROY_CLIENTID : DESTROY_CLIENTID4args opdestroy_clientid;
cas OP_RECLAIM_COMPLETE : RECLAIM_COMPLETE4args opreclaim_complete;

/* Opérations non nouvelles pour NFSv4.1 */
cas OP_ILLEGAL :      Vide
};

struct COMPOUND4args {
        utf8str_cs     tag;
        uint32_t       minorversion;
        nfs_argop4     argarray<>;
};
```

16.2.2 Résultats

```
union nfs_resop4 switch (nfs_opnum4 resop) {
cae OP_ACCESS:        ACCESS4res opaccess;
cas OP_CLOSE:         CLOSE4res opclose;
cas OP_COMMIT:        COMMIT4res opcommit;
cas OP_CREATE:        CREATE4res opcreate;
cas OP_DELEGPURGE:    DELEGPURGE4res opdelegpurge;
cas OP_DELEGRETURN:   DELEGRETURN4res opdelegreturn;
cas OP_GETATTR:       GETATTR4res opgetattr;
cas OP_GETFH:        GETFH4res opgetfh;
cas OP_LINK:         LINK4res oplink;
cas OP_LOCK:          LOCK4res oplock;
cas OP_LOCKT:        LOCKT4res oplockt;
cas OP_LOCKU:        LOCKU4res oplocku;
cas OP_LOOKUP:       LOOKUP4res oplookup;
cas OP_LOOKUPP:       LOOKUPP4res oplookupp;
cas OP_NVERIFY:       NVERIFY4res opnverify;
cas OP_OPEN:           OPEN4res opopen;
cas OP_OPENATTR:      OPENATTR4res opopenattr;

/* Pas pour NFSv4.1 */
cas OP_OPEN_CONFIRM:       OPEN_CONFIRM4res opopen_confirm;
cas OP_OPEN_DOWNGRADE: OPEN_DOWNGRADE4res opopen_downgrade;
cas OP_PUTFH:                 PUTFH4res opputfh;
cas OP_PUTPUBFH:             PUTPUBFH4res opputpubfh;
cas OP_PUTROOTFH:           PUTROOTFH4res opputrootfh;
cas OP_READ:                  READ4res opread;
```

```
cas OP_READDIR:        READDIR4res opreaddir;
cas OP_READLINK:       READLINK4res opreadlink;
cas OP_REMOVE:         REMOVE4res opremove;
cas OP_RENAME:         RENAME4res oprename;

/* Pas pour NFSv4.1 */
cas OP_RENEW:           RENEW4res oprenew;
cas OP_RESTOREFH:      RESTOREFH4res oprestorefh;
cas OP_SAVEFH:         SAVEFH4res opsavefh;
cas OP_SECINFO:        SECINFO4res opsecinfo;
cas OP_SETATTR:        SETATTR4res opsetattr;

/* Pas pour NFSv4.1 */
cas OP_SETCLIENTID: SETCLIENTID4res opsetclientid;

/* Pas pour NFSv4.1 */
cas OP_SETCLIENTID_CONFIRM : SETCLIENTID_CONFIRM4res opsetclientid_confirm;
cas OP_VERIFY :       VERIFY4res opverify;
cas OP_WRITE :        WRITE4res opwrite;

/* Pas pour NFSv4.1 */
cas OP_RELEASE_LOCKOWNER : RELEASE_LOCKOWNER4res oprelease_lockowner;

/* Les opérations nouvelles pour NFSv4.1 */
cas OP_BACKCHANNEL_CTL:        BACKCHANNEL_CTL4res opbackchannel_ctl;
cas OP_BIND_CONN_TO_SESSION: BIND_CONN_TO_SESSION4res opbind_conn_to_session;
cas OP_EXCHANGE_ID:             EXCHANGE_ID4res opexchange_id;
cas OP_CREATE_SESSION:          CREATE_SESSION4res opcreate_session;
cas OP_DESTROY_SESSION:         DESTROY_SESSION4res opdestroy_session;
cas OP_FREE_STATEID:            FREE_STATEID4res opfree_stateid;
cas OP_GET_DIR_DELEGATION:    GET_DIR_DELEGATION4res opget_dir_delegation;
cas OP_GETDEVICEINFO:           GETDEVICEINFO4res opgetdeviceinfo;
cas OP_GETDEVICELIST:           GETDEVICELIST4res opgetdevicelist;
cas OP_LAYOUTCOMMIT:            LAYOUTCOMMIT4res oplayoutcommit;
cas OP_LAYOUTGET:               LAYOUTGET4res oplayoutget;
cas OP_LAYOUTRETURN:            LAYOUTRETURN4res oplayoutreturn;
cas OP_SECINFO_NO_NAME:         SECINFO_NO_NAME4res opsecinfo_no_name;
cas OP_SEQUENCE:                SEQUENCE4res opsequence;
cas OP_SET_SSV:                 SET_SSV4res opset_ssv;
cas OP_TEST_STATEID:            TEST_STATEID4res optest_stateid;
cas OP_WANT_DELEGATION:       WANT_DELEGATION4res opwant_delegation;
cas OP_DESTROY_CLIENTID:        DESTROY_CLIENTID4res opdestroy_clientid;
cas OP_RECLAIM_COMPLETE:        RECLAIM_COMPLETE4res opreclaim_complete;

/* Opérations non nouvelles pour NFSv4.1 */
cas OP_ILLEGAL :      ILLEGAL4res opillegal;
};

struct COMPOUND4res {
      nfsstat4        status;
      utf8str_cs      tag;
      nfs_resop4      resarray<>;
};
```

16.2.3 Description

La procédure COMPOUND est utilisée pour combiner une ou plusieurs opérations NFSv4 en une seule demande RPC. Le serveur interprète chaque opération à son tour. Si une opération est exécutée par le serveur et si l'état de cette opération est NFS4_OK, alors la prochaine opération dans la procédure COMPOUND est exécutée. Le serveur continue ce processus jusqu'à ce qu'il n'y ait plus d'opération à exécuter ou qu'une des opérations ait une valeur d'état autre que NFS4_OK.

Dans le traitement de la procédure COMPOUND, le serveur peut trouver qu'il n'a pas les ressources disponibles pour exécuter tout ou partie des opérations dans la séquence COMPOUND. Voir au paragraphe 2.10.6.4 une discussion plus détaillée.

Le serveur va généralement choisir entre deux méthodes de décodage de la demande du client. La première serait le décodage XDR traditionnel en une fois. Si il y a une erreur de décodage XDR dans ce cas, l'erreur de décodage XDR RPC va être retournée. La seconde méthode serait de faire un passage initial pour décoder la demande COMPOUND de base et ensuite de faire le décodage XDR des opérations individuelles ; le plus intéressant est le décodage des attributs. Dans ce cas, le serveur peut rencontrer une erreur de décodage XDR durant le second passage. Si c'est le cas, le serveur va retourner l'erreur NFS4ERR_BADXDR pour signifier l'erreur de décodage.

Les arguments COMPOUND contiennent un champ "minorversion". Pour NFSv4.1, la valeur de ce champ est 1. Si le serveur reçoit une procédure COMPOUND avec un champ minorversion d'une valeur qu'il ne prend pas en charge, le serveur DOIT retourner une erreur de NFS4ERR_MINOR_VERS_MISMATCH et un dispositif resultdata de longueur zéro. Un champ "status" est contenu dans le résultat de COMPOUND. Si la longueur du dispositif de résultat n'est pas zéro, cet état doit être équivalent à l'état de la dernière opération exécutée dans la procédure COMPOUND. Donc, si une opération a subi une erreur alors la valeur "status" va être la même valeur d'erreur que retournée pour l'opération qui a échoué.

Noter que les opérations zéro et un ne sont pas définies pour la procédure COMPOUND. L'opération 2 n'est pas définie et est réservée pour une future définition et utilisation avec la version mineure. Si le serveur reçoit un dispositif d'opération qui contient l'opération 2 et si le champ minorversion a une valeur de zéro, une erreur de NFS4ERR_OP_ILLEGAL, comme décrit au prochain paragraphe, est retournée au client. Si un dispositif d'opération contient une opération 2 et si le champ minorversion n'est pas zéro et si le serveur ne prend pas en charge la version mineure, le serveur retourne une erreur de NFS4ERR_MINOR_VERS_MISMATCH. Donc, l'erreur NFS4ERR_MINOR_VERS_MISMATCH prend le pas sur toutes les autres erreurs.

Il est possible que le serveur reçoive une demande qui contient une opération inférieure à la première opération légale (OP_ACCESS) ou supérieure à la dernière opération légale (OP_RELEASE_LOCKOWNER). Dans ce cas, la réponse du serveur va coder le opcode OP_ILLEGAL plutôt que le opcode illégal de la demande. Le champ Status dans le résultat de retour ILLEGAL va être réglé à NFS4ERR_OP_ILLEGAL. Le résultat de retour de la procédure COMPOUND va aussi être NFS4ERR_OP_ILLEGAL.

La définition de "tag" dans la demande est laissé à la mise en œuvre. Elle peut être utilisée pour résumer le contenu de la demande Compound pour le bénéfice des renifleurs de paquets et des ingénieurs qui déboguent les mises en œuvre. Cependant, la valeur de "tag" dans la réponse DEVRAIT être la même valeur que fournie dans la demande. Cela s'applique aussi au champ Tag de la procédure CB_COMPOUND.

16.2.3.1 Bride de fichier et identifiant d'état courants

La procédure COMPOUND offre un simple environnement pour l'exécution des opérations spécifiées par le client. Les deux premières se rapportent à la bride de fichier tandis que les deux secondes se rapportent à l'identifiant d'état courant

16.2.3.1.1 Bride de fichier courante

Les brides de fichier courante et sauvegardée sont utilisées dans tout le protocole. La plupart des opérations utilisent implicitement la bride de fichier courante comme argument, et beaucoup règlent la bride de fichier courante au titre du résultat. La combinaison des séquences d'opérations spécifiées par le client et des arguments et résultats de bride de fichier courante et sauvegardée permet une plus grande souplesse du protocole. Le meilleur ou plus facile exemple d'usage de bride de fichier courante est une séquence comme celle qui suit :

```
PUTFH fh1           {fh1}
LOOKUP "compA"      {fh2}
GETATTR             {fh2}
LOOKUP "compB"      {fh3}
GETATTR             {fh3}
LOOKUP "compC"      {fh4}
GETATTR             {fh4}
GETFH
```

Figure 2

Dans cet exemple, l'opération PUTFH (paragraphe 18.19) règle explicitement la valeur courante de bride de fichier tandis que le résultat de chaque opération LOOKUP règle la valeur courante de bride de fichier à l'objet résultant de système de fichiers. Aussi, le client est capable d'insérer des opérations GETATTR en utilisant la bride de fichier courante comme argument.

Les opérations PUTROOTFH (paragraphe 18.21) et PUTPUBFH (paragraphe 18.20) règlent aussi la bride de fichier courante. L'exemple ci-dessus remplace "PUTFH fh1" avec PUTROOTFH ou PUTPUBFH par un argument non de bride de fichier afin de réaliser le même effet (dans l'hypothèse où "compA" est directement en dessous de la racine de l'espace de noms).

Avec la bride de fichier courante, il y a une bride de fichier sauvegardée. Alors que la bride de fichier courante est établie comme résultat d'opérations comme LOOKUP, la bride de fichier sauvegardée doit être établie directement avec l'utilisation de l'opération SAVEFH. L'opération SAVEFH copie la valeur courante de bride de fichier sur la valeur sauvegardée. La valeur sauvegardée de bride de fichier est utilisée en combinaison avec la valeur courante de bride de fichier pour les opérations LINK et RENAME. L'opération RESTOREFH va copier la valeur sauvegardée de bride de fichier à la valeur courante de bride de fichier ; par suite, la valeur sauvegardée de bride de fichier peut être utilisée comme une sorte de zone "zéro" pour la série des opérations du client.

16.2.3.1.2 Identifiant d'état courant

Avec NFSv4.1, l'ajout d'un identifiant d'état courant et d'un identifiant d'état sauvegardé a été fait à l'environnement de traitement COMPOUND ; cela permet de passer les identifiants d'état entre les opérations. Il n'y a pas de changement à la syntaxe du protocole, seulement des changements à la signification de quelques opérations.

Un "identifiant d'état courant" est l'identifiant d'état qui est associé à la bride de fichier courante. L'identifiant d'état courant peut seulement être changé par une opération qui modifie la bride de fichier courante ou retourne un identifiant d'état. Si une opération retourne un identifiant d'état, elle DOIT régler l'identifiant d'état courant à la valeur retournée. Si une opération règle la bride de fichier courante mais ne retourne pas d'identifiant d'état, l'identifiant d'état courant DOIT être réglé à l'identifiant d'état spécial tout de zéros, c'est-à-dire, (seqid, autre) = (0, 0). Si une opération utilise un identifiant d'état comme argument mais ne retourne pas un identifiant d'état, l'identifiant d'état courant NE DOIT PAS être changé. Par exemple, PUTFH, PUTROOTFH, et PUTPUBFH vont changer l'état du serveur courant de {ocfh, (osid)} à {cfh, (0, 0)}, tandis que LOCK va changer l'état courant de {cfh, (osid} à {cfh, (nsid)}. Les opérations comme LOOKUP qui transforment une bride de fichier courante et un nom de composant en une nouvelle bride de fichier courante vont aussi changer l'état courant en {0, 0}. Les opérations SAVEFH et RESTOREFH vont sauvegarder et restaurer la bride de fichier et l'identifiant d'état courants comme un ensemble.

L'exemple suivant est le cas courant d'une simple opération READ avec un identifiant d'état normal montrant que PUTFH initialise l'identifiant d'état courant à (0, 0). Le READ suivant avec l'identifiant d'état (sid1) laisse inchangé l'identifiant d'état courant.

```
PUTFH fh1                  - -> {fh1, (0, 0)}
READ (sid1), 0, 1024   {fh1, (0, 0)} -> {fh1, (0, 0)}
```

Figure 3

L'exemple suivant effectue un OPEN avec la bride de fichier racine et, par suite, génère un identifiant d'état (sid1). L'opération suivante spécifie le READ avec l'identifiant d'état réglé de telle façon que (seqid, autre) soit égal à (1, 0) mais l'identifiant d'état courant réglé par l'opération précédente est actuellement utilisé quand l'opération est évaluée. Cela permet une interaction correcte avec tous verrous existants, potentiellement en conflit.

```
PUTROOTFH                  - -> {fh1, (0, 0)}
OPEN "compA"           {fh1, (0, 0)} -> {fh2, (sid1)}
READ (1, 0), 0, 1024   {fh2, (sid1)} -> {fh2, (sid1)}
CLOSE (1, 0)           {fh2, (sid1)} -> {fh2, (sid2)}
```

Figure 4

L'exemple suivant est similaire au second en ce qu'il passe l'identifiant d'état sid2 généré par l'opération LOCK à la prochaine opération READ. Cela permet au client d'entourer explicitement une seule opération d'entrée/sortie avec un verrou et son identifiant d'état approprié pour garantir qu'il est correct à l'égard des verrous des autres clients. L'exemple

montre aussi comment SAVEFH et RESTOREFH peuvent sauvegarder et plus tard réutiliser une bride de fichier et un identifiant d'état, les passant comme bride de fichier et identifiant d'état courants à une opération READ.

```
PUTFH fh1                   - -> {fh1, (0, 0)}
LOCK 0, 1024, (sid1)      {fh1, (sid1)} -> {fh1, (sid2)}
READ (1, 0), 0, 1024      {fh1, (sid2)} -> {fh1, (sid2)}
LOCKU 0, 1024, (1, 0)     {fh1, (sid2)} -> {fh1, (sid3)}
SAVEFH                    {fh1, (sid3)} -> {fh1, (sid3)}

PUTFH fh2                 {fh1, (sid3)} -> {fh2, (0, 0)}
WRITE (1, 0), 0, 1024     {fh2, (0, 0)} -> {fh2, (0, 0)}

RESTOREFH                 {fh2, (0, 0)} -> {fh1, (sid3)}
READ (1, 0), 1024, 1024   {fh1, (sid3)} -> {fh1, (sid3)}
```

Figure 5

L'exemple final montre une utilisation interdite de l'identifiant d'état courant. Le client tente de passer implicitement un identifiant d'état anonyme spécial, (0,0), à l'opération READ. Le serveur DOIT retourner NFS4ERR_BAD_STATEID dans la réponse à l'opération READ.

```
PUTFH fh1                   - -> {fh1, (0, 0)}
READ (1, 0), 0, 1024      {fh1, (0, 0)} -> NFS4ERR_BAD_STATEID
```

Figure 6

16.2.4 Erreurs

COMPOUND va bien sûr retourner chaque erreur que chaque opération sur le canal avant peut retourner (voir Tableau 12). Cependant, si COMPOUND retourne zéro opération, évidemment l'erreur retournée par COMPOUND n'a rien à voir avec une erreur retournée par une opération. La liste des erreurs que COMPOUND va retourner si il traite zéro opération inclut :

Erreur	Notes
NFS4ERR_BADCHAR	L'argument tag a un caractère que le répondeur ne prend pas en charge.
NFS4ERR_BADXDR	
NFS4ERR_DELAY	
NFS4ERR_INVAL	L'argument tag n'est pas en codage UTF-8.
NFS4ERR_MINOR_VERS_MISMATCH	
NFS4ERR_SERVERFAULT	
NFS4ERR_TOO_MANY_OPS	
NFS4ERR_REP_TOO_BIG	
NFS4ERR_REP_TOO_BIG_TO_CACHE	
NFS4ERR_REQ_TOO_BIG	

Tableau 15 : Retours d'erreur sur COMPOUND

17. Opérations EXIGÉES, RECOMMANDÉES, ou FACULTATIVES

Les tableaux qui suivent résument les opérations du protocole NFSv4.1 et la désignation correspondante de EXIGÉ, RECOMMANDÉ, et FACULTATIF pour mettre en œuvre ou NE DOIT PAS mettre en œuvre. La désignation de NE DOIT PAS mettre en œuvre est réservée aux opérations qui ont été définies dans NFSv4.0 et NE DOIVENT PAS être mises en œuvre dans NFSv4.1.

Pour la plus grande part, la désignation EXIGÉ, RECOMMANDÉ, ou FACULTATIF pour les opérations envoyées par le client est pour la mise en œuvre de serveur. Le client est généralement obligé de mettre en œuvre les opérations nécessaires pour l'environnement de fonctionnement qu'il dessert. Par exemple, un client NFSv4.1 en lecture seule ne va pas avoir besoin de mettre en œuvre l'opération WRITE et n'est pas obligé de le faire.

La désignation EXIGÉ ou FACULTATIF pour les opérations de rappel envoyées par le serveur est pour le client et le serveur. Généralement, le client a l'option de créer le canal de retour et d'envoyer les opérations sur le canal avant qui va être un catalyseur pour l'envoi par le serveur des opérations de rappel. Une exception partielle est CB_RECALL_SLOT ; la seule façon qu'a le client pour éviter de prendre en charge cette opération est de ne pas créer de canal de retour.

Comme c'est un résumé des opérations et de leur désignation, certaines subtilités ne sont pas présentées ici. Donc, si il y a une question sur les exigences de mise en œuvre, les descriptions d'opération elles mêmes doivent être consultées avec les autres textes explicatifs pertinents dans cette spécification.

Les abréviations utilisées dans les colonnes deux et trois du tableau sont définies comme suit :
REQ : de mise en œuvre EXIGÉE
REC : de mise en œuvre RECOMMANDÉE
OPT : de mise en œuvre FACULTATIVE
MNI : NE DOIT PAS être mis en œuvre

Pour les caractéristiques NFSv4.1 qui sont FACULTATIVES, les opérations qui prennent en charge ces caractéristiques sont FACULTATIVES, et le serveur va retourner NFS4ERR_NOTSUPP en réponse à l'utilisation par le client de ces opérations. Si une caractéristique FACULTATIVE est prise en charge, il est possible qu'un ensemble d'opérations relatives à la caractéristique devienne de mise en œuvre EXIGÉE. La troisième colonne du tableau désigne les caractéristiques et si l'opération est EXIGÉE ou FACULTATIVE en présence de la prise en charge de la caractéristique.

Les caractéristiques FACULTATIVES identifiées et leur abréviation sont les suivantes :
pNFS : NFS parallèle
FDELG : délégations de fichier
DDELG : délégations de répertoire

Opération	Exigence	Caractéristique	Paragraphe de définition
ACCESS	REQ		18.1
BACKCHANNEL_CTL	REQ		18.33
BIND_CONN_TO_SESSION	REQ		18.34
CLOSE	REQ		18.2
COMMIT	REQ		18.3
CREATE	REQ		18.4
CREATE_SESSION	REQ		18.36
DELEGPURGE	OPT	FDELG (REQ)	18.5
DELEGRETURN	OPT	FDELG, DDELG, pNFS (REQ)	18.6
DESTROY_CLIENTID	REQ		18.50
DESTROY_SESSION	REQ		18.37
EXCHANGE_ID	REQ		18.35
FREE_STATEID	REQ		18.38
GETATTR	REQ		18.7
GETDEVICEINFO	OPT	pNFS (REQ)	18.40
GETDEVICELIST	OPT	pNFS (OPT)	18.41
GETFH	REQ		18.8
GET_DIR_DELEGATION	OPT	DDELG (REQ)	18.39
LAYOUTCOMMIT	OPT	pNFS (REQ)	18.42
LAYOUTGET	OPT	pNFS (REQ)	18.43
LAYOUTRETURN	OPT	pNFS (REQ)	18.44
LINK	OPT		18.9
LOCK	REQ		18.10
LOCKT	REQ		18.11
LOCKU	REQ		18.12
LOOKUP	REQ		18.13
LOOKUPP	REQ		18.14
NVERIFY	REQ		18.15
OPEN	REQ		18.16
OPENATTR	OPT		18.17
OPEN_CONFIRM	MNI		N/A
OPEN_DOWNGRADE	REQ		18.18
PUTFH	REQ		18.19
PUTPUBFH	REQ		18.20

PUTROOTFH	REQ		18.21
READ	REQ		18.22
READDIR	REQ		18.23
READLINK	OPT		18.24
RECLAIM_COMPLETE	REQ		18.51
RELEASE_LOCKOWNER	MNI		N/A
REMOVE	REQ		18.25
RENAME	REQ		18.26
RENEW	MNI		N/A
RESTOREFH	REQ		18.27
SAVEFH	REQ		18.28
SECINFO	REQ		18.29
SECINFO_NO_NAME	REC	disposition fichier pNFS (REQ)	18.45, 13.12
SEQUENCE	REQ		18.46
SETATTR	REQ		18.30
SETCLIENTID	MNI		N/A
SETCLIENTID_CONFIRM	MNI		N/A
SET_SSV	REQ		18.47
TEST_STATEID	REQ		18.48
VERIFY	REQ		18.31
WANT_DELEGATION	OPT	FDELG (OPT)	18.49
WRITE	REQ		18.32

Tableau 16 : Opérations

Opération	Exigence	Caractéristiques	Paragraphe de définition
CB_GETATTR	OPT	FDELG (REQ)	20.1
CB_LAYOUTRECALL	OPT	pNFS (REQ)	20.3
CB_NOTIFY	OPT	DDELG (REQ)	20.4
CB_NOTIFY_DEVICEID	OPT	pNFS (OPT)	20.12
CB_NOTIFY_LOCK	OPT		20.11
CB_PUSH_DELEG	OPT	FDELG (OPT)	20.5
CB_RECALL	OPT	FDELG, DDELG, pNFS (REQ)	20.2
CB_RECALL_ANY	OPT	FDELG, DDELG, pNFS (REQ)	20.6
CB_RECALL_SLOT	REQ		20.8
CB_RECALLABLE_OBJ_AVAIL	OPT	DDELG, pNFS (REQ)	20.7
CB_SEQUENCE	OPT	FDELG, DDELG, pNFS (REQ)	20.9
CB_WANTS_CANCELLED	OPT	FDELG, DDELG, pNFS (REQ)	20.10

Tableau 17 : Opérations de rappel

18. Opérations NFSv4.1

18.1 Opération 3 : ACCESS - vérification des droits d'accès

18.1.1 Arguments

```
const ACCESS4_READ      = 0x00000001;
const ACCESS4_LOOKUP    = 0x00000002;
const ACCESS4_MODIFY    = 0x00000004;
const ACCESS4_EXTEND    = 0x00000008;
const ACCESS4_DELETE    = 0x00000010;
const ACCESS4_EXECUTE   = 0x00000020;

struct ACCESS4args {                          /* CURRENT_FH : objet */
      uint32_t      accès ;
};
```

18.1.2 Résultats

```
struct ACCESS4resok {
      uint32_t      pris en charge ;
      uint32_t      accès ;
};

union ACCESS4res switch (nfsstat4 status) {
   cas NFS4_OK :  ACCESS4resok   resok4;
   défaut :  vide
};
```

18.1.3 Description

ACCESS détermine les droits d'accès qu'un utilisateur, tel qu'identifié par les accréditifs dans la demande RPC, a à l'égard de l'objet de système de fichiers spécifié par la bride de fichier courante. Le client code l'ensemble des droits d'accès qui sont à vérifier dans le gabarit binaire "access". Le serveur vérifie les permissions codées dans le gabarit binaire. Si un état de NFS4_OK est retourné, deux gabarits binaires sont inclus dans la réponse. Le premier, "pris en charge", représente les droits d'accès pour lesquels le serveur peut faire une vérification fiable. Le second, "accès, représente les droits d'accès disponibles à l'utilisateur pour la bride de fichier fournie. En cas de succès, la bride de fichier courante conserve sa valeur.

Noter que les champs Prise en charge et Accès de la réponse NE DOIVENT PAS contenir plus de valeurs que réglé à l'origine dans le champ Accès de la demande. Par exemple, si le client envoie une opération ACCESS avec juste la valeur ACCESS4_READ établie et si le serveur prend en charge cette valeur, le serveur NE DOIT PAS établir plus que ACCESS4_READ dans le champ Pris en charge même si il pourrait avoir vérifié fiablement d'autres valeurs.

Le champ Accès de la réponse NE DOIT PAS contenir plus de valeurs que le champ Pris en charge.

Les résultats de cette opération sont nécessairement indicatifs par nature. Le retour d'un état de NFS4_OK et le bit approprié réglé dans le gabarit binaire n'impliquent pas qu'un tel accès va être permis à l'objet de système de fichiers à l'avenir. C'est parce que les droits d'accès peuvent être révoqués à tout moment par le serveur.

Les permissions d'accès suivantes peuvent être demandées :
ACCESS4_READ : lire les données d'un fichier ou d'un répertoire.
ACCESS4_LOOKUP : chercher un nom dans un répertoire (non significatif pour les objets non de répertoire).
ACCESS4_MODIFY : réécrire les données existantes d'un fichier ou modifier les entrées existantes d'un répertoire.
ACCESS4_EXTEND : écrire de nouvelles données ou ajouter des entrées de répertoire.
ACCESS4_DELETE : supprimer une entrée existante de répertoire.
ACCESS4_EXECUTE : exécute un fichier régulier (non significatif pour un répertoire).

En cas de succès, la bride de fichier courante conserve sa valeur.

ACCESS4_EXECUTE est un défi sémantique à mettre en œuvre parce que NFS fournit l'accès au fichier distant, pas l'exécution à distance. Cela conduit à ce qui suit :

* Si un fichier régulier est exécutable devrait être de la responsabilité du client NFS et non du serveur. Et cependant l'opération ACCESS est spécifiée comme exigeant d'un serveur qu'il possède cette responsabilité.
* Quand un client exécute un fichier régulier, il doit lire le fichier provenant du serveur. Strictement parlant, le serveur ne devrait pas permettre au client de lire un fichier en cours d'exécution à moins que l'utilisateur ait la permission de lecture sur le fichier. Exiger la permission explicite de lecture sur les fichiers exécutables afin d'y accéder sur NFS ne va pas être acceptable pour certains utilisateurs et administrateurs de mémorisation. Historiquement, les serveurs NFS ont permis à un utilisateur de lire un fichier si l'utilisateur a l'accès exécuter sur le fichier.

Pour un exemple pratique, la spécification UNIX [IEEE-1003.1] déclare qu'une mise en œuvre revendiquant la conformité à UNIX peut indiquer dans le résultat access() de l'interface de programmation qu'un utililateur privilégié a les droits d'exécution, même si les bits de permission de non exécution sont établis sur les attributs du fichier régulier. Il est possible de revendiquer la conformité à la spécification UNIX et de ne pas indiquer les droits d'exécution dans cette situation, qui est vraie pour certains environnements de fonctionnement. Supposons que les environnements de fonctionnement du client et du serveur mettent différemment en œuvre la sémantique de access() pour les utilisateurs privilégiés, et que les mises en œuvre de l'opération ACCESS du client et du serveur suivent leur sémantique respective de access(). Cela peut causer un comportement non désiré :

* Supposons que l'interface access() du client retourne X_OK si l'utilisateur est privilégié et si les bits de permission de non exécution sont établis sur l'attribut du fichier régulier, et que l'interface access() du serveur ne retourne pas X_OK dans cette situation. Le client va alors être incapable d'exécuter les fichiers mémorisés sur le serveur NFS qui pourraient être exécutés si ils étaient mémorisés sur un système de fichiers non NFS.

* Supposons que l'interface access() du client ne retourne pas X_OK si l'utilisateur est privilégié, et si les bits de permission de non exécution sont établis sur l'attribut du fichier régulier, et que l'interface access() du serveur retourne X_OK dans cette situation. Alors :
  - le client va être capable d'exécuter les fichiers mémorisés sur le serveur NFS qui pourraient être exécutés si ils étaient mémorisés sur un système de fichiers non NFS, sauf si le sous-système d'exécution du client vérifie aussi les bits de permission d'exécuter ;
  - même si le sous-système d'exécution vérifie les bits de permission d'exécuter, il y a d'autres problèmes potentiels. Par exemple, supposons que le client invoque access() pour construire un "tableau de recherche de chemin" de tous les fichiers exécutables dans le "chemin de recherche" de l'utilisateur, où le chemin est une liste de répertoires contenant chacun des fichiers exécutables. Supposons qu'il y ait deux fichiers, chacun dans des répertoires séparés du chemin de recherche, tels que ces fichiers aient le même nom de composant. Dans le premier répertoire le fichier a les bits de permission de non exécution établis, et dans le second répertoire le fichier a les bits d'exécution établis. Le tableau de recherche de chemin va indiquer que le premier répertoire a le fichier exécutable, mais le sous système d'exécution va échouer à l'exécuter. La coquille de commande pourrait échouer à essayer le second fichier dans le second répertoire. Et même si il avait réussi, il y a un problème potentiel de performances. En clair, le résultat désiré pour le client est que le tableau de recherche de chemin ne contienne pas le premier fichier.

Pour traiter les problèmes décrits ci-dessus, le principe du "client intelligent, serveur stupide" est utilisé. Le client possède la responsabilité globale de déterminer l'accès pour exécution et s'appuie sur le serveur pour analyser les permissions d'exécution dans les attributs mode, acl, et dacl du fichier. Les règles pour le client et le serveur suivent :

* Si le client envoie ACCESS pour déterminer si l'utilisateur peut lire le fichier, le client DEVRAIT établir ACCESS4_READ dans le champ accès de la demande.

* Si l'environnement de fonctionnement du client accorde seulement l'exécution à l'utilisateur si celui-ci a exécuté l'accès conformément à la permission d'exécuter dans les attributs mode, acl, et dacl, alors si le client veut déterminer l'accès d'exécution, le client DEVRAIT envoyer une demande ACCESS avec le bit ACCESS4_EXECUTE établi dans le champ Accès de la demande.

* Si l'environnement de fonctionnement du client accorde l'exécution à l'utilisateur même si l'utilisateur n'a pas exécuté l'accès conformément aux permissions d'exécuter dans les attributs mode, acl, et dacl, alors si le client veut déterminer l'accès d'exécution, il DEVRAIT envoyer une demande ACCESS avec les deux bits ACCESS4_EXECUTE et ACCESS4_READ établis dans le champ Accès de la demande. De cette façon, si une permission read ou execute accorde à l'utilisateur l'accès en lecture ou exécution (ou si le serveur interprète l'utilisateur comme étant privilégié) comme indiqué par la présence de ACCESS4_EXECUTE et/ou ACCESS4_READ dans le champ Accès de la réponse, le client va être capable d'accorder à l'utilisateur l'accès en exécution au fichier.

* Si le serveur prend en charge les bits de permission d'exécuter, ou une autre méthode pour noter la possibilité d'exécuter (par exemple, le suffixe du nom du fichier pourrait indiquer exécute) il DOIT vérifier seulement les permissions d'exécuter, pas les permissions de lire, quand il détermine si la réponse a ou non le bit ACCESS4_EXECUTE établi dans le champ Accès. Le serveur NE DOIT PAS aussi examiner les bits de permission de lecture quand il détermine si la réponse va avoir ou non le bit ACCESS4_EXECUTE établi dans le champ Accès. Même si l'environnement de fonctionnement du serveur accorderait l'accès en exécution à l'utilisateur (par exemple, l'utilisateur est privilégié) le serveur NE DOIT PAS répondre avec ACCESS4_EXECUTE établi dans le champ Accès de la réponse sauf si il y a au moins un bit de permission d'exécuter dans les attributs mode, acl, ou dacl. Dans le cas de acl et dacl, le "bit de permission d'exécuter" DOIT être un bit ACE4_EXECUTE établi dans un ACE ALLOW.

* Si le serveur ne prend pas en charge les bits de permission d'exécuter ou d'autre méthode pour noter l'exécutabilité, il NE DOIT PAS établir ACCESS4_EXECUTE dans les champs Pris en charge et Accès de la réponse. Si le client établit ACCESS4_EXECUTE dans le champ Accès de la demande ACCESS, et si ACCESS4_EXECUTE n'est pas établi dans le champ Pris en charge de la réponse, alors le client va devoir envoyer une demande ACCESS avec le bit ACCESS4_READ établi dans le champ Accès de la demande.

* Si le serveur prend en charge les bits de permission de lecture, il DOIT seulement vérifier les permissions d'écriture dans les attributs mode, acl, et dacl quand il reçoit une demande ACCESS avec ACCESS4_READ établi dans le champ Accès. Le serveur NE DOIT PAS aussi examiner les bits de permission d'exécution quand il détermine si la réponse va

avoir ACCESS4_READ établi ou non dans le champ Accès.

Noter que si la réponse ACCESS a ACCESS4_READ ou ACCESS_EXECUTE établi, alors l'utilisateur a aussi les permissions à OPEN (paragraphe 18.16) ou READ (paragraphe 18.22) sur le fichier. En d'autres termes, si le client envoie une demande ACCESS avec ACCESS4_READ et ACCESS_EXECUTE établis dans le champ Accès (ou deux demandes séparées, une avec ACCESS4_READ établi et l'autre avec ACCESS4_EXECUTE établi) et si la réponse a juste ACCESS4_EXECUTE établi dans le champ Accès (ou si juste une réponse a ACCESS4_EXECUTE établi) alors l'utilisateur a l'autorisation OPEN ou READ sur le fichier.

18.1.4 Mise en œuvre

En général, il n'est pas suffisant que le client tente de déduire les permissions d'accès de l'inspection des champs uid, gid, et mode dans les attributs du fichier ou tente d'interpréter le contenu de l'attribut ACL. C'est parce que le serveur peut effectuer une transposition de uid ou gid ou appliquer des restrictions supplémentaires de contrôle d'accès. Il est aussi possible que le serveur ne puisse pas être dans le même espace d'identifiant que le client. Dans ce cas (et peut-être d'autres) le client ne peut pas effectuer fiablement une vérification d'accès avec seulement les attributs de fichier courants.

Dans le protocole NFSv2, la seule façon fiable de déterminer si une opération était permise était d'essayer de voir si elle a réussi ou échoué. En utilisant l'opération ACCESS dans le protocole NFSv4.1, le client peut demander au serveur d'indiquer si une ou plusieurs classes d'opérations sont ou non permises. L'opération ACCESS est fournie pour permettre aux clients de vérifier avant de faire une série d'opérations qui vont résulter en un échec d'accès. L'opération OPEN fournit un point où le serveur peut vérifier l'accès à l'objet de fichier et une méthode pour retourner cette information au client. L'opération ACCESS est encore utile pour les opérations de répertoire ou pour être utilisée dans le cas où l'interface access() UNIX est utilisée sur le client.

Les informations retournées par le serveur en réponse à un appel ACCESS ne sont pas permanentes. Elles étaient correctes à l'instant exact où le serveur a effectué les vérifications, mais pas nécessairement ensuite. Le serveur peut révoquer la permission d'accès à tout moment.

Le client devrait utiliser les accréditifs effectifs de l'utilisateur pour construire les informations d'authentification dans la demande ACCESS utilisée pour déterminer les droits d'accès. Ce sont les accréditifs effectifs d'utilisateur et de groupe qui sont utilisés dans les opérations READ et WRITE suivantes.

De nombreuses mises en œuvre ne prennent pas directement en charge la permission ACCESS4_DELETE. Des systèmes d'exploitation comme UNIX vont ignorer le bit ACCESS4_DELETE si il est établi sur une demande d'accès sur un objet non de répertoire. Dans ces systèmes, la permission de suppression sur un fichier est déterminée par les permissions d'accès sur le répertoire dans lequel le fichier réside, au lieu d'être déterminée par les permissions du fichier lui-même. Donc, le gabarit retourné qui énumère quels droits d'accès peuvent être déterminés va avoir la valeur de ACCESS4_DELETE réglée à 0. Cela indique au client que le serveur a été incapable de vérifier ce droit d'accès particulier. Le bit ACCESS4_DELETE dans le gabarit d'accès retourné va alors être ignoré par le client.

18.2 Opération 4 : CLOSE - fermeture de fichier

18.2.1 Arguments

```
struct CLOSE4args {                                  /* CURRENT_FH : objet */
        seqid4        seqid;
        stateid4      open_stateid;
  };
```

18.2.2 Résultats

```
union CLOSE4res switch (nfsstat4 status) {
   cas NFS4_OK :          stateid4      open_stateid;
   défaut :  vide
  };
```

18.2.3 Description

L'opération CLOSE libère le partage de réservation pour le fichier régulier ou attribut nommé comme spécifié par la bride de fichier courante. Les informations de partage de réservation et autre état libérées chez le serveur par suite de ce CLOSE sont seulement celles associées à l'identifiant d'état fourni. L'état associé aux autres OPEN n'est pas affecté.

Si des verrous de gamme d'octets sont détenus, le client DEVRAIT libérer tous les verrous avant d'envoyer un CLOSE. Le serveur PEUT libérer tous les verrous en instance sur un CLOSE, mais certains serveurs peuvent ne pas prendre en charge le CLOSE d'un fichier qui a encore des verrous de gamme d'octets détenus. Le serveur DOIT retourner un échec si des verrous existeraient après le CLOSE.

L'argument seqid PEUT avoir toute valeur, et le serveur DOIT ignorer le seqid.

En cas de succès, la bride de fichier courante conserve sa valeur.

Le serveur PEUT exiger que la combinaison de principal, nuance de sécurité, et, si applicable, mécanisme GSS qui a envoyé la demande OPEN soit aussi celle du CLOSE sur le fichier. Cela pourrait n'être pas possible si les accréditifs pour le principal ne sont plus disponibles. Le serveur PEUT permettre l'accréditif de machine ou de SSV (voir le paragraphe 18.35) pour envoyer le CLOSE.

18.2.4 Mise en œuvre

Même si CLOSE retourne un identifiant d'état, cet identifiant d'état n'est pas utile au client et devrait être traité comme déconseillé. CLOSE "ferme" l'état associé à tous les OPEN pour le fichier par un seul propriétaire d'ouverture. Comme noté ci-dessus, CLOSE va libérer tout l'état de verrouillage de fichier ou retourner une erreur. Donc, l'identifiant d'état retourné par CLOSE n'est pas utile pour les opérations qui suivent. Pour aider à toute utilisation de cet identifiant d'état par les clients, le serveur DEVRAIT retourner l'identifiant d'état spécial invalide (la valeur "autre" est zéro et le champ "seqid" est NFS4_UINT32_MAX, voir le paragraphe 8.2.3).

Une opération CLOSE peut rendre des délégations accordables alors qu'elles ne l'étaient pas précédemment. Les serveurs peuvent choisir de répondre immédiatement si il y a des demandes de délégation en instance ou peuvent répondre à la situation ultérieurement.

18.3 Opération 5 : COMMIT - engagement des données en antémémoire

18.3.1 Arguments

```
struct COMMIT4args {                                /* CURRENT_FH : objet */
      offset4        offset;
      count4         count;
};
```

18.3.2 Résultats

```
struct COMMIT4resok {
        verifier4       writeverf;
};

union COMMIT4res switch (nfsstat4 status) {
   cas NFS4_OK :  COMMIT4resok   resok4;
   défaut : vide
  };
```

18.3.3 Description

L'opération COMMIT force ou purge les données modifiées non engagées, dans une mémorisation stable pour le fichier spécifié par la bride de fichier courante. Les données purgées sont celles qui ont été précédemment écrites avec une ou plusieurs opérations WRITE qui avaient le champ "committed" de leur champ de résultat réglé à UNSTABLE4.

Le décalage spécifie dans le fichier la position où la purge va commencer. Une valeur de décalage de zéro signifie de purger les données en commençant au début du fichier. Le compte spécifie le nombre d'octets de données à purger. Si le compte est zéro, une purge à partir du décalage à la fin du fichier est faite.

Le serveur retourne un vérificateur d'écriture à l'achèvement réussi du COMMIT. Le vérificateur d'écriture est utilisé par le client pour déterminer si le serveur a redémarré entre les opérations WRITE initiales et le COMMIT. Le client fait cela en comparant le vérificateur d'écriture retourné des opérations WRITE initiales et le vérificateur retourné par l'opération COMMIT. Le serveur doit faire varier la valeur du vérificateur d'écriture à chaque événement de serveur ou instanciation qui peut conduire à une perte de données non engagées. Le plus généralement, ceci se produit quand le serveur est redémarré ; cependant, d'autres événements chez le serveur peuvent résulter aussi en données non engagées.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.3.4 Mise en œuvre

L'opération COMMIT est similaire en fonctionnement et signification à l'interface système POSIX fsync() [IEEE-1003.1] qui synchronise l'état d'un fichier avec le disque (les données et métadonnées du fichier sont purgées du disque ou d'une mémorisation stable). COMMIT effectue la même opération pour un client, purgeant toutes les données et métadonnées non synchronisées sur le serveur du disque ou mémorisation stable du serveur pour le fichier spécifié. Comme avec fsync(), il peut y avoir des données modifiées ou non modifiées à synchroniser. Les données peuvent avoir été synchronisées par l'activité normale de synchronisation périodique d'antémémoire du serveur. COMMIT devrait retourner NFS4_OK, sauf si il y a eu une erreur inattendue.

COMMIT diffère de fsync() en ce qu'il est possible que le client purge une gamme du fichier (probablement déclenchée par un schéma de réclamation de mémoire tampon chez le client avant que le fichier ait été complètement écrit).

La mise en œuvre de serveur de COMMIT est raisonnablement simple. Si le serveur reçoit une demande COMMIT de fichier complet, c'est-à-dire, commençant au décalage zéro et compte zéro, il devrait faire l'équivalent de l'application de fsync() au fichier entier. Autrement, il devrait s'arranger pour avoir les données modifiées dans la gamme spécifiée par le décalage et le compte à purger dans une mémorisation stable. Dans les deux cas, toutes les métadonnées associées au fichier doivent être purgées dans la mémorisation stable avant de revenir. Ce n'est pas une erreur qu'il n'y ait rien à purger sur le serveur. Cela signifie que les données et métadonnées qu'il était nécessaire de purger ont déjà été purgées ou perdues durant la dernière défaillance du serveur.

La mise en œuvre de client de COMMIT est un peu plus complexe. Il y a deux raisons pour vouloir engager une mémoire tampon de client à une mémorisation stable. La première est que le client veut réutiliser une mémoire tampon. Dans ce cas, le décalage et le compte de la mémoire tampon sont envoyés au serveur dans la demande COMMIT. Le serveur purge alors toutes les données modifiées sur la base du décalage et du compte, et purge toutes les métadonnées modifiées associées au fichier. Il retourne alors l'état de la purge et du vérificateur d'écriture. La seconde raison que le client génère un COMMIT est pour la purge du fichier complet, comme cela peut être fait à la clôture. Dans ce cas, le client va rassembler toutes les mémoires tampons pour ce fichier qui contiennent des données non engagées, faire l'opération COMMIT avec un décalage de zéro et un compte de zéro, et ensuite libérer toutes ces mémoires tampons. Tous les autres contenus de mémoire tampon vont être envoyés au serveur de la façon normale.

Après qu'une mémoire tampon est écrite (via l'opération WRITE) par le client avec le champ "committed" dans le résultat de WRITE réglé à UNSTABLE4, la mémoire tampon doit être considérée comme modifiée par le client jusqu'à ce que la mémoire tampon soit purgée via une opération COMMIT ou écrite via une opération WRITE avec le champ "committed" dans le résultat réglé à FILE_SYNC4 ou DATA_SYNC4. Ceci est fait pour empêcher la mémoire tampon d'être libérée et réutilisée avant que les données puissent être purgées dans une mémorisation stable sur le serveur.

Quand une réponse est retournée d'une opération WRITE ou COMMIT et qu'elle contient un vérificateur d'écriture qui diffère de celui précédemment retourné par le serveur, le client va devoir retransmettre toutes les mémoires tampons contenant des données non engagées sur le serveur. Comment le faire dépend de la mise en œuvre. Si il y a seulement une mémoire tampon intéressée, alors elle devrait être envoyée dans une demande WRITE avec le paramètre stable FILE_SYNC4. Si il y a plus d'une mémoire tampon, cela pourrait valoir la peine de retransmettre toutes les mémoires tampons dans des opérations WRITE avec le paramètre stable réglé à UNSTABLE4 et ensuite de retransmettre l'opération COMMIT pour purger toutes les données sur le serveur dans une mémorisation stable. Cependant, si le serveur retourne de façon répétée au COMMIT un vérificateur qui diffère de celui retourné par WRITE, la seule façon de s'assurer d'un progrès est de retransmettre toutes les mémoires tampons avec des demandes WRITE comportant le paramètre stable FILE_SYNC4.

La description ci-dessus s'applique aux systèmes fondés sur la mise en antémémoire de page ainsi qu'aux systèmes fondés sur la mise en antémémoire de mémoire tampon. Dans les premiers systèmes, le système de mémoire virtuelle va devoir être modifié à la place de l'antémémoire de mémoire tampon.

18.4 Opération 6 : CREATE - crée un objet de fichier non régulier

18.4.1 Arguments

```
union createtype4 switch (nfs_ftype4 type) {
   cas NF4LNK :          linktext4 linkdata;
   cas NF4BLK :
   cas NF4CHR :          specdata4 devdata;
   cas NF4SOCK :
   cas NF4FIFO :
   cas NF4DIR :  vide
   défaut :   vide                                  /* le serveur devrait retourner NFS4ERR_BADTYPE */
  };

struct CREATE4args {                               /* CURRENT_FH : répertoire pour création */
        createtype4     objtype ;
        component4      objname ;
        fattr4          createattrs ;
  };
```

18.4.2 Résultats

```
struct CREATE4resok {
      change_info4    cinfo ;
      bitmap4         attrset ;                    /* attributs établis */
};

union CREATE4res switch (nfsstat4 status) {
   cas NFS4_OK :  CREATE4resok resok4;             /* nouveau CURRENTFH : objet créé */
   défaut : vide
  };
```

18.4.3 Description

L'opération CREATE crée un objet de fichier autre qu'un fichier ordinaire dans un répertoire avec un nom donné. L'opération OPEN DOIT être utilisée pour créer un fichier régulier ou un attribut nommé.

La bride de fichier courante doit être un répertoire : un objet de type NF4DIR. Si la bride de fichier courante est un répertoire d'attribut (type NF4ATTRDIR) l'erreur NFS4ERR_WRONG_TYPE est retournée. Si la bride de fichier courante désigne tout autre type d'objet, l'erreur NFS4ERR_NOTDIR résulte.

Le objname spécifie le nom pour le nouvel objet. Le objtype détermine le type de l'objet à créer : layout, symlink, etc. Si le type d'objet spécifié est celui d'un fichier ordinaire, d'un attribut nommé, ou d'un répertoire des attributs nommés, l'erreur NFS4ERR_BADTYPE résulte.

Si un objet du même nom existe déjà dans le répertoire, le serveur va retourner l'erreur NFS4ERR_EXIST.

Pour le répertoire où le nouvel objet de fichier a été créé, le serveur retourne les informations change_info4 dans cinfo. Avec le champ atomique de type de données change_info4, le serveur va indiquer si les attributs de changement avant et après ont été obtenus de façon atomique à l'égard de la création de l'objet de fichier.

Si le objname a une longueur de zéro, ou si objname ne respecte pas la définition en UTF-8, l'erreur NFS4ERR_INVAL va être retournée.

La bride de fichier courante est remplacée par celle du nouvel objet.

Les createattrs spécifient l'ensemble initial des attributs pour l'objet. L'ensemble des attributs peut inclure tout attribut écrivable pour le type d'objet. Quand l'opération réussit, le serveur va retourner au client un gabarit d'attribut signifiant quels attributs ont été établis avec succès pour l'objet.

Si createattrs n'inclut ni l'attribut de propriétaire ni une ACL avec une ACE pour le propriétaire, et si le système de fichiers du serveur à la fois prend en charge et exige un attribut de propriétaire (ou une ACE de propriétaire) alors le serveur DOIT déduire le propriétaire (ou l'ACE de propriétaire). Cela va normalement être du principal indiqué dans les accréditifs RPC de l'appel, mais l'environnement de fonctionnement du serveur ou la sémantique du système de fichiers peut imposer d'autres méthodes de déduction. De même, si createattrs n'inclut ni l'attribut de groupe ni une ACE de groupe, et si le système de fichiers du serveur à la fois prend en charge et exige la notion d'un attribut de groupe (ou ACE de groupe) le serveur DOIT déduire l'attribut de groupe (ou l'ACE de propriétaire correspondant) pour le fichier. Cela pourrait être à partir des accréditifs de l'appel RPC, comme le principal du groupe si les accréditifs l'incluent (comme avec AUTH_SYS) à partir de l'identifiant de groupe associé au principal dans les accréditifs (par exemple, les systèmes POSIX ont une base de données d'utilisateurs [IEEE-1003.1] qui a un identifiant de groupe pour chaque identifiant d'utilisateur) hérités du répertoire dans lequel l'objet est créé, ou n'importe quoi d'autre qu'impose le système de fonctionnement du serveur ou la sémantique du système de fichiers. Cela s'applique aussi à l'opération OPEN.

À l'inverse, il est possible que le client spécifie dans createattrs un attribut de propriétaire, un attribut de groupe, ou une ACL pour lequel le principal indiqué dans les accréditifs de l'appel RPC n'a pas les permissions de créer de fichiers. L'erreur à retourner dans cette instance est NFS4ERR_PERM. Cela s'applique aussi à l'opération OPEN.

Si la bride de fichier courante désigne un répertoire pour lequel un autre client détient une délégation de répertoire, alors, sauf si la délégation est telle que la situation peut se résoudre en envoyant une notification, la délégation DOIT être rappelée, et l'opération CREATE NE DOIT PAS se poursuivre tant que la délégation n'est pas retournée ou révoquée. Sauf lorsque cela se produit très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites pendant que la délégation reste en instance.

Quand la bride de fichier courante désigne un répertoire pour lequel existent une ou plusieurs délégations de répertoire, alors, quand ces délégations demandent de telles notifications, NOTIFY4_ADD_ENTRY va être généré par suite de cette opération.

Si la capacité FSCHARSET_CAP4_ALLOWS_ONLY_UTF8 est établie (paragraphe 14.4) et si une liaison symbolique est créée, alors le contenu de la liaison symbolique DOIT être en codage UTF-8.

18.4.4 Mise en œuvre

Si le client désire régler des valeurs d'attribut après le create, une opération SETATTR peut être ajoutée à la demande COMPOUND afin que les attributs appropriés soient établis.

18.5 Opération 7 : DELEGPURGE - purge les délégations en attente de récupération

18.5.1 Arguments

```
struct DELEGPURGE4args {
        clientid4       clientid;
};
```

18.5.2 Résultats

```
struct DELEGPURGE4res {
      nfsstat4        status;
};
```

18.5.3 Description

Cette opération purge toutes les délégations en attente de récupération pour un client donné. C'est utile pour les clients qui

n'engagent pas les informations de délégation à une mémorisation stable pour indiquer que des demandes en conflit ne doivent pas être retardées par le serveur en attente de la récupération des informations de délégation.

Le client N'est PAS spécifié par le champ clientid de la demande. Le client DEVRAIT régler le champ client à zéro, et le serveur DOIT ignorer le champ clientid. À la place, le serveur DOIT déduire l'identifiant de client de la valeur de l'identifiant de session dans les arguments de l'opération SEQUENCE qui précède DELEGPURGE dans la demande COMPOUND.

L'opération DELEGPURGE devrait être utilisée par les clients qui enregistrent les informations de délégation sur une mémorisation stable chez le client. Dans ce cas, après que le client récupère de toutes les délégations qu'il connaît, il devrait immédiatement envoyer une opération DELEGPURGE. Le faire va notifier au serveur qu'aucune délégation supplémentaire pour le client ne va être récupérée, lui permettant de libérer les ressources, et éviter de retarder les autres clients qui ont fait des demandes en conflit avec les délégations non récupérées. L'ensemble de délégations connues du serveur et du client pourrait être différent. La raison en est que après l'envoi d'une demande qui a résulté en une délégation, le client pourrait subir une défaillance avant qu'il ait à la fois reçu la délégation et engagé la délégation à la mémorisation stable du client.

Le serveur PEUT prendre en charge DELEGPURGE, mais si il ne le fait pas, il NE DOIT prendre en charge ni CLAIM_DELEGATE_PREV ni CLAIM_DELEG_PREV_FH.

18.6 Opération 8 : DELEGRETURN - retourne la délégation

18.6.1 Arguments

```
struct DELEGRETURN4args {                       /* CURRENT_FH : objet délégué */
        stateid4        deleg_stateid;
};
```

18.6.2 Résultats

```
struct DELEGRETURN4res {
        nfsstat4        status;
};
```

18.6.3 Description

L'opération DELEGRETURN retourne la délégation représentée par la bride de fichier et l'identifiant d'état courants.

Les délégations peuvent être retournées volontairement (c'est-à-dire, avant que le serveur les ait rappelées) ou quand elles sont rappelées. Dans l'un et l'autre cas, le client doit propager correctement l'état changé dans le contexte de la délégation au serveur avant de retourner la délégation.

Le serveur PEUT exiger que la combinaison de principal, nuance de sécurité, et si applicable, mécanisme de GSS, qui a acquis la délégation soit aussi celle à qui envoyer le DELEGRETURN sur le fichier. Cela pourrait n'être pas possible si les accréditifs pour le principal ne sont plus disponibles. Le serveur PEUT permettre les accréditifs de machine ou les accréditifs de SSV (voir le paragraphe 18.35) pour envoyer DELEGRETURN.

18.7 Opération 9 : GETATTR - obtient les attributs

18.7.1 Arguments

```
struct GETATTR4args {                           /* CURRENT_FH : objet */
     bitmap4        attr_request;
};
```

18.7.2 Résultats

```
struct GETATTR4resok {
```

```
        fattr4          obj_attributes;
};

union GETATTR4res switch (nfsstat4 status) {
   cas NFS4_OK :  GETATTR4resok  resok4;
   défaut : vide
};
```

18.7.3 Description

L'opération GETATTR va obtenir les attributs pour l'objet système de fichiers spécifié par la bride de fichier courante. Le client établit un bit dans l'argument bitmap pour chaque valeur d'attribut qu'il aimerait que le serveur retourne. Le serveur retourne un attribut bitmap qui indique les valeurs d'attribut qu'il est capable de retourner, qui va inclure tous les attributs demandés par le client qui sont les attributs pris en charge par le serveur pour le système de fichiers cible. Cet argument bitmap est suivi par les valeurs d'attribut ordonnées avec le plus faible numéro d'attribut en premier.

Le serveur DOIT retourner une valeur pour chaque attribut que le client demande si l'attribut est pris en charge par le serveur pour le système de fichiers cible. Si le serveur ne prend pas en charge un attribut particulier sur le système de fichiers cible, alors il NE DOIT PAS retourner la valeur de l'attribut et NE DOIT PAS établir le bit d'attribut dans l'argument bitmap de résultat. Le serveur DOIT retourner une erreur si il prend en charge un attribut sur la cible mais ne peut pas obtenir sa valeur. Dans ce cas, aucune valeur d'attribut ne va être retournée.

Les systèmes de fichiers qui sont absents devraient être traités comme ayant la prise en charge d'un très petit ensemble d'attributs comme décrit au paragraphe 11.4.1, même si précédemment, quand le système de fichiers était présent, plus d'attributs étaient pris en charge.

Tous les serveurs DOIVENT prendre en charge les attributs EXIGÉS comme spécifié au paragraphe 5.6, pour tous les systèmes de fichiers, à l'exception des systèmes de fichiers absents.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.7.4 Mise en œuvre

Supposons qu'une délégation OPEN_DELEGATE_WRITE soit détenue par un autre client pour le fichier en question et que size et/ou change soient parmi l'ensemble des attributs interrogés. Le serveur a un choix. D'abord, le serveur peut obtenir la valeur courant réelle de ces attributs du client qui détient la délégation en utilisant le rappel CB_GETATTR. Ou, le serveur, en particulier quand le client délégué ne répond pas, peut rappeler la délégation en question. Le GETATTR NE DOIT PAS se faire avant que ce produise une des conditions suivantes :

* les valeurs d'attribut demandées sont retournées dans la réponse à CB_GETATTR,
* la délégation OPEN_DELEGATE_WRITE est retournée,
* la délégation OPEN_DELEGATE_WRITE est révoquée.

Sauf si une des conditions ci-dessus se produit très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées alors qu'une délégation est en instance.

18.8 Opération 10 : GETFH - obtient la bride de fichier courante

18.8.1 Arguments

```
/* CURRENT_FH: */
Vide
```

18.8.2 Résultats

```
struct GETFH4resok {
        nfs_fh4         objet ;
  };
```

```
union GETFH4res switch (nfsstat4 status) {
   cas NFS4_OK : GETFH4resok     resok4;
   défaut : vide
};
```

18.8.3 Description

Cette opération retourne la valeur courante de bride de fichier.

En cas de succès, la bride de fichier courante conserve sa valeur.

Comme décrit au paragraphe 2.10.6.4, il est EXIGÉ ou RECOMMANDÉ que GETFH suive immédiatement certaines opérations, et les serveurs sont libres de rejeter de telles opérations si le client échoue à insérer GETFH dans la demande comme EXIGÉ ou RECOMMANDÉ. Le paragraphe 18.16.4.1 fournit des justifications supplémentaires sur les raisons pour lesquelles GETFH DOIT suivre OPEN.

18.8.4 Mise en œuvre

Les opérations qui changent la bride de fichier courante comme LOOKUP ou CREATE ne retournent pas automatiquement la nouvelle bride de fichier comme résultat. Par exemple, si un client a besoin de chercher une entrée de répertoire et obtient sa bride de fichier, alors la demande suivante est nécessaire.

```
 PUTFH (bride de fichier de répertoire)
 LOOKUP (nom d'entrée)
 GETFH
```

18.9 Opération 11 : LINK - crée une liaison à un fichier

18.9.1 Arguments

```
struct LINK4args {
      /* SAVED_FH : objet de source */
      /* CURRENT_FH : répertoire cible*/
      component4      newname;
};
```

18.9.2 Résultats

```
struct LINK4resok {
        change_info4    cinfo;
};

union LINK4res switch (nfsstat4 status) {
   cas NFS4_OK :          LINK4resok resok4;
   défaut :   vide
};
```

18.9.3 Description

L'opération LINK crée un nouveau nom supplémentaire pour le fichier représenté par la bride de fichier sauvegardée, comme établi par l'opération SAVEFH, dans le répertoire représenté par la bride de fichier courante. Le fichier existant et le répertoire cible doivent résider dans le même système de fichiers sur le serveur. En cas de succès, la bride de fichier courante va continuer d'être le répertoire cible. Si un objet existe dans le répertoire cible avec le même nom que newname, le serveur doit retourner NFS4ERR_EXIST.

Pour le répertoire cible, le serveur retourne les informations change_info4 dans cinfo. Avec le champ atomique de type de données change_info4, le serveur va indiquer si les attributs de changement avant et après ont été obtenus de façon atomique par rapport à la création de la liaison.

Si le newname a une longueur de zéro, ou si newname ne respecte pas la définition UTF-8, l'erreur NFS4ERR_INVAL va être retournée.

18.9.4 Mise en œuvre

Le serveur PEUT imposer des restrictions à l'opération LINK telles que LINK ne puisse pas être fait quand le fichier est ouvert ou quand cette ouverture est faite par des protocoles particuliers, ou avec des options ou modes d'accès particuliers. Quand LINK est rejeté à cause de telles restrictions, l'erreur NFS4ERR_FILE_OPEN est retournée.

Si un serveur met en œuvre de telles restrictions et si ces restrictions incluent des cas d'ouvertures NFSv4 qui empêchent le succès de l'exécution d'une liaison, le serveur doit rappeler toute délégation qui pourrait cacher l'existence d'ouvertures relevant de cette décision. La raison est que quand un client détient une délégation, le serveur pourrait n'avoir pas un compte précis des ouvertures pour ce client, car le client peut exécuter des OPEN et des CLOSE en local. L'opération LINK doit être retardée seulement jusqu'à ce qu'un résultat définitif puisse être obtenu. Par exemple, supposons qu'il y ait plusieurs délégations et que l'une d'elles établisse une ouverture dont la présence empêcherait la liaison. Étant donnée la sémantique de serveur, NFS4ERR_FILE_OPEN peut être retourné à l'appelant aussitôt que cette délégation est retournée sans attendre que d'autres délégations soient retournées. De même, si de telles ouvertures ne sont pas associées à des délégations, NFS4ERR_FILE_OPEN peut être retourné immédiatement sans qu'un rappel de délégation soit fait.

Si la bride de fichier courante désigne un répertoire pour lequel un autre client détient une délégation de répertoire, alors, sauf si la délégation est telle que la situation peut être résolue en envoyant une notification, la délégation DOIT être rappelée, et l'opération ne peut pas être effectuée avec succès avant que la délégation soit retournée ou révoquée. Sauf lorsque cela arrive très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites alors que la délégation reste en instance.

Quand la bride de fichier courante désigne un répertoire pour lequel une ou plusieurs délégations de répertoire existent, alors, quand ces délégations demandent de telles notifications, au lieu d'un rappel, NOTIFY4_ADD_ENTRY va être généré par suite de l'opération LINK.

Si le système de fichiers courant prend en charge l'attribut numlinks, et si d'autres clients ont des délégations sur le fichier relié, alors ces délégations DOIVENT être rappelées et l'opération LINK NE DOIT PAS avoir lieu jusqu'à ce que toutes les délégations soient retournées ou révoquées. Sauf lorsque cela arrive très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites alors que la délégation reste en instance.

Les changements à toute propriété des fichiers reliés "physiquement" sont reflétés dans tous les fichiers reliés. Quand une liaison est faite à un fichier, les attributs pour le fichier devraient avoir une valeur pour numlinks supérieure de un à la valeur avant l'opération LINK.

La déclaration "le fichier et le répertoire cible doivent résider dans le même système de fichiers sur le serveur" signifie que les champs fsid dans les attributs pour les objets sont les mêmes. Si ils résident sur des systèmes de fichiers différents, l'erreur NFS4ERR_XDEV est retournée. Cette erreur peut être retournée par certains serveurs quand il y a une partition interne d'un système de fichiers que l'opération LINK violerait.

Sur certains serveurs, "." et ".." sont des valeurs illégales pour newname et l'erreur NFS4ERR_BADNAME va être retournée si elles sont spécifiées.

Quand la bride de fichier courante désigne un répertoire des attributs nommés et que l'objet à relier (la bride de fichier sauvegardée) n'est pas un attribut nommé pour le même objet, l'erreur NFS4ERR_XDEV DOIT être retournée. Quand la bride de fichier sauvegardée désigne un attribut nommé et que la bride de fichier courante n'est pas le répertoire des attributs nommés approprié, l'erreur NFS4ERR_XDEV DOIT aussi être retournée.

Quand la bride de fichier courante désigne un répertoire des attributs nommés et que l'objet à relier (la bride de fichier sauvegardée) est un attribut nommé dans ce répertoire, le serveur peut retourner l'erreur NFS4ERR_NOTSUPP.

Dans le cas où newname est déjà relié au fichier représenté par la bride de fichier sauvegardée, le serveur va retourner NFS4ERR_EXIST.

Noter que les liaisons symboliques sont créés avec l'opération CREATE.

18.10 Opération 12 : LOCK - crée un verrou

18.10.1 Arguments

```
/* * Pour LOCK, transition de open_stateid et lock_owner à un identifiant d'état de verrou. */
struct open_to_lock_owner4 {
        seqid4          open_seqid;
        stateid4        open_stateid;
        seqid4          lock_seqid;
        lock_owner4     lock_owner;
};

/* * Pour LOCK, l'identifiant d'état de verrou existant continue de demander un nouveau verrou de fichier pour le même
lock_owner et open_stateid. */
struct exist_lock_owner4 {
        stateid4        lock_stateid;
        seqid4          lock_seqid;
};

union locker4 switch (bool new_lock_owner) {
   cas VRAI :     open_to_lock_owner4     open_owner;
   cas FAUX :     exist_lock_owner4       lock_owner;
};

/* * LOCK/LOCKT/LOCKU : gestion de verrou d'enregistrement */
struct LOCK4args {                                          /* CURRENT_FH : fichier */
        nfs_lock_type4  locktype;
        bool            reclaim;
        offset4         offset;
        length4         length;
        locker4         locker;
};
```

18.10.2 Résultats

```
struct LOCK4denied {
        offset4         offset;
        length4         length;
        nfs_lock_type4  locktype;
        lock_owner4     owner;
};

struct LOCK4resok {
        stateid4        lock_stateid;
};

union LOCK4res switch (nfsstat4 status) {
   cas NFS4_OK :    LOCK4resok     resok4;
   cas NFS4ERR_DENIED : LOCK4denied    denied;
   défaut :   vide
};
```

18.10.3 Description

L'opération LOCK demande un verrou de gamme d'octets pour la gamme d'octets spécifiée par les paramètres de décalage et de longueur, et le type de verrou spécifié dans le paramètre locktype. Si c'est une demande de réclamation, le paramètre reclaim va être VRAI.

Les octets dans un fichier peuvent être verrouillés même si ces octets ne sont pas actuellement alloués au fichier. Pour verrouiller le fichier à partir d'un décalage spécifique jusqu'à la fin de fichier (quelle que soit la longueur réelle du fichier)

on utilise un champ Longueur égal à NFS4_UINT64_MAX. Le serveur DOIT retourner NFS4ERR_INVAL sous les combinaisons suivantes de longueur et décalage :

* Longueur est égal à zero.
* Longueur n'est pas égal à NFS4_UINT64_MAX, et la somme de longueur et décalage excède NFS4_UINT64_MAX.

Les serveurs de 32 bits sont des serveurs qui prennent en charge le verrouillage pour les décalages d'octets qui tiennent dans 32 bits (c'est-à-dire, inférieur ou égal à NFS4_UINT32_MAX). Si le client spécifie une gamme qui recouvre un ou plusieurs octets au delà du décalage NFS4_UINT32_MAX mais ne se termine pas au décalage NFS4_UINT64_MAX, alors un tel serveur de 32 bits DOIT retourner l'erreur NFS4ERR_BAD_RANGE.

Si le serveur retourne NFS4ERR_DENIED, le propriétaire, le décalage, et la longueur du verrou en conflit sont retournés.

L'argument locker spécifie le propriétaire de verrou qui est associé à l'opération LOCK. La structure est une union commutée qui indique si le client a déjà créé l'état de verrouillage de gamme d'octets associé au fichier ouvert et propriétaire de verrou courants. Dans le cas où il l'a fait, l'argument est juste un identifiant d'état représentant l'ensemble de verrous associés à ce fichier ouvert et propriétaire de verrou, ainsi qu'avec une valeur de lock_seqid qui PEUT être n'importe quelle valeur et DOIT être ignorée du serveur. Dans le cas où aucun état de verrouillage de gamme d'octets n'a été établi, ou lorsque le client n'a pas d'identifiant d'état disponible, l'argument contient l'identifiant d'état du fichier ouvert avec lequel ce verrou va être associé, ainsi qu'avec le propriétaire de verrou auquel le verrou est à associer. Le cas open_to_lock_owner couvre le tout premier verrou fait par un propriétaire de verrou pour un fichier ouvert donné et offre une méthode pour utiliser l'état établi de open_stateid à transiter à l'utilisation d'un identifiant d'état de verrou.

Les champs suivants du paramètre locker PEUVENT être réglés à toute valeur par le client et DOIVENT être ignorés par le serveur :

* Le champ clientid du champ lock_owner du champ open_owner (locker.open_owner.lock_owner.clientid). La raison pour laquelle le serveur DOIT ignorer le champ clientid est que le serveur DOIT déduire l'identifiant de client de l'identifiant de session de l'opération SEQUENCE de la demande COMPOUND.

* Les champs open_seqid et lock_seqid du champ open_owner (locker.open_owner.open_seqid et locker.open_owner.lock_seqid).

* Le champ lock_seqid du champ lock_owner (locker.lock_owner.lock_seqid).

Noter que l'identifiant de client qui apparaît dans une structure LOCK4denied est le client réel associé au verrou en conflit, si c'est l'identifiant de client associé à la session courante ou à une différente. Donc, si le serveur retourne NFS4ERR_DENIED, il DOIT régler le champ clientid du champ de propriétaire au champ refusé.

Si la bride de fichier courante n'est pas un fichier ordinaire, une erreur va être retournée au client. Dans le cas où la bride de fichier courante représente un objet de type NF4DIR, NFS4ERR_ISDIR est retourné. Si la bride de fichier courante désigne une liaison symbolique, NFS4ERR_SYMLINK est retourné. Dans tous les autres cas, NFS4ERR_WRONG_TYPE est retourné.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.10.4 Mise en œuvre

Si le serveur est incapable de déterminer le décalage exact et la longueur du verrou de gamme d'octets en conflit, les mêmes décalage et longueur que fournis dans les arguments devraient être retournés dans les résultats refusés.

Les opérations LOCK sont soumises à des vérifications de permission et à des vérifications sur le type d'accès du fichier associé. Cependant, les droits et modes spécifiques requis pour les divers types de verrous reflètent la sémantique du système de fichiers exporté par le serveur, et ne sont pas spécifiés par le protocole. Par exemple, Windows 2000 permet de verrouiller en écriture un fichier ouvert pour un accès en lecture, tandis qu'un système conforme à POSIX ne le fait pas.

Quand le client envoie une opération LOCK qui correspond à une gamme que le propriétaire de verrou a déjà verrouillé (avec le même type de verrou ou un type différent) ou à une sous gamme d'une telle gamme, ou à une gamme d'octets qui inclut plusieurs verrous déjà accordés à ce propriétaire de verrou, en tout ou en partie, et le serveur ne prend pas en charge de telles opérations de verrouillage (c'est-à-dire, ne prend pas en charge la sémantique de verrouillage POSIX) le serveur va retourner l'erreur NFS4ERR_LOCK_RANGE. Dans ce cas, le client peut retourner une erreur, ou il peut émuler les

opérations requises, utilisant seulement LOCK pour les gammes qui n'incluent pas d'octets déjà verrouillés par ce propriétaire de verrou et LOCKU pour les verrous détenus par ce propriétaire de verrou (en spécifiant une correspondance exacte de gamme et type). De même, quand le client envoie une opération LOCK qui revient à mettre à niveau (en changeant d'un verrou READ_LT à un verrou WRITE_LT) ou à dégrader (changer d'un verrou WRITE_LT à un verrou READ_LT) un verrou de gamme d'octets existant, et le serveur ne prend pas en charge un tel verrou, le serveur va retourner NFS4ERR_LOCK_NOTSUPP. De telles opérations peuvent ne pas refléter parfaitement la sémantique requise en présence d'opérations LOCK en conflit provenant d'autres clients.

Quand un client détient une délégation OPEN_DELEGATE_WRITE, le client qui détient cette délégation est assuré qu'il n'y a pas d'ouvertures par d'autres clients. Donc, il ne peut pas y avoir d'opérations LOCK en conflit provenant de tels clients. Donc, le client peut traiter les demandes de verrouillage en local, sans faire d'opérations LOCK sur le serveur. Si il fait cela, il doit être prêt à mettre à jour l'état de verrou sur le serveur, en envoyant les opérations LOCK et LOCKU appropriées avant de retourner la délégation.

Quand un ou plusieurs clients détiennent des délégations OPEN_DELEGATE_READ, toute opération LOCK où le serveur met en œuvre la sémantique de verrouillage obligatoire DOIT résulter en le rappel de toutes ces délégations. L'opération LOCK ne peut pas être accordée avant que toutes ces délégations soient retournées ou révoquées. Sauf lorsque cela se produit très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites alors que la délégation reste en instance.

18.11 Opération 13 : LOCKT - vérification de verrou

18.11.1 Arguments

```
struct LOCKT4args {                                    /* CURRENT_FH : fichier */
      nfs_lock_type4   locktype;
      offset4          offset;
      length4          length;
      lock_owner4      owner;
};
```

18.11.2 Résultats

```
union LOCKT4res switch (nfsstat4 status) {
   cas NFS4ERR_DENIED : LOCK4denied    denied;
   cas NFS4_OK : vide
   défaut :  vide
};
```

18.11.3 Description

L'opération LOCKT vérifie le verrou comme spécifié dans les arguments. Si un verrou en conflit existe, le propriétaire, le décalage, la longueur, et le type du verrou en conflit sont retournés. Le champ owner dans le résultat inclut l'identifiant de client du propriétaire du verrou en conflit, si c'est l'identifiant de client associé à la session en cours ou un identifiant de client différent. Si aucun verrou n'est détenu, rien d'autre que NFS4_OK n'est retourné. Les types de verrou READ_LT et READW_LT sont traités de la même façon que dans une vérification de verrou en conflit sans égard au blocage ou non blocage. La même chose est vraie pour WRITE_LT et WRITEW_LT.

Les gammes sont spécifiées comme pour LOCK. Les erreurs NFS4ERR_INVAL et NFS4ERR_BAD_RANGE sont retournées dans les mêmes circonstances que pour LOCK.

Le champ clientid du propriétaire PEUT être réglé à toute valeur par le client et DOIT être ignoré par le serveur. La raison pour laquelle le serveur DOIT ignorer le champ clientid est que le serveur DOIT déduire l'identifiant de client de l'identifiant de session provenant de l'opération SEQUENCE de la demande COMPOUND.

Si la bride de fichier courante n'est pas un fichier ordinaire, une erreur va être retournée au client. Dans le cas où la bride de fichier courante représente un objet de type NF4DIR, NFS4ERR_ISDIR est retourné. Si la bride de fichier courante désigne une liaison symbolique, NFS4ERR_SYMLINK est retourné. Dans tous les autres cas, NFS4ERR_WRONG_TYPE est retourné.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.11.4 Mise en œuvre

Si le serveur est incapable de déterminer le décalage et la longueur exacts du verrou en conflit, les mêmes décalage et longueur que fournis dans les arguments devraient être retournés dans le résultat refusé.

LOCKT utilise un lock_owner4 plutôt qu'un stateid4, comme est utilisé dans LOCK pour identifier le propriétaire. C'est parce que le client n'a pas à ouvrir le fichier pour vérifier l'existence d'un verrou, de sorte qu'un identifiant d'état pourrait n'être pas disponible.

Comme noté au paragraphe 18.10.4, certains serveurs peuvent retourner NFS4ERR_LOCK_RANGE à certaines opérations LOCK (par ailleurs non en conflit) qui chevauchent des gammes déjà accordées au propriétaire de verrou actuel.

La vérification de l'opération LOCKT pour des verrous en conflit DEVRAIT exclure les verrous pour le propriétaire de verrou actuel, et donc devrait retourner NFS4_OK dans ce cas. Noter que cela signifie qu'un serveur pourrait retourner NFS4_OK à une demande LOCKT même si une opération LOCK pour la même gamme et propriétaire de verrou échouerait avec NFS4ERR_LOCK_RANGE.

Quand un client détient une délégation OPEN_DELEGATE_WRITE, il peut choisir (voir le paragraphe 18.10.4) de traiter en local les demandes LOCK. Dans ce cas, les demandes LOCKT vont similairement être traitées en local.

18.12 Opération 14 : LOCKU - déverrouillage de fichier

18.12.1 Arguments

```
struct LOCKU4args {                                    /* CURRENT_FH : fichier */
      nfs_lock_type4   locktype;
      seqid4           seqid;
      stateid4         lock_stateid;
      offset4          offset;
      length4          length;
};
```

18.12.2 Résultats

```
union LOCKU4res switch (nfsstat4 status) {
   cas   NFS4_OK :   stateid4      lock_stateid;
   défaut :  vide
};
```

18.12.3 Description

L'opération LOCKU déverrouille le verrou de gamme d'octets spécifié par les paramètres. Le client peut régler le champ locktype à toute valeur légale pour le type énuméré nfs_lock_type4, et le serveur DOIT accepter toute valeur légale pour locktype. Une valeur légale pour locktype n'a pas d'effet sur le succès ou l'échec de l'opération LOCKU.

Les gammes sont spécifiées comme pour LOCK. Les erreurs NFS4ERR_INVAL et NFS4ERR_BAD_RANGE sont retournées dans les mêmes circonstances que pour LOCK.

Le paramètre seqid PEUT avoir n'importe quelle valeur et le serveur DOIT l'ignorer.

Si la bride de fichier courante n'est pas un fichier ordinaire, une erreur va être retournée au client. Dans le cas où la bride de fichier courante représente un objet de type NF4DIR, NFS4ERR_ISDIR est retourné. Si la bride de fichier courante désigne une liaison symbolique, NFS4ERR_SYMLINK est retourné. Dans tous les autres cas, NFS4ERR_WRONG_TYPE est retourné.

En cas de succès, la bride de fichier courante conserve sa valeur.

Le serveur PEUT exiger que la combinaison de principal, nuance de sécurité, et si applicable, mécanisme GSS, qui a envoyé une opération LOCK soit aussi celle à qui envoyer LOCKU sur le fichier. Cela pourrait n'être pas possible si les accréditifs pour le principal ne sont plus disponibles. Le serveur PEUT permettre que les accréditifs de machine ou les accréditifs de SSV (voir le paragraphe 18.35) envoient LOCKU.

18.12.4 Mise en œuvre

Si la zone à déverrouiller ne correspond pas exactement à un verrou actuellement détenu par le propriétaire de verrou, le serveur peut retourner l'erreur NFS4ERR_LOCK_RANGE. Cela inclut le cas où la zone n'est pas verrouillée, où la zone est une sous gamme de la zone verrouillée, où elle chevauche la zone verrouillée sans correspondre exactement, ou la zone spécifiée inclut plusieurs verrous détenus par le propriétaire de verrou. Dans tous ces cas, permis par la sémantique de verrouillage POSIX [IEEE-1003.1], un client recevant cette erreur devrait, si il désire prendre en charge de telles opérations, simuler l'opération en utilisant LOCKU sur des gammes correspondant aux verrous qu'il détient actuellement, éventuellement suivi par des opérations LOCK pour les sous gammes à ne pas déverrouiller.

Quand un client détient une délégation OPEN_DELEGATE_WRITE, il peut choisir (voir le paragraphe 18.10.4) de traiter les demandes LOCK en local. Dans ce cas, les opérations LOCKU vont aussi être traitées en local.

18.13.1 Arguments

18.13 Opération 15 : LOOKUP - recherche un nom de fichier

```
struct LOOKUP4args {                                    /* CURRENT_FH : répertoire */
      component4      objname;
};
```

18.13.2 Résultats

```
struct LOOKUP4res {                                     /* Nouvelle CURRENT_FH : objet */
      nfsstat4        status;
};
```

18.13.3 Description

L'opération LOOKUP cherche ou trouve un objet système de fichiers en utilisant le répertoire spécifié par la bride de fichier courante. LOOKUP évalue le composant et si l'objet existe, la bride de fichier courante est remplacée par la bride de fichier du composant.

Si le composant ne peut pas être évalué soit parce que il n'existe pas, soit parce que le client n'a pas la permission d'évaluer le composant, alors une erreur va être retournée et la bride de fichier courante va être inchangée.

Si le composant est une chaîne de longueur zéro, ou si un ou des composants ne respectent pas la définition UTF-8, l'erreur NFS4ERR_INVAL va être retournée.

18.13.4 Mise en œuvre

Si le client veut réaliser l'effet d'une recherche multi composants, il peut construire une demande COMPOUND telle que (et obtenir chaque bride de fichier) :

```
PUTFH  (bride de fichier de répertoire)
LOOKUP "pub"
GETFH
LOOKUP "foo"
GETFH
LOOKUP "bar"
GETFH
```

À la différence de NFSv3, NFSv4.1 permet que des demandes LOOKUP franchissent des points de montage sur le serveur. Le client peut détecter la traversée d'un point de montage en comparant l'attribut fsid du répertoire à l'attribut fsid du répertoire verrouillé. Si les fsid sont différents, alors le nouveau répertoire est un point de montage de serveur. Les clients UNIX qui détectent une traversée de point de montage vont avoir besoin de monter le système de fichiers du serveur. Cela doit être fait pour maintenir les mécanismes de vérification d'identité d'objet de fichier communs aux clients UNIX.

Les serveurs qui limitent l'accès NFS aux systèmes de fichiers "partagés" ou "exportés" devraient fournir un pseudo système de fichiers dans lequel les systèmes de fichiers exportés peuvent être intégrés, afin que les clients puissent naviguer sur l'espace de noms du serveur. La vue des clients d'un pseudo système de fichiers va être limitée aux chemins qui conduisent aux systèmes de fichiers exportés.

Note : les précédentes versions du protocole allouaient une signification particulière aux noms "." et "..". NFSv4.1 n'accorde pas de signification spéciale à ces noms. L'opérateur LOOKUPP doit être utilisé pour chercher un répertoire parent.

Noter que cette opération ne suit pas les liaisons symboliques. Le client est responsable de l'analyse de tous les noms de fichier y compris de ceux qui sont modifiés par des liaisons symboliques rencontrées durant le processus de recherche.

Si la bride de fichier courante fournie n'est pas un répertoire mais un lien symbolique, l'erreur NFS4ERR_SYMLINK est retournée comme erreur. Pour tout autre type de fichier non répertoire, l'erreur NFS4ERR_NOTDIR est retournée.

18.14 Opération 16 : LOOKUPP - recherche du répertoire parent

18.14.1 Arguments

```
/* CURRENT_FH : objet */
Vide
```

18.14.2 Résultats

```
struct LOOKUPP4res {                    /* nouvelle CURRENT_FH : répertoire parent */
      nfsstat4      status;
};
```

18.14.3 Description

La bride de fichier courante est supposée se référer à un répertoire régulier ou à un répertoire des attributs nommés. LOOKUPP alloue la bride de fichier pour son répertoire parent comme étant la bride de fichier courante. Si il n'y a pas de répertoire parent, une erreur NFS4ERR_NOENT doit être retournée. Donc, NFS4ERR_NOENT va être retourné par le serveur quand la bride de fichier courante est à la racine ou au sommet de l'arborescence de fichiers du serveur.

Comme c'est le cas avec LOOKUP, LOOKUPP va aussi traverser des points de montage.

Si la bride de fichier courante n'est pas un répertoire ou un répertoire des attributs nommés, l'erreur NFS4ERR_NOTDIR est retournée.

Si la nuance de sécurité du demandeur ne correspond pas à celle configurée pour le répertoire parent, alors le serveur DEVRAIT retourner NFS4ERR_WRONGSEC (une future révision mineure de NFSv4 peut relever ceci en un DOIT) dans la réponse LOOKUPP. Cependant, si le serveur fait cela, il DOIT prendre en charge l'opération SECINFO_NO_NAME (paragraphe 18.45) afin que le client puisse déterminer en douceur la nuance de sécurité correcte.

Si la bride de fichier courante est un répertoire des attributs nommés qui est associé à un objet système de fichiers via OPENATTR (c'est-à-dire, pas un sous répertoire d'un répertoire des attributs nommés) LOOKUPP DEVRAIT retourner la bride de fichier de l'objet système de fichiers associé.

18.14.4 Mise en œuvre

Un problème à noter est la navigation vers l'amont à partir des répertoires d'attributs nommés. Les répertoires d'attributs

nommés sont essentiellement détachés de l'espace de noms, et cette propriété devrait être représentée en toute sécurité dans l'environnement de fonctionnement du client. LOOKUPP sur un répertoire des attributs nommés peut retourner la bride de fichier du fichier associé, et convoyer cela aux applications pourrait être non sûr car de nombreuses applications s'attendent à ce que le parent d'un objet soit toujours un répertoire. Donc, le client peut vouloir cacher le parent des répertoires d'attributs nommés (représenté comme ".." dans UNIX) ou représenter le répertoire des attributs nommés comme son propre parent (comme c'est normalement fait pour le répertoire racine de système de fichiers dans UNIX).

18.15 Opération 17 : NVERIFY - vérifie la différence des attributs

18.15.1 Arguments

```
struct NVERIFY4args {                                   /* CURRENT_FH : objet */
      fattr4        obj_attributes;
};
```

18.15.2 Résultats

```
struct NVERIFY4res {
      nfsstat4      status;
};
```

18.15.3 Description

Cette opération est utilisée pour mettre un préfixe à une séquence d'opérations à effectuer si un ou plusieurs attributs ont changé dans un objet système de fichiers. Si tous les attributs correspondent, alors l'erreur NFS4ERR_SAME DOIT être retournée.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.15.4 Mise en œuvre

Cette opération est utile comme opérateur de validation d'antémémoire. Si l'objet auquel appartiennent les attributs a changé, alors les opérations suivantes peuvent obtenir de nouvelles données associées à cet objet, par exemple, pour vérifier si un fichier a changé et obtenir les nouvelles données si il a changé :

```
SEQUENCE
PUTFH fh
NVERIFY attrbits attrs
READ 0 32767
```

À la différence de NFSv3, qui envoie d'abord un GETATTR dans un aller-retour de demande/réponse, et si les attributs indiquent que l'antémémoire du client est périmée, envoie alors un READ dans un autre aller-retour de demande/réponse.

Dans le cas où un attribut RECOMMANDÉ est spécifié dans l'opération NVERIFY et où le serveur ne prend pas en charge cet attribut pour l'objet système de fichiers, l'erreur NFS4ERR_ATTRNOTSUPP est retournée au client.

Quand l'attribut rdattr_error ou tout attribut set-only (par exemple, time_modify_set) est spécifié, l'erreur NFS4ERR_INVAL est retournée au client.

18.16 Opération 18 : OPEN - ouvre un fichier régulier

18.16.1 Arguments

```
/** Diverses définitions pour OPEN */
enum createmode4 {
      UNCHECKED4      = 0,
      GUARDED4        = 1,
      EXCLUSIVE4      = 2,                      /* Déconseillé dans NFSv4.1. */
```

```
/* * Nouveau pour NFSv4.1. Si la session est persistante, GUARDED4 DOIT être utilisé. Autrement, utiliser
        EXCLUSIVE4_1 au lieu de EXCLUSIVE4. */
      EXCLUSIVE4_1    = 3
};

struct creatverfattr {
       verifier4      cva_verf;
       fattr4         cva_attrs;
};

union createhow4 switch (createmode4 mode) {
 cas UNCHECKED4 :
 cas GUARDED4 : fattr4 createattrs;
 cas EXCLUSIVE4 : verifier4 createverf;
 cas EXCLUSIVE4_1 : creatverfattr  ch_createboth;
};

enum opentype4 {
      OPEN4_NOCREATE  = 0,
      OPEN4_CREATE       = 1
};

union openflag4 switch (opentype4 opentype) {
 cas OPEN4_CREATE : createhow4     how;
 défaut : vide
};

/* Prochaines définitions utilisées pour délégation OPEN */
enum limit_by4 {
      NFS_LIMIT_SIZE         = 1,
      NFS_LIMIT_BLOCKS   = 2                          /* autres comme nécessaire */
};

struct nfs_modified_limit4 {
      uint32_t       num_blocks;
      uint32_t       bytes_per_block;
};

union nfs_space_limit4 switch (limit_by4 limitby) {
   /* limite spécifiée comme taille de fichier */
   cas NFS_LIMIT_SIZE : uint64_t              filesize;
   /* limite spécifiée par nombre de blocs */
   cas NFS_LIMIT_BLOCKS : nfs_modified_limit4    mod_blocks;
} ;

/* * Constantes de partage d'accès et de refus pour l'argument open  */
  const OPEN4_SHARE_ACCESS_READ   = 0x00000001;
  const OPEN4_SHARE_ACCESS_WRITE  = 0x00000002;
  const OPEN4_SHARE_ACCESS_BOTH   = 0x00000003;

  const OPEN4_SHARE_DENY_NONE     = 0x00000000;
  const OPEN4_SHARE_DENY_READ     = 0x00000001;
  const OPEN4_SHARE_DENY_WRITE    = 0x00000002;
  const OPEN4_SHARE_DENY_BOTH     = 0x00000003;

/* Nouveaux fanions pour le champ share_access de OPEN4args */
  const OPEN4_SHARE_ACCESS_WANT_DELEG_MASK        = 0xFF00;
  const OPEN4_SHARE_ACCESS_WANT_NO_PREFERENCE   = 0x0000;
  const OPEN4_SHARE_ACCESS_WANT_READ_DELEG        = 0x0100;
  const OPEN4_SHARE_ACCESS_WANT_WRITE_DELEG       = 0x0200;
  const OPEN4_SHARE_ACCESS_WANT_ANY_DELEG         = 0x0300;
```

```
  const OPEN4_SHARE_ACCESS_WANT_NO_DELEG         = 0x0400;
  const OPEN4_SHARE_ACCESS_WANT_CANCEL           = 0x0500;
  const OPEN4_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_RESRC_AVAIL = 0x10000;
  const OPEN4_SHARE_ACCESS_WANT_PUSH_DELEG_WHEN_UNCONTENDED = 0x20000;

enum open_delegation_type4 {
      OPEN_DELEGATE_NONE         = 0,
      OPEN_DELEGATE_READ         = 1,
      OPEN_DELEGATE_WRITE        = 2,
      OPEN_DELEGATE_NONE_EXT  = 3          /* nouveau pour v4.1 */
};

enum open_claim_type4 {
/* * Pas une réclamation. */
        CLAIM_NULL                  = 0,
        CLAIM_PREVIOUS              = 1,
        CLAIM_DELEGATE_CUR      = 2,
        CLAIM_DELEGATE_PREV     = 3,
/* * Pas une réclamation. Comme CLAIM_NULL, mais objet identifié par la bride de fichier courante. */
        CLAIM_FH                    = 4,          /* nouveau pour v4.1 */
/* * Comme CLAIM_DELEGATE_CUR, mais objet identifié par bride de fichier courante. */
        CLAIM_DELEG_CUR_FH      = 5,          /* nouveau pour v4.1 */
/* * Comme CLAIM_DELEGATE_PREV, mais objet identifié par bride de fichier courante. */
        CLAIM_DELEG_PREV_FH     = 6           /* nouveau pour v4.1 */
};

struct open_claim_delegate_cur4 {
      stateid4          delegate_stateid;
      component4      file;
};

union open_claim4 switch (open_claim_type4 claim) {
/* * Pas de droits spéciaux sur le fichier. OPEN ordinaire du fichier spécifié. */
  cas CLAIM_NULL :  component4      file;          /* CURRENT_FH : répertoire */
/* * Droit sur le fichier établi par un open précédent le réamorçage du serveur. Fichier identifié par la bride de fichier
        obtenue à ce moment plutôt que par nom. */
  cas CLAIM_PREVIOUS : open_delegation_type4   delegate_type;    /* CURRENT_FH : fichier réclamé */
/* * Droit sur le fichier fondé sur une délégation accordée par le serveur. Le fichier est spécifié par nom. */
  cas CLAIM_DELEGATE_CUR :  open_claim_delegate_cur4    delegate_cur_info;     /* CURRENT_FH : répertoire */
/* * Droit sur le fichier fondé sur une délégation accordée à une précédente instance de réamorçage du client. Le fichier est
        spécifié par nom. */
  cas CLAIM_DELEGATE_PREV :  component4      file_delegate_prev;         /* CURRENT_FH: répertoire */
/* * Comme CLAIM_NULL. Pas de droits spéciaux sur le fichier. OPEN ordinaire du fichier spécifié par la bride de fichier
        courante. */
  cas CLAIM_FH : vide                    /* nouveau pour v4.1 */  /* CURRENT_FH : fichier régulier à ouvrir */
/* * Comme CLAIM_DELEGATE_PREV. Droit sur le fichier fondé sur une délégation OPEN accordée à une précédente
        instance de réamorçage du client. Le fichier est identifié par la bride de fichier. */
  cas CLAIM_DELEG_PREV_FH : vide       /* nouveau to v4.1 */ /* CURRENT_FH: fichier à ouvrir */
/* * Comme CLAIM_DELEGATE_CUR. Droit sur le fichier fondé sur une délégation accordée par le serveur. Le fichier
        est identifié par la bride de fichier. */
  cas CLAIM_DELEG_CUR_FH :  stateid4       oc_delegate_stateid;   /* nouveau pour v4.1 */
        /* CURRENT_FH : fichier à ouvrir */
};

/* * OPEN : ouvre un fichier, potentiellement recevant une délégation OPEN */
struct OPEN4args {
      seqid4          seqid;
      uint32_t        share_access;
      uint32_t        share_deny;
      open_owner4   owner;
      openflag4       openhow;
```

```
    open_claim4    claim;
};
```

18.16.2 Résultats

```
struct open_read_delegation4 {
   stateid4  stateid;              /* identifiant d'état pour délégation */
   bool      recall;               /* fanion pré-rappel pour délégations obtenues par réclamation (CLAIM_PREVIOUS) */
   nfsace4   permissions;          /* définit des utilisateurs qui n'ont pas besoin d'un ACCESS pour ouvrir en lecture */
};

struct open_write_delegation4 {
   stateid4 stateid;               /* identifiant d'état pour délégation */
   bool     recall;                /* fanion pré-rappel pour délégations obtenues par réclamation (CLAIM_PREVIOUS) */
   nfs_space_limit4    space_limit; /* Définit des conditions que le client doit vérifier pour déterminer si le fichier doit être
purgé sur le serveur à sa clôture.  */
   nfsace4   permissions;          /* Définit des utilisateurs qui n'ont pas besoin d'un ACCESS au titre d'une ouverture
                                      déléguée. */
};

enum why_non_delegation4 {                                  /* nouveau pour v4.1 */
      WND4_NOT_WANTED                    = 0,
      WND4_CONTENTION                    = 1,
      WND4_RESOURCE                      = 2,
      WND4_NOT_SUPP_FTYPE                = 3,
      WND4_WRITE_DELEG_NOT_SUPP_FTYPE = 4,
      WND4_NOT_SUPP_UPGRADE              = 5,
      WND4_NOT_SUPP_DOWNGRADE            = 6,
      WND4_CANCELLED                     = 7,
      WND4_IS_DIR                        = 8
};

union open_none_delegation4                                 /* nouveau pour v4.1 */
switch (why_non_delegation4 ond_why) {
      cas WND4_CONTENTION : bool ond_server_will_push_deleg;
      cas WND4_RESOURCE : bool ond_server_will_signal_avail;
      défaut : vide
};

union open_delegation4
switch (open_delegation_type4 delegation_type) {
      cas OPEN_DELEGATE_NONE : vide
      cas OPEN_DELEGATE_READ : open_read_delegation4 read;
      cas OPEN_DELEGATE_WRITE : open_write_delegation4 write;
      cas OPEN_DELEGATE_NONE_EXT :                          /* nouveau pour v4.1 */
            open_none_delegation4 od_whynone;
};

/* * Fanions de résultat */

/* Le client doit confirmer l'ouverture */
  const OPEN4_RESULT_CONFIRM      = 0x00000002;
/* Type de comportement de verrouillage de fichier chez le serveur */
  const OPEN4_RESULT_LOCKTYPE_POSIX = 0x00000004;
/* Le serveur va préserver le fichier si il est supprimé alors qu'il est ouvert */
  const OPEN4_RESULT_PRESERVE_UNLINKED = 0x00000008;

/* * Le serveur peut utiliser CB_NOTIFY_LOCK sur les verrous déduits de cette ouverture */
  const OPEN4_RESULT_MAY_NOTIFY_LOCK = 0x00000020;
```

```
  struct OPEN4resok {
   stateid4          stateid;                    /* identifiant d'état pour ouverture */
   change_info4   cinfo;                         /* informations de changement de répertoire */
   uint32_t          rflags;                     /* fanions de résultat*/
   bitmap4           attrset;                    /* attribut établi pour une création*/
   open_delegation4 delegation;                  /* Informations sur toute délégation ouverte */
  };

union OPEN4res switch (nfsstat4 status) {
   cas NFS4_OK :   OPEN4resok      resok4;              /* nouvelle CURRENT_FH : fichier ouvert */
   défaut : vide
};
```

18.16.3 Description

L'opération OPEN ouvre un fichier régulier dans un répertoire avec le nom ou bride de fichier fourni. OPEN peut aussi créer un fichier si un nom est fourni, et si le client spécifie qu'il veut créer un fichier. La spécification de si un fichier est à créer ou non, et la méthode de création sont via le paramètre openhow. Le paramètre openhow consiste en une union commutée (type de données opengflag4) qui commute sur la valeur de opentype (OPEN4_NOCREATE ou OPEN4_CREATE). Si OPEN4_CREATE est spécifié, cela conduit à une autre union commutée (type de données createhow4) qui prend en charge quatre cas de méthodes de création : UNCHECKED4, GUARDED4, EXCLUSIVE4, ou EXCLUSIVE4_1. Si opentype est OPEN4_CREATE, alors le champ claim du champ de la réclamation DOIT être un de CLAIM_NULL, CLAIM_DELEGATE_CUR, ou CLAIM_DELEGATE_PREV, parce que ces méthodes de réclamation incluent un composant d'un nom de fichier.

En cas de succès (qui pourrait entraîner la création d'un nouveau fichier) la bride de fichier courante est remplacée par celle de l'objet créé ou existant.

Si la bride de fichier courante est un répertoire des attributs nommés, OPEN va alors créer ou ouvrir un attribut nommé file. Noter que la création exclusive d'un attribut nommé n'est pas prise en charge. Si le createmode est EXCLUSIVE4 ou EXCLUSIVE4_1 et si la bride de fichier courante est un répertoire des attributs nommés, le serveur va retourner EINVAL.

UNCHECKED4 signifie que le fichier devrait être créé si un fichier de ce nom n'existe pas et que rencontrer un fichier régulier existant de ce nom n'est pas une erreur. Pour ce type de création, createattrs spécifie l'ensemble initial d'attributs pour le fichier. L'ensemble d'attributs peut inclure tout attribut écrivable valide pour les fichiers réguliers. Quand une création UNCHECKED4 rencontre un fichier existant, les attributs spécifiés par createattrs ne sont pas utilisés, sauf que quand createattrs spécifie l'attribut de taille avec une taille de zéro, le fichier existant est tronqué.

Si GUARDED4 est spécifié, le serveur vérifie la présence d'un objet dupliqué par nom avant d'effectuer la création. Si un dupliqué existe, NFS4ERR_EXIST est retourné. Si l'objet n'existe pas, la demande est effectuée comme décrit pour UNCHECKED4.

Pour les cas de UNCHECKED4 et GUARDED4, lorsque l'opération est réussie, le serveur va retourner au client un gabarit d'attribut signifiant quels attributs ont réussi à être établis pour l'objet.

EXCLUSIVE4_1 et EXCLUSIVE4 spécifient que le serveur va suivre la sémantique de création exclusive, en utilisant le vérificateur pour s'assurer de la création exclusive de la cible. Le serveur devrait vérifier la présence d'un objet dupliqué par nom. Si l'objet n'existe pas, le serveur crée l'objet et mémorise le vérificateur avec l'objet. Si l'objet existe et si le vérificateur mémorisé correspond au vérificateur fourni par le client, le serveur utilise l'objet existant comme objet nouvellement créé. Si le vérificateur mémorisé ne correspond pas, alors une erreur de NFS4ERR_EXIST est retournée.

Si on utilise EXCLUSIVE4, et si le serveur utilise des attributs pour mémoriser le vérificateur de création exclusive, le serveur va signifier quels attributs il a utilisé en établissant les bits appropriés dans le gabarit d'attribut qui est retourné dans le résultat. À la différence de UNCHECKED4, GUARDED4, et EXCLUSIVE4_1, EXCLUSIVE4 ne prend pas en charge le réglage des attributs à la création du fichier, et après un OPEN réussi via EXCLUSIVE4, le client DOIT envoyer un SETATTR pour régler les attributs à un état connu.

Dans NFSv4.1, EXCLUSIVE4 a été déconseillé en faveur de EXCLUSIVE4_1. À la différence de EXCLUSIVE4, des attributs peuvent être fournis dans le cas de EXCLUSIVE4_1, mais parce que le serveur peut utiliser les attributs de l'objet cible pour mémoriser le vérificateur, l'ensemble des attributs admissibles peut être moindre que l'ensemble d'attributs que

SETATTR permet. Les attributs admissibles pour EXCLUSIVE4_1 sont indiqués dans l'attribut suppattr_exclcreat (paragraphe 5.8.1.14). Si le client tente d'établir dans cva_attrs un attribut qui n'est pas dans suppattr_exclcreat, le serveur DOIT retourner NFS4ERR_INVAL. Le champ de réponse, attrset, indique à la fois quels attributs le serveur a établi à partir de cva_attrs et quels attributs le serveur a utilisé pour mémoriser le vérificateur. Comme décrit au paragraphe 18.16.4, le client peut comparer cva_attrs.attrmask avec attrset pour déterminer quels attributs ont été utilisés pour mémoriser le vérificateur.

Avec l'ajout des sessions persistantes et de pNFS, dans certaines conditions EXCLUSIVE4 NE DOIT PAS être utilisé par le client ou pris en charge par le serveur. Le tableau qui suit résume les méthodes appropriées et de création exclusive obligatoire pour les mises en œuvre de NFSv4.1 :

Antémémoire persistante de réponse activée	Prise en charge de pNFS du serveur	EXIGÉ du serveur	Permis au client
non	non	EXCLUSIVE4_1 et EXCLUSIVE4	EXCLUSIVE4_1 (DEVRAIT) ou EXCLUSIVE4 (NE DEVRAIT PAS)
non	oui	EXCLUSIVE4_1	EXCLUSIVE4_1
oui	non	GUARDED4	GUARDED4
oui	oui	GUARDED4	GUARDED4

Tableau 18 : Méthodes exigées pour Exclusive Create

Si CREATE_SESSION4_FLAG_PERSIST est établi dans le résultat de CREATE_SESSION, la mémoire tampon de réponse est persistante (voir le paragraphe 18.36). Si le fanion EXCHGID4_FLAG_USE_PNFS_MDS est établi dans le résultat de EXCHANGE_ID, le serveur est un serveur pNFS (voir le paragraphe 18.35). Si le client tente d'utiliser EXCLUSIVE4 sur une session persistante, ou une session déduite d'un identifiant de client EXCHGID4_FLAG_USE_PNFS_MDS, le serveur DOIT retourner NFS4ERR_INVAL.

Avec des sessions persistantes, la sémantique de création exclusive est pleinement réalisable via GUARDED4, et donc EXCLUSIVE4 ou EXCLUSIVE4_1 NE DOIT PAS être utilisé. Quand pNFS est utilisé, l'attribut layout_hint pourrait ne pas être pris en charge après la création du fichier. Seules les méthodes EXCLUSIVE4_1 et GUARDED de création exclusive de fichier permettent le réglage atomique des attributs.

Pour le répertoire cible, le serveur retourne les informations change_info4 dans cinfo. Avec le champ atomique de type de données change_info4, le serveur va indiquer si les attributs de changement avant et après ont été obtenus atomiquement par rapport à la création de lien.

L'opération OPEN fournit la capacité de réservation partagée Windows avec l'utilisation des champs share_access et share_deny des arguments OPEN. Le client spécifie à OPEN les modes exigés share_access et share_deny. Pour les clients qui ne prennent pas directement en charge les SHARE (c'est-à-dire, UNIX) la valeur de refus attendue est OPEN4_SHARE_DENY_NONE. Dans le cas où existe une réservation SHARE en conflit avec la demande OPEN, le serveur retourne l'erreur NFS4ERR_SHARE_DENIED. Pour plus de discussion sur la sémantique de SHARE, voir le paragraphe 9.7.

Pour chaque OPEN, le client fournit une valeur pour le champ propriétaire de l'argument OPEN. Le champ propriétaire est du type de données open_owner4, et contient un champ appelé clientid et un champ appelé owner. Le client peut régler le champ clientid à toute valeur et le serveur DOIT l'ignorer. À la place, le serveur DOIT déduire l'identifiant de client de l'identifiant de session de l'opération SEQUENCE de la demande COMPOUND.

Le champ "seqid" de la demande n'est pas utilisé dans NFSv4.1, mais il PEUT être n'importe quelle valeur et le serveur DOIT l'ignorer.

Dans le cas où le client récupère l'état d'une défaillance d'un serveur, le champ claim de l'argument OPEN est utilisé pour signifier que la demande est destinée à réclamer l'état détenu précédemment.

Le champ "claim" de l'argument OPEN est utilisé pour spécifier le fichier à ouvrir et les informations d'état que le client prétend posséder. Il y a sept types de réclamation qui sont les suivants :

Type d'ouverture	Description
CLAIM_NULL, CLAIM_FH	Pour le client, c'est une nouvelle demande OPEN et il n'y a pas d'état précédent associé au fichier pour le client. Avec CLAIM_NULL, le fichier est identifié par la bride de fichier courante et le nom de composant spécifié. Avec CLAIM_FH (nouveau pour NFSv4.1) le

	fichier est juste identifié par la bride de fichier courante.
CLAIM_PREVIOUS	Le client réclame l'état de base OPEN pour un fichier qui était détenu avant un redémarrage de serveur. Généralement utilisé quand un serveur retourne des brides de fichier persistantes ; le client peut ne pas avoir le nom de fichier pour réclamer le OPEN.
CLAIM_DELEGATE_CUR, CLAIM_DELEG_CUR_FH	Le client réclame une délégation pour OPEN telle qu'accordée par le serveur. Généralement, c'est fait au titre du rappel d'une délégation. Avec CLAIM_DELEGATE_CUR, le fichier est identifié par la bride de fichier courante et le nom de composant spécifié. Avec CLAIM_DELEG_CUR_FH (nouveau pour NFSv4.1) le fichier est juste identifié par la bride de fichier courante.
CLAIM_DELEGATE_PREV, CLAIM_DELEG_PREV_FH	Le client réclame une délégation accordée à une précédente instance de client ; utilisé après que le client redémarre. Le serveur PEUT prendre en charge CLAIM_DELEGATE_PREV et/ou CLAIM_DELEG_PREV_FH (nouveau pour NFSv4.1). Si il ne prend en charge aucun des types de réclamation, CREATE_SESSION NE DOIT PAS supprimer l'état de délégation du client, et le serveur DOIT prendre en charge l'opération DELEGPURGE.

Tableau 19

Pour les demandes OPEN qui atteignent le serveur durant la période de grâce, le serveur retourne une erreur de NFS4ERR_GRACE. Les types de réclamation suivants sont des exceptions :

* les demandes OPEN spécifiant le type de réclamation CLAIM_PREVIOUS sont dédiées à réclamer des ouvertures après un redémarrage de serveur et sont normalement seulement valides durant la période de grâce.

* les demandes OPEN spécifiant les types de réclamation CLAIM_DELEGATE_CUR et CLAIM_DELEG_CUR_FH sont valides à la fois durant et après la période de grâce. Comme l'accord de la délégation à laquelle elles sont subordonnées assure qu'il n'y a pas de conflit avec les verrous à réclamer par d'autres clients, le serveur n'a pas besoin de retourner NFS4ERR_GRACE quand elles sont reçues durant la période de grâce.

Pour toute demande OPEN, le serveur peut retourner une délégation OPEN, qui permet que plus d'ouvertures et clôtures soient traitées localement chez le client comme décrit au paragraphe 10.4. Noter que la décision de délégation appartient au serveur. Le client ne devrait jamais supposer que la délégation va être accordée ou non dans une instance particulière. Il devrait toujours être prêt pour l'un ou l'autre cas. Une exception partielle est le cas de la réclamation (CLAIM_PREVIOUS) dans lequel un type de délégation est réclamé. Dans ce cas, la délégation va toujours être accordée, bien que le serveur puisse spécifier un rappel immédiat dans la structure de délégation.

Les rflags retournés par un OPEN réussi permettent au serveur de retourner les informations qui gouvernent comment le fichier ouvert est à traiter :
* OPEN4_RESULT_CONFIRM est déconseillé et NE DOIT PAS être retourné par un serveur NFSv4.1.
* OPEN4_RESULT_LOCKTYPE_POSIX indique que le comportement de verrouillage de gamme d'octets du serveur prend en charge l'ensemble complet de techniques de verrouillage de POSIX [IEEE-1003.1]. À partir de là, le client peut choisir de gérer l'état de verrouillage de gamme d'octets de façon à traiter une discordance de gestion de verrouillage de gamme d'octets.
* OPEN4_RESULT_PRESERVE_UNLINKED indique que le serveur va préserver le fichier ouvert si le client (ou tout autre client) supprime le fichier tant qu'il est ouvert. De plus, le serveur promet de préserver le fichier à travers la période de grâce après le redémarrage de serveur, donnant par là au client l'opportunité de réclamer son ouverture.
* OPEN4_RESULT_MAY_NOTIFY_LOCK indique que le serveur peut tenter CB_NOTIFY_LOCK rappels pour les verrous sur ce fichier. Ce fanion est seulement un conseil, et peut être ignoré en toute sécurité par le client.

Si le composant est de longueur zéro, NFS4ERR_INVAL va être retourné. Le composant est aussi soumis aux vérifications normales UTF-8, de prise en charge de caractères, et de nom. Voir au paragraphe 14.5 pour les détails.

Quand un OPEN est fait et que le propriétaire d'ouverture spécifié a déjà la bride de fichier résultante ouverte, le résultat est de "OUixer" ensemble le nouvel état de partage et refus avec l'état existant. Dans ce cas, seulement un CLOSE doit être fait, même si plusieurs OPEN ont été réalisés. Quand un tel OPEN est fait, vérifier le partage de réservation pour le nouvel OPEN se fait normalement, sans exception pour le OPEN existant détenu par le même propriétaire d'ouverture. Dans ce cas, l'identifiant d'état retourné comme un champ "autre" qui correspond à celui du précédent open alors que le champ "seqid" est incrémenté pour refléter le changement d'état dû à la nouvelle ouverture.

Si le système de fichiers sous-jacent chez le serveur est seulement accessible en mode lecture seule et si la demande OPEN a spécifié ACCESS_WRITE ou ACCESS_BOTH, le serveur va retourner NFS4ERR_ROFS pour indiquer un système de fichiers en lecture seule.

Comme avec l'opération CREATE, le serveur DOIT déduire le propriétaire, ACE de propriétaire, groupe, ou ACE de groupe si un des quatre attributs est exigé et pris en charge par le système de fichiers du serveur. Pour un OPEN avec le mode de création EXCLUSIVE4, le serveur n'a pas de choix, car ces appels OPEN n'incluent pas de champ createattrs. À l'inverse, si createattrs (UNCHECKED4 ou GUARDED4) ou cva_attrs (EXCLUSIVE4_1) est spécifié, et inclut un propriétaire, owner_group, ou ACE pour lequel le principal dans les accréditifs d'appel RPC n'a pas l'autorisation de créer des fichiers, alors le serveur peut retourner NFS4ERR_PERM.

Dans le cas d'un OPEN qui spécifie une taille de zéro (par exemple, troncature) et où le fichier a des attributs nommés, les attributs nommés sont laissés tels qu'ils sont et ne sont pas supprimés.

NFSv4.1 donne un contrôle plus précis aux clients sur l'acquisition de délégations via les nouveaux fanions suivants pour le champ share_access de OPEN4args :
OPEN4_SHARE_ACCESS_WANT_READ_DELEG
OPEN4_SHARE_ACCESS_WANT_WRITE_DELEG
OPEN4_SHARE_ACCESS_WANT_ANY_DELEG
OPEN4_SHARE_ACCESS_WANT_NO_DELEG
OPEN4_SHARE_ACCESS_WANT_CANCEL
OPEN4_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_RESRC_AVAIL
OPEN4_SHARE_ACCESS_WANT_PUSH_DELEG_WHEN_UNCONTENDED

Si (share_access & OPEN4_SHARE_ACCESS_WANT_DELEG_MASK) ne sont pas à zéro, alors le client devra spécifier un seul de :
OPEN4_SHARE_ACCESS_WANT_READ_DELEG
OPEN4_SHARE_ACCESS_WANT_WRITE_DELEG
OPEN4_SHARE_ACCESS_WANT_ANY_DELEG
OPEN4_SHARE_ACCESS_WANT_NO_DELEG
OPEN4_SHARE_ACCESS_WANT_CANCEL

Autrement, le client n'indique ni un désir ni un non désir d'une délégation, et le serveur PEUT ou non retourner une délégation dans la réponse OPEN.

Si le serveur prend en charge les nouveaux fanions _WANT_ et si le client envoie un ou plusieurs des nouveaux fanions, alors au cas où le serveur ne retourne pas une délégation, il DOIT retourner un type de délégation de OPEN_DELEGATE_NONE_EXT. Le champ ond_why dans la réponse indique pourquoi aucune délégation n'a été retournée et va être une de :
WND4_NOT_WANTED : le client a spécifié OPEN4_SHARE_ACCESS_WANT_NO_DELEG.
WND4_CONTENTION : il y a une délégation ou ouverture en conflit sur le fichier.
WND4_RESOURCE : des limitations de ressources empêchent le serveur d'accorder une délégation.
WND4_NOT_SUPP_FTYPE : le serveur ne prend pas en charge les délégations sur ce type de fichier.
WND4_WRITE_DELEG_NOT_SUPP_FTYPE : le serveur ne prend pas en charge les délégations OPEN_DELEGATE_WRITE sur ce type de fichier.
WND4_NOT_SUPP_UPGRADE : le serveur ne prend pas en charge la mise à niveau atomique d'une délégation OPEN_DELEGATE_READ à une délégation OPEN_DELEGATE_WRITE.
WND4_NOT_SUPP_DOWNGRADE : le serveur ne prend pas en charge la dégradation atomique d'une délégation OPEN_DELEGATE_WRITE à une délégation OPEN_DELEGATE_READ.
WND4_CANCELED : le client a spécifié OPEN4_SHARE_ACCESS_WANT_CANCEL et maintenant tout "want" pour cet objet de fichier est annulé.
WND4_IS_DIR : l'objet de fichier spécifié est un répertoire, et l'opération est OPEN ou WANT_DELEGATION, qui ne prend pas en charge les délégations sur des répertoires.

OPEN4_SHARE_ACCESS_WANT_READ_DELEG, OPEN_SHARE_ACCESS_WANT_WRITE_DELEG, ou OPEN_SHARE_ACCESS_WANT_ANY_DELEG signifient, respectivement, que le client veut une OPEN_DELEGATE_READ, OPEN_DELEGATE_WRITE, ou toute délégation sans considération du fanion OPEN4_SHARE_ACCESS_READ, OPEN4_SHARE_ACCESS_WRITE, ou OPEN4_SHARE_ACCESS_BOTH qui est établi. Si le client a une délégation OPEN_DELEGATE_READ sur un fichier et demande une délégation OPEN_DELEGATE_WRITE, alors le client demande une mise à niveau atomique de sa délégation OPEN_DELEGATE_READ en une délégation OPEN_DELEGATE_WRITE. Si le client a une délégation OPEN_DELEGATE_WRITE sur un fichier et demande une délégation OPEN_DELEGATE_READ, alors le client demande une dégradation atomique d'une délégation OPEN_DELEGATE_READ. Un serveur PEUT prendre en charge la mise à niveau ou dégradation atomique. Si il le fait, alors le type de délégation retourné de OPEN_DELEGATE_READ ou

OPEN_DELEGATE_WRITE qui est différent du type de délégation qu'a actuellement le client, indique la réussite de la mise à niveau ou dégradation. Si le serveur ne prend pas en charge la mise à niveau ou dégradation atomique de délégation, alors ond_why va être réglé à WND4_NOT_SUPP_UPGRADE ou WND4_NOT_SUPP_DOWNGRADE.

OPEN4_SHARE_ACCESS_WANT_NO_DELEG signifie que le client ne veut aucune délégation.

OPEN4_SHARE_ACCESS_WANT_CANCEL signifie que le client ne veut aucune délégation et veut annuler tout "want" précédemment enregistré pour une délégation.

Le client peut établir OPEN4_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_RESRC_AVAIL et/ou OPEN4_SHARE_ACCESS_WANT_PUSH_DELEG_WHEN_UNCONTENDED. Cependant, ils n'auront d'effet que si un des suivants est établi :

* OPEN4_SHARE_ACCESS_WANT_READ_DELEG
* OPEN4_SHARE_ACCESS_WANT_WRITE_DELEG
* OPEN4_SHARE_ACCESS_WANT_ANY_DELEG

Si le client spécifie OPEN4_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_RESRC_AVAIL, alors il souhaite enregistrer un "want" pour une délégation, dans le cas où le résultat de OPEN n'inclurait pas de délégation. Si c'est le cas et si le serveur refuse la délégation à cause de ressources insuffisantes, le serveur PEUT plus tard informer le client, via l'opération CB_RECALLABLE_OBJ_AVAIL, que la condition de limitation de ressource a cessé. Le serveur va dire au client qu'il a l'intention d'envoyer une future opération CB_RECALLABLE_OBJ_AVAIL en établissant le type de délégation dans le résultat à OPEN_DELEGATE_NONE_EXT, ond_why à WND4_RESOURCE, et ond_server_will_signal_avail réglé à VRAI. Si ond_server_will_signal_avail est réglé à VRAI, le serveur DOIT ensuite envoyer une opération CB_RECALLABLE_OBJ_AVAIL.

Si le client spécifie OPEN4_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_UNCONTENDED, alors il souhaite enregistrer un "want" pour une délégation, dans le cas où le résultat de OPEN n'inclurait pas de délégation. Si c'est le cas et si le serveur refuse la délégation à cause de contention, le serveur PEUT ensuite informer le client, via l'opération CB_PUSH_DELEG, que la condition de contention a cessé. Le serveur va dire au client qu'il a l'intention d'envoyer une future opération CB_PUSH_DELEG en réglant le type de délégation dans le résultat à OPEN_DELEGATE_NONE_EXT, ond_why à WND4_CONTENTION, et ond_server_will_push_deleg à VRAI. Si ond_server_will_push_deleg est VRAI, le serveur DOIT ensuite envoyer une opération CB_PUSH_DELEG.

Si le client a précédemment enregistré un want pour une délégation sur un fichier, et ensuite envoie une demande d'enregistrer un want pour une délégation sur le même fichier, le serveur DOIT retourner une nouvelle erreur : NFS4ERR_DELEG_ALREADY_WANTED. Si le client souhaite enregistrer un type différent de want délégation pour le même fichier, il DOIT annuler la délégation WANT existante.

18.16.4 Mise en œuvre

En l'absence d'une session persistante, le client invoque une création exclusive en réglant le paramètre how à EXCLUSIVE4 ou EXCLUSIVE4_1. Dans ce cas, le client fournit un vérificateur qui peut être raisonnablement attendu être unique. Une combinaison d'un identifiant de client, peut-être l'adresse réseau du client, et d'un numéro unique généré par le client, peut-être l'identifiant de transaction RPC, peut être appropriée.

Si l'objet n'existe pas, le serveur crée l'objet et mémorise le vérificateur dans une mémorisation stable. Pour les systèmes de fichiers qui ne fournissent pas un mécanisme pour la mémorisation d'attributs de fichier arbitraires, le serveur peut utiliser un ou plusieurs éléments des métadonnées de l'objet pour mémoriser le vérificateur. Le vérificateur DOIT être mémorisé dans une mémorisation stable pour empêcher des erreurs ou défaillances à la retransmission de la demande. Il est supposé qu'une création exclusive est effectuée parce que des sémantiques exclusives sont critiques pour l'application. À cause de l'utilisation attendue, un CREATE exclusif ne s'appuie pas seulement sur la mémoire tampon de réponse du serveur pour la mémorisation du vérificateur. Une mémoire tampon de réponse non persistante ne survit pas à une défaillance et la session et la mémoire tampon de réponse peuvent être supprimées après une partition de réseau qui excède la durée de prêt, ouvrant donc des fenêtres de défaillance.

Un serveur NFSv4.1 NE DEVRAIT PAS mémoriser le vérificateur dans des attributs RECOMMANDÉS ou EXIGÉS du fichier. Si il le fait, le serveur DEVRAIT utiliser time_modify_set ou time_access_set pour mémoriser le vérificateur. Le serveur NE DEVRAIT PAS mémoriser le vérificateur dans les attributs suivants :

acl (il est souhaitable pour le contrôle d'accès qu'il soit établi à la création),
dacl (idem),

mode (idem),
owner (idem),
owner_group (idem),
retentevt_set (il peut être souhaitable d'établir la rétention à la création)
retention_hold (idem),
retention_set (idem),
sacl (il est souhaitable pour le contrôle d'examen qu'il soit établi à la création),
size (sur certains serveurs, size peut avoir une gamme limitée de valeurs),
mode_set_masked (comme avec mode),
time_creation (une création de fichier significative devrait être réglée quand le fichier est créé).

Une autre solution pour le serveur est d'utiliser un attribut nommé pour mémoriser le vérificateur.

Parce que la méthode de création EXCLUSIVE4 ne spécifie pas d'attributs initiaux dans le traitement d'une création EXCLUSIVE4, le serveur

* DEVRAIT régler le propriétaire du fichier à celui correspondant à l'accréditif de l'en-tête RPC de la demande.
* NE DEVRAIT PAS laisser le contrôle d'accès du fichier à quelqu'un d'autre que le propriétaire du fichier.

Si le serveur ne peut pas prendre en charge la sémantique de création exclusive, éventuellement à cause de l'exigence d'engager le vérificateur à une mémorisation stable, il devrait faire échouer la demande OPEN avec l'erreur NFS4ERR_NOTSUPP.

Durant une demande CREATE exclusive, si l'objet existe déjà, le serveur reconstruit le vérificateur de l'objet et le compare au vérificateur dans la demande. Si ils correspondent, le serveur traite la demande comme réussie. La demande est présumée être un dupliqué d'une demande antérieure réussie pour laquelle la réponse a été perdue et que le mécanisme de mise en antémémoire de demande dupliqué du serveur n'a pas détecté. Si les vérificateurs ne correspondent pas, la demande est rejetée avec l'état NFS4ERR_EXIST.

Après que le client a effectué une création exclusive réussie, la réponse attrset indique quels attributs ont été utilisés pour mémoriser le vérificateur. Si EXCLUSIVE4 a été utilisé, les attributs réglés dans attrset ont été utilisés pour le vérificateur. Si EXCLUSIVE4_1 a été utilisé, le client détermine les attributs utilisés pour le vérificateur en comparant attrset avec cva_attrs.attrmask ; tous les bits établis dans le premier mais pas dans le dernier identifient les attributs utilisés pour mémoriser le vérificateur. Le client DOIT immédiatement envoyer un SETATTR pour régler les attributs utilisés pour mémoriser le vérificateur. Jusqu'à ce qu'il le fasse, les attributs utilisés pour mémoriser le vérificateur ne peuvent pas être considérés comme fiables. Le SETATTR suivant NE DOIT PAS survenir dans la même demande COMPOUND que le OPEN.

Sauf si une session persistante est utilisée, l'utilisation de l'attribut GUARDED4 ne fournit pas la sémantique de exactement une fois. En particulier, si une réponse est perdue et si le serveur ne détecte pas la retransmission de la demande, l'opération peut échouer avec NFS4ERR_EXIST, même si la création a été effectuée avec succès. Le client utiliserait ce comportement dans le cas où l'application n'a pas demandé une création exclusive mais a demandé que le fichier soit tronqué quand il a été ouvert. Dans le cas du client arrivant en fin de temporisation et qui retransmet la demande de création, le client peut utiliser GUARDED4 pour empêcher qu'une séquence comme création, écriture, création (retransmise) se produise.

Pour les réservations SHARE, la valeur de l'expression (share_access & ~OPEN4_SHARE_ACCESS_WANT_DELEG_MASK) DOIT être une de OPEN4_SHARE_ACCESS_READ, OPEN4_SHARE_ACCESS_WRITE, ou OPEN4_SHARE_ACCESS_BOTH. Sinon, le serveur DOIT retourner NFS4ERR_INVAL. La valeur de share_deny DOIT être une de OPEN4_SHARE_DENY_NONE, OPEN4_SHARE_DENY_READ, OPEN4_SHARE_DENY_WRITE, ou OPEN4_SHARE_DENY_BOTH. Sinon, le serveur DOIT retourner NFS4ERR_INVAL.

Sur la base de la valeur de share_access (OPEN4_SHARE_ACCESS_READ, OPEN4_SHARE_ACCESS_WRITE, ou OPEN4_SHARE_ACCESS_BOTH) le client devrait vérifier que le demandeur a les droits d'accès appropriés pour effectuer l'opération spécifiée. Cela va généralement être le résultat de l'application des règles d'accès d'ACL au fichier pour le demandeur actuel. Cependant, tout comme avec l'opération ACCESS, le client ne devrait pas tenter de remettre en question les décisions du serveur, car les droits d'accès peuvent changer et peuvent être soumis à des contrôles administratifs du serveur en dehors du cadre de l'ACL. Si l'opération READ ou WRITE du demandeur n'est pas autorisée (selon la valeur de share_access) le serveur DOIT retourner NFS4ERR_ACCESS.

Noter que si l'identifiant de client n'a pas été créé avec la capacité EXCHGID4_FLAG_BIND_PRINC_STATEID établie dans la réponse à EXCHANGE_ID, alors le serveur NE DOIT PAS imposer d'exigence que les READ et WRITE envoyés

pour un fichier ouvert aient les mêmes accréditifs que le OPEN lui-même, et il est EXIGÉ du serveur qu'il effectue une vérification d'accès sur les READ et WRITE eux-mêmes. Autrement, si la réponse à EXCHANGE_ID n'avait pas EXCHGID4_FLAG_BIND_PRINC_STATEID établi, alors avec une exception, les accréditifs utilisés dans la demande OPEN DOIVENT correspondre à ceux utilisés dans les READ et WRITE, et les identifiants d'état dans les READ et WRITE DOIVENT correspondre, ou être déduits de l'identifiant d'état provenant de la réponse à OPEN. L'exception est si la protection d'état SP4_SSV ou SP4_MACH_CRED est utilisée, et si le spo_must_allow résultant de EXCHANGE_ID inclut des opérations READ et/ou WRITE. Dans ce cas, l'accréditif de machine ou de SSV va pouvoir être permis pour envoyer READ et/ou WRITE. Voir au paragraphe 18.35.

Si le composant fourni à OPEN est une liaison symbolique, l'erreur NFS4ERR_SYMLINK va être retournée au client, tandis que si c'est un répertoire, l'erreur NFS4ERR_ISDIR va être retournée. Si le composant n'est aucun de ceux-là mais pas un fichier ordinaire, l'erreur NFS4ERR_WRONG_TYPE est retournée. Si la bride de fichier courante n'est pas un répertoire, l'erreur NFS4ERR_NOTDIR va être retournée.

L'utilisation du fanion de résultat OPEN4_RESULT_PRESERVE_UNLINKED permet à un client d'éviter la pratique courante de mise en œuvre de renommer un fichier ouvert en ".nfs<unique valeur>" après qu'il supprime le fichier. Après que le serveur retourne OPEN4_RESULT_PRESERVE_UNLINKED, si un client envoie une opération REMOVE qui réduirait le compte de liaison du fichier à zéro, le serveur DEVRAIT rapporter une valeur de zéro pour l'attribut numlinks sur le fichier.

Si un autre client a une délégation du fichier ouvert qui est en conflit avec l'ouverture faite (parfois cela dépend de la valeur share_access ou share_deny spécifiée) la ou les délégations DOIVENT être rappelées, et l'opération ne peut pas avoir lieu jusqu'à ce que chacune de ces délégations soit retournée ou révoquée. Sauf lorsque cela arrive très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites pendant que la délégation reste en instance. Dans le cas d'une délégation OPEN_DELEGATE_WRITE, toute ouverture par un client différent va être en conflit, tandis que pour une délégation OPEN_DELEGATE_READ, seule les ouvertures avec une des caractéristiques suivantes vont être considérées comme étant en conflit :

* La valeur de share_access inclut le bit OPEN4_SHARE_ACCESS_WRITE.
* La valeur de share_deny spécifie OPEN4_SHARE_DENY_READ ou OPEN4_SHARE_DENY_BOTH.
* OPEN4_CREATE est spécifié avec UNCHECKED4, l'attribut taille est spécifié comme zéro (pour troncature) et un fichier existant est tronqué.

Si OPEN4_CREATE est spécifié et si le fichier n'existe pas et la bride de fichier courante désigne un répertoire pour lequel un autre client détient une délégation de répertoire, alors, sauf si la délégation est telle que la situation peut se résoudre par l'envoi d'une notification, la délégation DOIT être rappelée, et l'opération ne peut pas se faire avant que la délégation soit retournée ou révoquée. Sauf lorsque cela arrive très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites pendant que la délégation reste en instance.

Si OPEN4_CREATE est spécifié et si le fichier n'existe pas et si la bride de fichier courante désigne un répertoire pour lequel une ou plusieurs délégations de répertoire existent, alors, quand ces délégations demandent de telles notifications, NOTIFY4_ADD_ENTRY va être généré par suite de cette opération.

18.16.4.1 Avis pour les mises en œuvre de client

OPEN ressemble à LOOKUP en ce qu'il génère une bride de fichier à utiliser par le client. À la différence de LOOKUP cependant, OPEN crée un état de serveur sur la bride de fichier. Dans des circonstances normales, le client peut seulement libérer cet état avec une opération CLOSE. CLOSE utilise la bride de fichier courante pour déterminer quel fichier clore. Donc, le client DOIT faire suivre chaque opération OPEN d'une opération GETFH dans la même procédure COMPOUND. Cela va fournir au client la bride de fichier telle que CLOSE puisse être utilisé de façon appropriée.

Attendre simplement que le prêt sur le fichier expire est insuffisant parce que le serveur peut maintenir l'état indéfiniment tant qu'un autre client ne tente pas de faire un accès en conflit sur le même fichier. Voir aussi le paragraphe 2.10.6.4.

18.17 Opération 19 : OPENATTR - ouvre le répertoire d'attributs désigné

18.17.1 Arguments

```
struct OPENATTR4args {                          /* CURRENT_FH : objet */
     bool    createdir;
};
```

18.17.2 Résultats

```
struct OPENATTR4res {
/* * Si status est NFS4_OK, nouvelle CURRENT_FH : répertoire des attributs nommés */
        nfsstat4        status;
  };
```

18.17.3 Description

L'opération OPENATTR est utilisée pour obtenir la bride de fichier du répertoire des attributs nommés associé à la bride de fichier courante. Le résultat de OPENATTR va être une bride de fichier à un objet de type NF4ATTRDIR. À partir de cette bride de fichier, les opérations READDIR et LOOKUP peuvent être utilisées pour obtenir des brides de fichier pour les divers attributs nommés associés à l'objet original de système de fichiers. Les brides de fichier retournées dans le répertoire des attributs nommés vont désigner des objets de type NF4NAMEDATTR.

L'argument createdir permet au client de signifier si un répertoire des attributs nommés devrait être créé par suite de l'opération OPENATTR. Certains clients peuvent utiliser l'opération OPENATTR avec une valeur de FAUX pour createdir pour déterminer si des attributs nommés existent pour l'objet. Si il n'en existe pas, alors NFS4ERR_NOENT va être retourné. Si createdir a la valeur de VRAI et qu'aucun répertoire des attributs nommés n'existe, il en est créé un et sa bride de fichier devient la bride de fichier courante. Par ailleurs, si createdir a la valeur de VRAI et si le répertoire des attributs nommés existe déjà, aucune erreur ne résulte et la bride de fichier du répertoire existant devient la bride de fichier courante. La création d'un répertoire des attributs nommés suppose que le serveur a mis en œuvre la prise en charge d'attribut nommé de cette façon et n'est pas obligé de faire ainsi par cette définition.

Si la bride de fichier courante désigne un objet de type NF4NAMEDATTR (un attribut nommé) ou NF4ATTRDIR (un répertoire des attributs nommés) une erreur de NFS4ERR_WRONG_TYPE est retournée au client. Les attributs nommés ou un répertoire des attributs nommés NE DOIVENT PAS avoir leurs propres attributs nommés.

18.17.4 Mise en œuvre

Si le serveur ne prend pas en charge les attributs nommés pour la bride de fichier courante, une erreur de NFS4ERR_NOTSUPP va être retournée au client.

18.18 Opération 21 : OPEN_DOWNGRADE - réduit l'accès en ouverture de fichier

18.18.1 Arguments

```
struct OPEN_DOWNGRADE4args {                    /* CURRENT_FH : fichier ouvert */
      stateid4      open_stateid;
      seqid4        seqid;
      uint32_t      share_access;
      uint32_t      share_deny;
};
```

18.18.2 Résultats

```
struct OPEN_DOWNGRADE4resok {
        stateid4        open_stateid;
};

union OPEN_DOWNGRADE4res switch(nfsstat4 status) {
   cas NFS4_OK : OPEN_DOWNGRADE4resok    resok4;
   défaut : vide
};
```

18.18.3 Description

Cette opération est utilisée pour ajuster les états d'accès et de refus pour une ouverture donnée. C'est nécessaire quand un certain propriétaire d'ouverture ouvre le même fichier plusieurs fois avec des valeurs différentes d'accès et refus. Dans cette situation, une clôture d'une des ouvertures peut changer les fanions appropriés share_access et share_deny pour supprimer les bits associés à des ouvertures qui ne sont plus en vigueur.

Les valeurs valides pour l'expression (share_access & ~OPEN4_SHARE_ACCESS_WANT_DELEG_MASK) sont OPEN4_SHARE_ACCESS_READ, OPEN4_SHARE_ACCESS_WRITE, ou OPEN4_SHARE_ACCESS_BOTH. Si le client spécifie d'autres valeurs, le serveur DOIT répondre avec NFS4ERR_INVAL.

Les valeurs valides pour le champ share_deny sont OPEN4_SHARE_DENY_NONE, OPEN4_SHARE_DENY_READ, OPEN4_SHARE_DENY_WRITE, ou OPEN4_SHARE_DENY_BOTH. Si le client spécifie d'autres valeurs, le serveur DOIT répondre avec NFS4ERR_INVAL.

Après la vérification des valeurs valides de share_access et share_deny, le serveur remplace les modes courants d'accès et refus sur le fichier par share_access et share_deny sous réserve des contraintes suivantes :

* Les bits dans share_access DEVRAIENT être égaux à l'union des bits share_access (non inclus les bits OPEN4_SHARE_WANT_*) spécifiés pour un sous ensemble des OPEN en vigueur pour le propriétaire d'ouverture courant sur le fichier courant.
* Les bits dans share_deny DEVRAIENT être égaux à l'union des bits share_deny spécifiés pour un sous ensemble des OPEN en vigueur pour le propriétaire d'ouverture courant sur le fichier courant.

Si les contraintes ci-dessus ne sont pas respectées, le serveur DEVRAIT retourner l'erreur NFS4ERR_INVAL. Comme les bits share_access et share_deny devraient être des sous ensembles de ceux déjà accordés, à part un défaut de mise en œuvre chez le client ou le serveur, il n'est pas possible que la demande OPEN_DOWNGRADE soit refusée à cause de conflits de partage de réservation.

L'argument seqid n'est pas utilisé dans NFSv4.1, PEUT voir n'importe quelle valeur, et DOIT être ignoré par le serveur.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.18.4 Mise en œuvre

Une opération OPEN_DOWNGRADE peut rendre les délégations OPEN_DELEGATE_READ accordables alors qu'elles ne l'étaient pas précédemment. Les serveurs peuvent choisir de répondre immédiatement si il y a une demande "veut délégation" en instance ou peut répondre plus tard à la situation.

18.19 Opération 22 : PUTFH - établit la bride de fichier courante

18.19.1 Arguments

```
struct PUTFH4args {
        nfs_fh4         object;
};
```

18.19.2 Résultats

```
struct PUTFH4res {
/* * Si status est NFS4_OK, nouvelle CURRENT_FH : argument à PUTFH */
        nfsstat4        status;
};
```

18.19.3 Description

Cette opération remplace la bride de fichier courante par la bride de fichier fournie comme argument. Elle supprime l'identifiant d'état courant.

Si le mécanisme de sécurité utilisé par le demandeur ne satisfait pas les exigences de la bride de fichier fournie à cette

opération, le serveur DOIT retourner NFS4ERR_WRONGSEC. Voir au paragraphe 16.2.3.1.1 les détails sur la bride de fichier courante. Voir au paragraphe 16.2.3.1.2 les détails sur l'identifiant d'état courant.

18.19.4 Mise en œuvre

Cette opération est utilisée dans une demande NFS d'établir le contexte pour les opérations d'accès au fichier qui suivent dans la même demande COMPOUND.

18.20 Opération 23 : PUTPUBFH - établit une bride de fichier publique

18.20.1 Argument

Vide

18.20.2 Résultat

```
struct PUTPUBFH4res {
/* * Si status est NFS4_OK, nouvelle CURRENT_FH : bride de fichier publique */
      nfsstat4       status;
};
```

18.20.3 Description

Cette opération remplace la bride de fichier courante par la bride de fichier qui représente la bride de fichier publique de l'espace de noms du serveur. Cette bride de fichier peut être différente de la bride de fichier "racine" qui peut être associée à un autre répertoire sur le serveur.

PUTPUBFH supprime aussi l'identifiant d'état courant".

La bride de fichier publique représente les concepts incorporés dans la [RFC2054], la [RFC2055], et la [RFC2224]. L'intention pour NFSv4.1 est que la bride de fichier publique (représentée par l'opération PUTPUBFH) soit utilisée comme une méthode pour fournir la compatibilité de serveur WebNFS avec NFSv3.

La bride de fichier publique et la bride de fichier racine (représentée par l'opération PUTROOTFH) DEVRAIENT être équivalentes. Si les brides de fichier publique et racine ne sont pas équivalentes, alors le répertoire correspondant à la bride de fichier publique DOIT être un descendant du répertoire correspondant à la bride de fichier racine. Voir plus de détails au paragraphe 16.2.3.1.1 sur la bride de fichier courante. Voir au paragraphe 16.2.3.1.2 plus de détails sur l'identifiant d'état courant.

18.20.4 Mise en œuvre

Cette opération est utilisée dans une demande NFS pour régler le contexte pour les opérations d'accès au fichier qui suivent dans la même demande COMPOUND.

Avec la bride de fichier publique NFSv3, le client est capable de spécifier si le nom de chemin fourni dans LOOKUP devrait être évalué comme un chemin absolu par rapport à la racine du serveur ou par rapport à la bride de fichier publique. La [RFC2224] contient unz discussion de la fonction. Avec NFSv4.1, ce type de spécification n'est pas directement disponible dans l'opération LOOKUP. La raison de cela est parce que les séparateurs de composants nécessaires pour spécifier absolu et relatif ne sont pas permis dans NFSv4. Donc, le client est responsable de la construction de sa demande de telle façon que l'utilisation de PUTROOTFH ou PUTPUBFH signifie respectivement une évaluation absolue ou relative d'un URL NFS.

Noter qu'il y a des avertissements mentionnés dans la [RFC2224] à l'égard de l'utilisation de l'évaluation absolue et des restrictions que le serveur peut faire sur cette évaluation à l'égard de la quantité d'espace de noms qui est disponible. Ces mêmes avertissements s'appliquent à NFSv4.1. Il est donc probable que à cause de détails de mise en œuvre de serveur, une recherche de bride de fichier publique NFSv3 absolue peut se comporter différemment qu'une résolution NFSv4.1 absolue.

Il y a une forme de négociation de sécurité décrite dans la [RFC2755] qui utilise la bride de fichier publique et une

surcharge du nom de chemin. Cette méthode n'est pas disponible avec NFSv4.1 car les brides de fichier ne sont pas surchargées avec une signification spéciale et donc ne fournissent pas le même cadre que NFSv3. Les clients devraient donc utiliser les mécanismes de négociation de sécurité décrits au paragraphe 2.6.

18.21 Opération 24 : PUTROOTFH - établir la bride de fichier racine

18.21.1 Arguments

Vide

18.21.2 Résultats

```
struct PUTROOTFH4res {                  /* * Si status est NFS4_OK, nouvelle CURRENT_FH : fh racine */
      nfsstat4      status;
};
```

18.21.3 Description

Cette opération remplace la bride de fichier courante par la bride de fichier qui représente la racine de l'espace de noms du serveur. À partir de cette bride de fichier, une opération LOOKUP peut localiser toute autre bride de fichier sur le serveur. Cette bride de fichier peut être différente de la bride de fichier "publique" qui peut être associée à quelque autre répertoire sur le serveur. PUTROOTFH supprime aussi l'identifiant d'état courant. Voir au paragraphe 16.2.3.1.1 les détails sur la bride de fichier courante. Voir au paragraphe 16.2.3.1.2 les détails sur l'identifiant d'état courant.

18.21.4 Mise en œuvre

Cette opération est utilisée dans une demande NFS pour régler le contexte pour les opérations d'accès au fichier qui suivent dans la même demande COMPOUND.

18.22 Opération 25 : READ - lire à partir du fichier

18.22.1 Arguments

```
struct READ4args {                              /* CURRENT_FH : fichier */
      stateid4      stateid;
      offset4       offset;
      count4        count;
};
```

18.22.2 Résultats

```
struct READ4resok {
        bool          eof;
        opaque        data<>;
  };

union READ4res switch (nfsstat4 status) {
   cas NFS4_OK :  READ4resok     resok4;
   défaut :  vide
};
```

18.22.3 Description

L'opération READ lit les données dans le fichier régulier identifié par la bride de fichier courante.

Le client fournit un décalage de où le READ va commencer et un compte de combien d'octets sont à lire. Un décalage de zéro signifie de lire les données depuis le début du fichier. Si le décalage est supérieur ou égal à la taille du fichier, l'état

NFS4_OK est retourné avec une longueur de données réglée à zéro et eof est réglé à VRAI. Le READ est soumis aux vérifications de permissions d'accéder.

Si le client spécifie une valeur de compte de zéro, le READ réussit et retourne zéro octet de données là encore sous réserve des vérifications de permissions d'accéder. Le serveur peut choisir de retourner moins d'octets que spécifié par le client. Le client doit vérifier cette condition et la traiter de façon appropriée.

Sauf quand un identifiant d'état spécial est utilisé, la valeur d'identifiant d'état pour une demande READ représente une valeur retournée d'une demande précédente de verrou de gamme d'octets ou de réservation partagée ou de l'identifiant d'état associé à une délégation. L'identifiant d'état identifie les propriétaires associés si il en est et est utilisé par le serveur pour vérifier que les verrous associés sont encore valides (par exemple, qu'ils n'ont pas été révoqués).

Si la lecture s'arrête à la fin de fichier (formellement, dans une opération READ correctement formée, si décalage + compte est égal à la taille du fichier) ou si l'opération READ s'étend au delà de la taille du fichier (si décalage + compte est supérieur à la taille du fichier) eof est retourné comme VRAI ; autrement, il est FAUX. Un READ réussi d'un fichier vide va toujours retourner eof comme VRAI.

Si la bride de fichier courante n'est pas un fichier ordinaire, une erreur va être retournée au client. Dans le cas où la bride de fichier courante représente un objet de type NF4DIR, NFS4ERR_ISDIR est retourné. Si la bride de fichier courante désigne une liaison symbolique, NFS4ERR_SYMLINK est retourné. Dans tous les autres cas, NFS4ERR_WRONG_TYPE est retourné.

Pour un READ avec une valeur d'identifiant d'état de tous les bits égaux à zéro, le serveur PEUT permettre que le READ soit servi sous réserve des verrous de gamme d'octets obligatoires ou des modes de refus de partage courants pour le fichier. Pour un READ avec une valeur d'identifiant d'état de tous les bits égaux à un, le serveur PEUT permettre que les opérations READ outrepassent les vérifications de verrouillage chez le serveur.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.22.4 Mise en œuvre

Si le serveur retourne une "lecture courte" (c'est-à-dire, moins de données que demandé et eof réglé à FAUX) le client devrait envoyer un autre READ pour obtenir les données restantes. Un serveur peut retourner moins de données que demandé dans plusieurs circonstances. Le fichier peut avoir été tronqué par un autre client ou peut-être sur le serveur lui-même, changeant la taille du fichier par rapport à ce que le client demandeur croit. Cela réduirait la quantité réelle de données disponibles au client. Il est possible que le serveur réduise la taille de transfert et donc retourne un résultat de lecture courte. L'épuisement des ressources du serveur peut aussi se produire dans une lecture courte.

Si le verrouillage de gamme d'octets obligatoire est en vigueur pour le fichier, et si la gamme d'octets correspondant aux données à lire sur le fichier est verrouillée par WRITE_LT par un propriétaire non associé à l'identifiant d'état, le serveur va retourner l'erreur NFS4ERR_LOCKED. Le client devrait essayer d'obtenir le READ_LT approprié via l'opération LOCK avant de réessayer le READ. Quand le READ s'achève, le client devrait libérer le verrou de gamme d'octets via LOCKU.

Si un autre client a une délégation OPEN_DELEGATE_WRITE pour le fichier à lire, la délégation doit être rappelée, et l'opération ne peut pas se faire tant que cette délégation n'est pas retournée ou révoquée. Sauf quand cela se fait très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites pendant que la délégation reste en instance. Normalement, les délégations ne vont pas être rappelées par suite d'une opération READ car le rappel va se produire par suite d'un OPEN antérieur. Cependant, comme il est possible qu'un READ soit fait avec un identifiant d'état spécial, le serveur a besoin de vérifier ce cas même si le client devrait avoir fait un OPEN précédemment.

18.23 Opération 26 : READDIR - lit le répertoire

18.23.1 Arguments

```
struct READDIR4args {                          /* CURRENT_FH: répertoire */
      nfs_cookie4    cookie;
      verifier4      cookieverf;
      count4         dircount;
      count4         maxcount;
```

```
        bitmap4         attr_request;
};
```

18.23.2 Résultats

```
struct entry4 {
        nfs_cookie4     cookie;
        component4      name;
        fattr4          attrs;
        entry4          *nextentry;
};

struct dirlist4 {
        entry4          *entries;
        bool            eof;
};

struct READDIR4resok {
         verifier4      cookieverf;
        dirlist4        réponse;
};

union READDIR4res switch (nfsstat4 status) {
   cas NFS4_OK : READDIR4resok  resok4;
   défaut : vide
};
```

18.23.3 Description

L'opération READDIR restitue un nombre variable d'entrées d'un répertoire de système de fichiers et retourne les attributs demandés par le client pour chaque entrée avec les informations pour permettre au client de demander des entrées de répertoire supplémentaires dans un READDIR suivant.

Les arguments contiennent une valeur de mouchard qui représente où le READDIR devrait commencer dans le répertoire. Une valeur de zéro pour le mouchard est utilisée pour commencer à lire au début du répertoire. Pour les demandes READDIR suivantes, le client spécifie une valeur de mouchard fournie par le serveur sur une demande READDIR précédente.

Le champ cookieverf de la demande devrait être réglé à 0 (zéro) quand le champ cookie de la demande est zéro (première lecture du répertoire). Sur les demandes suivantes, le champ cookieverf doit correspondre au cookieverf retourné par le READDIR dans lequel le mouchard a été acquis. Si le serveur détermine que le cookieverf n'est plus valide pour le répertoire, l'erreur NFS4ERR_NOT_SAME doit être retournée.

Le champ dircount de la demande est une indication du nombre maximum d'octets d'informations de répertoire qui devrait être retourné. Cette valeur représente la longueur totale des noms des entrées de répertoire et la valeur du mouchard pour ces entrées. Cette longueur représente le codage XDR des données (noms et mouchards) et non la longueur du format natif du serveur.

Le champ maxcount de la demande représente la taille maximum totale de toutes les données retournées dans la structure READDIR4resok et inclut les frais généraux de XDR. Le serveur PEUT retourner moins de données. Si le serveur est incapable de retourner une seule entrée de répertoire dans la limite de maxcount, l'erreur NFS4ERR_TOOSMALL DOIT être retourné au client.

Finalement, le champ attr_request de la demande représente la liste des attributs à retourner pour chaque entrée de répertoire fournie par le serveur. Une réponse réussie consiste en une liste d'entrées de répertoire. Chacune de ces entrées contient le nom de l'entrée de répertoire, une valeur de mouchard pour cette entrée, et les attributs associés comme demandé. Le fanion "eof" a une valeur de VRAI si il n'y a plus d'entrées dans le répertoire.

La valeur de mouchard n'a de signification que pour le serveur et est utilisée comme curseur pour l'entrée de répertoire.

Comme mentionné, ce mouchard est utilisé par le client pour les opérations READDIR suivantes afin qu'il puisse continuer de lire un répertoire. Le mouchard est un concept similaire à celui du décalage READ mais NE DOIT PAS être interprété comme tel par le client. Idéalement, la valeur de mouchard NE DEVRAIT PAS changer si le répertoire est modifié parce que le client peut mettre ces valeurs en antémémoire.

Dans certains cas, le serveur peut rencontrer une erreur en obtenant les attributs pour une entrée de répertoire. Au lieu de retourner une erreur pour l'opération READDIR entière, le serveur peut retourner l'attribut rdattr_error (paragraphe 5.8.1.12). Avec cela, le serveur est capable de communiquer la défaillance au client et de ne pas faire échouer l'opération entière au cas où ce pourrait être une défaillance transitoire. Évidemment, le client doit demander l'attribut fattr4_rdattr_error pour que cette méthode fonctionne correctement. Si le client ne demande pas l'attribut, le serveur n'a pas d'autre choix que de retourner un échec pour l'opération READDIR entière.

Pour certains environnements de système de fichiers, les entrées de répertoire "." et ".." ont une signification spéciale, et ne l'ont pas dans d'autres environnements. Si le serveur prend en charge ces entrées spéciales dans un répertoire, elles NE DEVRAIENT PAS être retournées au client au titre de la réponse READDIR. Pour permettre certains environnements de client, les valeurs de mouchard de zéro, 1, et 2 sont considérées comme réservées. Noter que le client UNIX va utiliser ces valeurs quand il combine la réponse du serveur et les représentations locales pour permettre une présentation pleinement formée de répertoire UNIX à l'application.

Pour les arguments READDIR, les valeurs de mouchard de un et deux NE DEVRAIENT PAS être utilisées, et pour les résultats de READDIR, les valeurs de mouchard de zéro, un, et deux NE DEVRAIENT PAS être retournées.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.23.4 Mise en œuvre

Les représentations de répertoire de système de fichiers de serveur peuvent avoir de grandes différences. Les interfaces de programmation d'un client peuvent aussi être liées à l'environnement de fonctionnement local d'une façon qui ne se traduit pas bien dans le protocole NFS. Donc, l'utilisation des champs dircount et maxcount est fournie pour permettre au client de donner des indications au serveur. Si le client collecte agressivement les attributs durant un READDIR, le serveur a une idée de comment limiter la réponse codée.

Si dircount est zéro, le serveur lie la taille de la réponse sur la base du champ maxcount de la demande.

Le cookieverf peut être utilisé par le serveur pour aider à gérer les valeurs de mouchard qui peuvent se périmer. Il devrait n'arriver que rarement qu'un serveur soit incapable de continuer de lire correctement un répertoire avec la paire de mouchard/cookieverf fournie. Le serveur DEVRAIT faire tous ses efforts pour éviter cette condition car l'application chez le client pourrait être incapable de traiter correctement ce type de défaillance.

L'utilisation de cookieverf va aussi protéger le client contre l'utilisation de valeurs de mouchard READDIR qui pourraient être périmées. Par exemple, si le système de fichiers a migré, le serveur pourrait ou non être capable d'utiliser les mêmes valeurs de mouchard pour servir READDIR comme les utilisait le précédent serveur. Avec le client qui fournit le cookieverf, le serveur est capable de donner la réponse appropriée au client. Cela empêche le cas où le serveur accepte une valeur de mouchard mais où le répertoire sous-jacent a changé et la réponse est invalide dans le contexte du précédent READDIR du client.

Comme certains serveurs ne vont pas retourner des entrées "." et ".." comme c'était fait avec les précédentes versions du protocole NFS, le client qui exige que ces entrées soient présentes dans les réponses READDIR doit les fabriquer.

18.24 Opération 27 : READLINK - lire le lien symbolique

18.24.1 Arguments

/* CURRENT_FH : symlink */
Vide

18.24.2 Résultats

struct READLINK4resok {

```
    linktext4      link;
};

union READLINK4res switch (nfsstat4 status) {
  cas NFS4_OK :   READLINK4resok resok4;
  défaut :  vide
};
```

18.24.3 Description

READLINK lit les données associées à une liaison symbolique. Selon la valeur de l'attribut de capacité UTF-8 (paragraphe 14.4) les données sont codées en UTF-8. Qu'elles soient créées par un client NFS ou créées localement sur le serveur, les données dans une liaison symbolique ne sont pas interprétées (sauf éventuellement pour le codage UTF-8 approprié) quand elles sont créées, mais sont simplement mémorisées.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.24.4 Mise en œuvre

Une liaison symbolique est nominalement un pointeur sur un autre fichier. Les données ne sont pas nécessairement interprétées par le serveur, juste mémorisées dans le fichier. Il est possible à une mise en œuvre de client de mémoriser un nom de chemin qui n'a pas de signification pour le système d'exploitation du serveur dans une liaison symbolique. Une opération READLINK retourne les données au client pour leur interprétation. Si des mises en œuvre différentes veulent partager l'accès à des liaisons symboliques, alors elles doivent s'accorder sur l'interprétation des données dans la liaison symbolique.

L'opération READLINK est seulement permise sur les objets de type NF4LNK. Le serveur devrait retourner l'erreur NFS4ERR_WRONG_TYPE si l'objet n'est pas de type NF4LNK.

18.25 Opération 28 : REMOVE - supprime l'objet de système de fichier

18.25.1 Arguments

```
struct REMOVE4args {                          /* CURRENT_FH : répertoire */
    component4      target;
};
```

18.25.2 Résultats

```
struct REMOVE4resok {
    change_info4    cinfo;
};

union REMOVE4res switch (nfsstat4 status) {
  cas NFS4_OK : REMOVE4resok   resok4;
  défaut :  vide
};
```

18.25.3 Description

L'opération REMOVE retire (supprime) une entrée de répertoire nommée par nom de fichier à partir du répertoire correspondant à la bride de fichier courante. Si l'entrée dans le répertoire était la dernière référence à l'objet de système de fichiers correspondant, l'objet peut être détruit. Le répertoire peut être de type NF4DIR ou NF4ATTRDIR.

Pour le répertoire où le nom de fichier a été supprimé, le serveur retourne les informations change_info4 dans cinfo. Avec le champ atomique de type de données change_info4, le serveur va indiquer si les attributs de changement avant et après ont été obtenus atomiquement par rapport à la suppression.

Si la cible a une longueur de zéro, ou si la cible ne respecte pas la définition UTF-8 (et si le serveur applique le codage UTF-8 ; voir le paragraphe 14.4) l'erreur NFS4ERR_INVAL va être retournée.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.25.4 Mise en œuvre

NFSv3 exigeait un opérateur RMDIR différent pour la suppression de répertoire et REMOVE pour une suppression non de répertoire. Cela permettait aux clients de sauter la vérification du type de fichier quand ils passaient un appel système de suppression non de répertoire (par exemple, unlink() [IEEE-1003.1] dans POSIX) pour supprimer un répertoire, ainsi que l'inverse (par exemple, un rmdir() sur un non répertoire) parce que ils savaient que le serveur voudrait vérifier le type de fichier. Le REMOVE de NFSv4.1 peut être utilisé pour supprimer une entrée de répertoire indépendamment de son type de fichier. La mise en œuvre des points d'entrée d'un client NFSv4.1 à partir des appels système unlink() et rmdir() devrait d'abord vérifier le type de fichier par rapport aux types que l'appel système a le droit de supprimer avant d'envoyer une opération REMOVE. Autrement, la mise en œuvre peut produire un appel COMPOUND qui inclut une séquence d'opérations LOOKUP/VERIFY pour vérifier le type de fichier avant une opération REMOVE dans le même appel COMPOUND.

Le concept de dernière référence est spécifique du serveur. Cependant, si le champ numlinks dans les attributs précédents de l'objet avait la valeur 1, le client ne devrait pas s'appuyer sur une référence à l'objet via une bride de fichier. De même, le client ne devrait pas compter que les ressources (espace de disque, entrée de répertoire, et ainsi de suite) précédemment associées à l'objet deviennent immédiatement disponibles. Donc, si un client a besoin d'être capable de continuer d'accéder à un fichier après avoir utilisé REMOVE pour le supprimer, le client devrait prendre des mesures pour s'assurer que le fichier va encore être accessible. Bien que le mécanisme traditionnel utilisé soit d'utiliser RENAME sur le fichier pour passer de son ancien nom à un nouveau nom caché, l'opération OPEN de NFSv4.1 PEUT retourner un fanion de résultat, OPEN4_RESULT_PRESERVE_UNLINKED, qui indique au client que le fichier va être préservé si le fichier a une instance ouverte (voir le paragraphe 18.16).

Si le serveur trouve que le fichier est encore ouvert quand le REMOVE arrive :

* Le serveur NE DEVRAIT PAS supprimer l'entrée de répertoire du fichier si le fichier a été ouvert avec OPEN4_SHARE_DENY_WRITE ou OPEN4_SHARE_DENY_BOTH.
* Si le fichier n'a pas été ouvert avec OPEN4_SHARE_DENY_WRITE ou OPEN4_SHARE_DENY_BOTH, le serveur DEVRAIT supprimer l'entrée de répertoire du fichier. Cependant, jusqu'au dernier CLOSE du fichier, le serveur PEUT continuer de permettre l'accès au fichier via sa bride de fichier.
* Le serveur NE DOIT PAS supprimer l'entrée de répertoire si la réponse au OPEN avait le fanion OPEN4_RESULT_PRESERVE_UNLINKED établi.

Le serveur PEUT mettre en œuvre ses propres restrictions à la suppression d'un fichier pendant qu'il est ouvert. Le serveur pourrait interdire un tel REMOVE (ou une suppression qui se produit au titre de RENAME). Les conditions qui influencent les restrictions à la suppression d'un fichier pendant qu'il est encore ouvert incluent :

* si certains protocoles d'accès (c'est-à-dire, pas juste NFS) détiennent le fichier ouvert,
* si des options particulières, modes d'accès, ou politiques sont activées sur le serveur.

Si un fichier a un OPEN en instance et si cela empêche la suppression de l'entrée de répertoire du fichier, l'erreur NFS4ERR_FILE_OPEN est retournée.

Lorsque la détermination ci-dessus ne peut pas être faite de façon certaine parce que des délégations sont détenues, elles DOIVENT être rappelées pour permettre la poursuite du traitement du REMOVE. Quand une délégation est détenue, le serveur n'a pas une connaissance fiable de l'état des OPEN pour ce client, donc sauf si il y a des fichiers ouverts avec les modes de refus particulier par des clients sans délégation, la détermination ne peut pas être faite tant que les délégations ne sont pas rappelées, et l'opération ne peut pas se faire avant que chaque délégation ait été retournée ou révoquée pour permettre au serveur de faire une détermination correcte.

Dans tous les cas où les délégations sont rappelées, le serveur va probablement retourner une ou plusieurs erreurs NFS4ERR_DELAY pendant que des délégations restent en instance.

Si la bride de fichier courante désigne un répertoire pour lequel un autre client détient une délégation de répertoire, alors, sauf si la situation peut être résolue par l'envoi d'une notification, la délégation de répertoire DOIT être rappelée, et l'opération NE DOIT PAS se poursuivre avant que la délégation soit retournée ou révoquée. Sauf lorsque cela arrive très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites pendant que la

délégation reste en instance.

Quand la bride de fichier courante désigne un répertoire pour lequel une ou plusieurs délégations de répertoire existent, alors, quand ces délégations demandent de telles notifications, NOTIFY4_REMOVE_ENTRY va être généré par suite de cette opération.

Noter que quand une suppression se produit par suite d'un RENAME, NOTIFY4_REMOVE_ENTRY va seulement être généré si la suppression se produit comme une opération séparée. Dans le cas où la suppression est intégrée et atomique avec RENAME, la notification de la suppression est intégrée à la notification pour le RENAME. Voir la discussion de la notification NOTIFY4_RENAME_ENTRY au paragraphe 20.4.

18.26 Opération 29 : RENAME - Renomme l'entrée de répertoire

18.26.1 Arguments

```
struct RENAME4args {
        component4      oldname;                        /* SAVED_FH : répertoire source */
        component4      newname;                        /* CURRENT_FH : répertoire cible */
};
```

18.26.2 Résultats

```
struct RENAME4resok {
        change_info4    source_cinfo;
        change_info4    target_cinfo;
};

union RENAME4res switch (nfsstat4 status) {
   cas NFS4_OK : RENAME4resok    resok4;
   défaut : vide
};
```

18.26.3 Description

L'opération RENAME renomme l'objet identifié par oldname dans le répertoire source correspondant à la bride de fichier sauvegardée, comme établi par l'opération SAVEFH, en newname dans le répertoire cible correspondant à la bride de fichier courante. Il est exigé que l'opération soit atomique pour le client. Les répertoires source et cible DOIVENT résider sur le même système de fichiers chez le serveur. En cas de succès, la bride de fichier courante va continuer d'être le répertoire cible.

Si le répertoire cible contient déjà une entrée avec le nom newname, l'objet de source DOIT être compatible avec la cible : soit les deux sont non répertoires, soit les deux sont répertoires et la cible DOIT être vide. Si ils sont compatibles, la cible existante est supprimée avant que le renommage survienne ou, de préférence, la cible est supprimée atomiquement au titre du renommage. Voir au paragraphe 18.25.4 les actions de client et de serveur chaque fois que une cible est supprimée. Noter cependant que quand la suppression est effectuée atomiquement avec le renommage, certaines parties de la suppression décrites ici sont intégrées au renommage. Par exemple, la notification de la suppression ne va pas être via un NOTIFY4_REMOVE_ENTRY mais va être indiquée au titre du NOTIFY4_ADD_ENTRY ou NOTIFY4_RENAME_ENTRY généré par le renommage.

Si l'objet source et la cible ne sont pas compatibles ou si la cible est un répertoire mais n'est pas vide, le serveur va retourner l'erreur NFS4ERR_EXIST.

Si oldname et newname se réfèrent tous les deux au même fichier (par exemple, ils pourraient être des liaisons physiques l'un de l'autre) alors sauf si le fichier est ouvert (voir le paragraphe 18.26.4) RENAME DOIT n'effectuer aucune action et retourner NFS4_OK.

Pour les deux répertoires impliqués dans le RENAME, le serveur retourne les informations de change_info4. Avec le champ atomique de type de données change_info4, le serveur va indiquer si les attributs de changement avant et après ont été obtenus atomiquement par rapport au renommage.

Si oldname se réfère à un attribut nommé et si les brides de fichier sauvegardée et courante se réfèrent à des objets différents de système de fichiers, le serveur va retourner NFS4ERR_XDEV tout comme si les brides de fichier sauvegardée et courante représentaient des répertoires sur des systèmes de fichiers différents.

Si oldname ou newname a une longueur de zéro, ou si oldname ou newname ne respectent pas la définition UTF-8, l'erreur NFS4ERR_INVAL va être retournée.

18.26.4 Mise en œuvre

Le serveur PEUT imposer des restrictions sur l'opération RENAME telles que RENAME puisse n'être pas fait quand le fichier à renommer est ouvert ou quand cette ouverture est faite par des protocoles particuliers, ou avec des options ou modes d'accès particuliers. Des restrictions similaires peuvent être appliquées quand un fichier existe avec le nom cible et est ouvert. Quand RENAME est rejeté à cause de telles restrictions, l'erreur NFS4ERR_FILE_OPEN est retournée.

Quand oldname et rename se réfèrent au même fichier et que ce fichier est ouvert d'une façon telle que RENAME va normalement être rejeté avec NFS4ERR_FILE_OPEN si oldname et newname étaient des fichiers différents, alors RENAME DEVRAIT être rejeté avec NFS4ERR_FILE_OPEN.

Si un serveur met en œuvre de telles restrictions et que ces restrictions incluent des cas d'ouvertures NFSv4 qui empêchent la réussite de l'exécution d'un renommage, le serveur doit rappeler toutes les délégations qui pourraient cacher l'existence d'ouvertures relevant de cette décision. C'est parce que quand un client détient une délégation, le serveur pourrait ne pas avoir un compte précis des ouvertures pour ce client, car le client peut exécuter en local les OPEN et les CLOSE. L'opération RENAME doit seulement être retardée jusqu'à ce qu'un résultat définitif puisse être obtenu. Par exemple, si il y a plusieurs délégations et si une d'elles établit une ouverture dont la présence empêcherait le renommage, selon la sémantique du serveur, NFS4ERR_FILE_OPEN peut être retourné à l'appelant aussitôt que cette délégation est retournée sans attendre que d'autres délégations soient retournées. De même, si de telles ouvertures ne sont pas associées à des délégations, NFS4ERR_FILE_OPEN peut être retourné immédiatement sans faire de rappel de délégation.

Si la bride de fichier courante ou sauvegardée désigne un répertoire pour lequel un autre client détient une délégation de répertoire, alors, sauf si la situation peut être résolue par l'envoi d'une notification, la délégation DOIT être rappelée, et l'opération ne peut pas se poursuivre avant que la délégation soit retournée ou révoquée. Sauf lorsque cela arrive très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites alors que la délégation reste en instance.

Quand les brides de fichier courante et sauvegardée sont la même et désignent un répertoire pour lequel une ou plusieurs délégations de répertoire existent, alors, quand ces délégations demandent de telles notifications, une notification de type NOTIFY4_RENAME_ENTRY va être générée par suite de cette opération. Quand oldname et rename se réfèrent au même fichier, aucune notification n'est générée (parce que, comme le déclare le paragraphe 18.26.3, le serveur DOIT ne prendre aucune action). Quand un fichier est supprimé parce que il a le même nom que la cible, si cette suppression est faite atomiquement avec le renommage, une notification NOTIFY4_REMOVE_ENTRY ne sera pas générée. À la place, la suppression du fichier va être rapportée au titre de la notification NOTIFY4_RENAME_ENTRY.

Quand les brides de fichier courante et sauvegardée ne sont pas les mêmes :

* Si la bride de fichier courante désigne un répertoire pour lequel une ou plusieurs délégations de répertoire existent, alors, quand ces délégations demandent de telles notifications, NOTIFY4_ADD_ENTRY va être généré par suite de cette opération. Quand un fichier est supprimé parce que il a le même nom que la cible, si cette suppression est faite atomiquement avec le renommage, une notification NOTIFY4_REMOVE_ENTRY ne va pas être générée. À la place, la suppression du fichier va être rapportée au titre de la notification NOTIFY4_ADD_ENTRY.

* Si la bride de fichier sauvegardée désigne un répertoire pour lequel une ou plusieurs délégations de répertoire existent, alors, quand ces délégations demandent de telles notifications, NOTIFY4_REMOVE_ENTRY va être généré par suite de cette opération.

Si l'objet renommé a des délégations de fichier détenues par des clients autres que celui qui fait le RENAME, les délégations DOIVENT être rappelées, et l'opération ne peut pas se faire tant que chacune de ces délégations n'est pas retournée ou révoquée. Noter que dans le cas de plusieurs fichiers reliés, les exigences de rappel de délégation s'appliquent même si la délégation a été obtenue sous un nom différent de celui qui est renommé. Dans tous les cas où des délégations sont rappelées, le serveur va probablement retourner une ou plusieurs erreurs NFS4ERR_DELAY pendant que la ou les

délégations restent en instance, mais il ne pourrait pas le faire si les délégations sont retournées rapidement.

L'opération RENAME doit être atomique pour le client. La déclaration "les répertoires de source et cible DOIVENT résider sur le même système de fichiers sur le serveur" signifie que les champs fsid dans les attributs pour les répertoires sont les mêmes. Si ils résident sur des systèmes de fichiers différents, l'erreur NFS4ERR_XDEV est retournée.

Sur la base de la valeur de l'attribut fh_expire_type pour l'objet, la bride de fichier peut ou non expirer sur un RENAME. Cependant, les mises en œuvre de serveur sont vivement encouragées à tenter d'empêcher les brides de fichier d'expirer de cette façon.

Sur certains serveurs, les noms de fichier "." et ".." sont illégaux comme oldname ou newname, et vont résulter en l'erreur NFS4ERR_BADNAME. De plus, sur de nombreux serveurs le cas de oldname ou newname comme alias pour le répertoire de source va être vérifié. Ces serveurs vont retourner l'erreur NFS4ERR_INVAL dans ce cas.

Si les brides de fichier de source ou cible ne sont pas de répertoires, le serveur va retourner NFS4ERR_NOTDIR.

18.27 Opération 31 : RESTOREFH - restaure la bride de fichier sauvegardée

18.27.1 Arguments

/* SAVED_FH: */
Vide

18.27.2 Résultats

```
struct RESTOREFH4res {
/* * Si status est NFS4_OK, nouvelle CURRENT_FH : valeur de la bride de fichier sauvegardée */
        nfsstat4        status;
};
```

18.27.3 Description

L'opération RESTOREFH règle la bride de fichier et l'identifiant d'état courants aux valeurs de la bride de fichier et identifiant d'état sauvegardés. Si il n'y a pas de bride de fichier sauvegardée, alors le serveur va retourner l'erreur NFS4ERR_NOFILEHANDLE. Voir au paragraphe 16.2.3.1.1 les détails sur la bride de fichier courante. Voir au paragraphe 16.2.3.1.2 les détails sur l'identifiant d'état courant.

18.27.4 Mise en œuvre

Les opérations comme OPEN et LOOKUP utilisent la bride de fichier courante pour représenter un répertoire et le remplacer par une nouvelle bride de fichier. En supposant que la bride de fichier précédente a été sauvegardée avec un opérateur SAVEFH, la bride de fichier précédente peut être restaurée comme bride de fichier courante. Ceci est couramment utilisé pour obtenir les attributs post opération pour le répertoire, par exemple,
PUTFH (répertoire bride de fichier)
SAVEFH
GETATTR attrbits (attributs de répertoire pré opération)
CREATE optbits "foo" attrs
GETATTR attrbits (attributs de fichier)
RESTOREFH
GETATTR attrbits (attributs de répertoire post opération)

18.28 Opération 32 : SAVEFH - sauvegarde la bride de fichier courante

18.28.1 Arguments

/* CURRENT_FH: */
Vide

18.28.2 Résultats

```
struct SAVEFH4res {
/* * Si status est NFS4_OK, nouvelle SAVED_FH : valeur de la bride de fichier courante */
      nfsstat4       status;
};
```

18.28.3 Description

L'opération SAVEFH sauvegarde la bride de fichier et l'identifiant d'état courants. Si une précédente bride de fichier était sauvegardée, alors elle n'est plus accessible. La bride de fichier sauvegardée peut être restaurée comme bride de fichier courante avec l'opérateur RESTOREFH. En cas de succès, la bride de fichier courante conserve sa valeur. Voir au paragraphe 16.2.3.1.1 les détails sur la bride de fichier courante. Voir au paragraphe 16.2.3.1.2 les détails sur l'identifiant d'état courant.

18.29 Opération 33 : SECINFO - obtient la sécurité disponible

18.29.1 Arguments

```
struct SECINFO4args {                                   /* CURRENT_FH : répertoire */
      component4      name;
};
```

18.29.2 Résultats

```
/* * D'après la RFC 2203 */
enum rpc_gss_svc_t {
      RPC_GSS_SVC_NONE          = 1,
      RPC_GSS_SVC_INTEGRITY   = 2,
      RPC_GSS_SVC_PRIVACY       = 3
};

struct rpcsec_gss_info {
      sec_oid4          oid;
      qop4              qop;
      rpc_gss_svc_t   service;
};

/* RPCSEC_GSS a une valeur de "6" - voir la RFC 2203 */
union secinfo4 switch (uint32_t flavor) {
   cas RPCSEC_GSS :   rpcsec_gss_info        flavor_info;
   défaut : vide
};

typedef secinfo4 SECINFO4resok<>;
union SECINFO4res switch (nfsstat4 status) {
   cas NFS4_OK :  SECINFO4resok resok4;                 /* CURRENTFH : consommée */
   défaut :   vide
};
```

18.29.3 Description

L'opération SECINFO est utilisée par le client pour obtenir une liste de nuances valides d'authentification RPC pour une bride de fichier de répertoire spécifique, une paire de fichier et de nom. SECINFO devrait appliquer la même méthodologie d'accès qu'utilisée pour LOOKUP quand on évalue le nom. Donc, si le demandeur n'a pas l'accès approprié pour faire LOOKUP sur le nom, alors SECINFO DOIT se comporter de la même façon et retourner NFS4ERR_ACCESS.

La résultat va contenir un dispositif qui représente les mécanismes de sécurité disponibles, avec un ordre correspondant aux préférences du serveur, le préféré étant le premier du dispositif. Le client est libre de prendre le mécanisme de sécurité qu'il

désire et prend en charge, ou de prendre dans l'ordre de préférence du serveur le premier qu'il prend en charge. Les entrées du dispositif sont représentées par la structure secinfo4. Le champ "flavor" va contenir une valeur de AUTH_NONE, AUTH_SYS (comme défini dans la [RFC5531]), ou RPCSEC_GSS (comme défini dans la [RFC2203]). Le champ "flavor" peut aussi être toute autre nuance de sécurité enregistrée par l'IANA.

Pour les nuances AUTH_NONE et AUTH_SYS, aucune information de sécurité supplémentaire n'est retournée. La même chose est vraie de nombreuses (sinon la plupart) des autres nuances de sécurité, y compris AUTH_DH. Pour une valeur de retour de RPCSEC_GSS, un triplet de sécurité est retourné qui contient l'identifiant d'objet du mécanisme (un OID, comme défini dans la [RFC2743]) la qualité de protection (comme définie dans la [RFC2743]) et le type de service (comme défini dans la [RFC2203]). Il est possible que SECINFO retourne plusieurs entrées avec une nuance égale à RPCSEC_GSS avec de valeurs de triplet de sécurité différentes.

En cas de succès, la bride de fichier courante est consommée (voir le paragraphe 2.6.3.1.1.8) et si la prochaine opération après SECINFO essaye d'utiliser la bride de fichier courante, cette opération va échouer avec l'état NFS4ERR_NOFILEHANDLE.

Si le nom a une longueur de zéro, ou si le nom ne respecte pas la définition UTF-8 (en supposant que les capacités UTF-8 sont activées ; voir le paragraphe 14.4) l'erreur NFS4ERR_INVAL va être retournée.

Voir au paragraphe 2.6 des informations supplémentaires sur l'utilisation de SECINFO.

18.29.4 Mise en œuvre

L'opération SECINFO est supposée être utilisée par le client NFS quand la valeur d'erreur de NFS4ERR_WRONGSEC est retournée d'une autre opération NFS. Cela signifie pour le client que la politique de sécurité du serveur est différente de ce que le client utilise actuellement. À ce point, le client est supposé obtenir une liste des nuances de sécurité possibles et choisir celle qui convient le mieux à sa politique.

Comme mentionné précédemment, les politiques de sécurité du serveur vont déterminer quand la demande du client reçoit NFS4ERR_WRONGSEC. Voir au Tableau 14 la liste des opérations qui peuvent retourner NFS4ERR_WRONGSEC. De plus, quand READDIR retourne des attributs, rdattr_error (paragraphe 5.8.1.12) peut contenir NFS4ERR_WRONGSEC. Noter que CREATE et REMOVE NE DOIVENT PAS retourner NFS4ERR_WRONGSEC. La raison pour CREATE est que sauf si le nom de la cible existe, il ne peut pas avoir une politique de sécurité différente de celle du répertoire parent, et la politique de sécurité du parent a été vérifiée quand sa bride de fichier a été injectée dans le flux d'opérations de la demande COMPOUND (pour des raisons similaires, une opération OPEN qui crée la cible NE DOIT PAS retourner NFS4ERR_WRONGSEC). Si le nom cible existe, alors qu'elle pourrait avoir une politique de sécurité séparée, cela n'est pas pertinent parce que CREATE DOIT retourner NFS4ERR_EXIST. La raison pour REMOVE est que alors que cette cible pourrait avoir une politique de sécurité séparée, la cible va être supprimée, et donc la politique de sécurité du parent prend le pas sur celle de l'objet à supprimer. RENAME et LINK PEUVENT retourner NFS4ERR_WRONGSEC, mais l'erreur NFS4ERR_WRONGSEC s'applique seulement à la bride de fichier sauvegardée (voir le paragraphe 2.6.3.1.2). Toute erreur NFS4ERR_WRONGSEC sur la bride de fichier courante utilisée par LINK et RENAME DOIT être retournée par l'opération PUTFH, PUTPUBFH, PUTROOTFH, ou RESTOREFH qui a injecté la bride de fichier courante.

À l'exception de LINK et RENAME, l'ensemble des opérations qui peuvent retourner NFS4ERR_WRONGSEC représente le point auquel le client peut injecter une bride de fichier dans la "bride de fichier courante" chez le serveur. La bride de fichier est soit fournie par le client (PUTFH, PUTPUBFH, PUTROOTFH) générée par suite d'une traduction de nom à bride de fichier (LOOKUP et OPEN) ou générée à partir de la bride de fichier sauvegardée via RESTOREFH. Comme le déclare le paragraphe 2.6.3.1.1.1, une opération Mettre une bride de fichier suivie par SAVEFH NE DOIT PAS retourner NFS4ERR_WRONGSEC. Donc, il est permis à l'opération RESTOREFH, dans certaines conditions (voir le paragraphe 2.6.3.1.1) de retourner NFS4ERR_WRONGSEC afin que les politiques de sécurité puissent être honorées.

L'opération READDIR ne va pas retourner directement l'erreur NFS4ERR_WRONGSEC. Cependant, si la demande READDIR incluait une demande d'attributs, il est possible que le triplet de sécurité de la demande READDIR ne corresponde pas à celui d'une entrée de répertoire. Si c'est le cas et si le client a demandé l'attribut rdattr_error, le serveur va retourner l'erreur NFS4ERR_WRONGSEC dans rdattr_error pour l'entrée.

Pour résoudre une erreur en retour de NFS4ERR_WRONGSEC, le client fait ce qui suit :

* Pour LOOKUP et OPEN, le client va utiliser SECINFO avec les mêmes bride de fichier courante et nom que fournis dans le LOOKUP ou OPEN original pour énumérer les triplets de sécurité disponibles.

* Pour la rdattr_error, le client va utiliser SECINFO avec la même bride de fichier courante que fournie dans le READDIR original. Le nom passé à SECINFO va être celui de l'entrée de répertoire (comme retourné de READDIR) qui avait l'erreur NFS4ERR_WRONGSEC dans l'attribut rdattr_error.

* Pour PUTFH, PUTROOTFH, PUTPUBFH, RESTOREFH, LINK, et RENAME, le client va utiliser SECINFO_NO_NAME { style = SECINFO_STYLE4_CURRENT_FH }. Le client va faire précéder l'opération SECINFO_NO_NAME de l'opération PUTFH, PUTPUBFH, ou PUTROOTFH appropriée qui a fourni la bride de fichier originellement fournie par l'opération PUTFH, PUTPUBFH, PUTROOTFH, ou RESTOREFH.

Note : dans NFSv4.0, il était exigé du client qu'il utilise SECINFO, et il devait reconstruire le parent de la bride de fichier originale et le nom de composant de la bride de fichier originale. L'introduction dans NFSv4.1 de SECINFO_NO_NAME supprime le besoin de reconstruction.

* Pour LOOKUPP, le client va utiliser SECINFO_NO_NAME { style = SECINFO_STYLE4_PARENT } et fournir la bride de fichier qui est égale à la bride de fichier originellement fournie à LOOKUPP.

Voir à la Section 21 la discussion sur les recommandations pour la nuance de sécurité utilisée par SECINFO et SECINFO_NO_NAME.

18.30 Opération 34 : SETATTR - établir les attributs

18.30.1 Arguments

```
struct SETATTR4args {                                   /* CURRENT_FH : objet cible */
      stateid4      stateid;
      fattr4        obj_attributes;
};
```

18.30.2 Résultats

```
struct SETATTR4res {
      nfsstat4      status;
      bitmap4       attrsset;
};
```

18.30.3 Description

L'opération SETATTR change un ou plusieurs des attributs d'un objet de système de fichiers. Les nouveaux attributs sont spécifiés avec un gabarit binaire et les attributs qui suivent le gabarit binaire dans l'ordre des bits.

L'argument d'identifiant d'état pour SETATTR est utilisé pour fournir le contexte de verrouillage de gamme d'octets qui est nécessaire pour les demandes SETATTR qui règlent l'attribut de taille. Comme le réglage de l'attribut de taille modifie les données du fichier, il a les mêmes exigences de verrouillage qu'un WRITE correspondant. Tout SETATTR qui règle l'attribut de taille est incompatible avec une réservation partagée qui spécifie OPEN4_SHARE_DENY_WRITE. La zone entre la vieille fin de fichier et la nouvelle fin de fichier est considérée être modifiée juste comme elle l'aurait été dans le cas où la zone en question aurait été spécifiée comme la cible de WRITE, pour les besoins de vérifier des conflits avec des verrous de gamme d'octets, pour les cas où un serveur met en œuvre le comportement de verrouillage de gamme d'octets obligatoire. Un identifiant d'état valide DEVRAIT toujours être spécifié. Quand l'attribut de taille de fichier n'est pas établi, l'identifiant d'état spécial consistant en tous les bits à zéro PEUT être passé.

En cas de succès ou d'échec de l'opération, le serveur va retourner le gabarit binaire attrsset pour représenter quels attributs (si il en est) ont réussi à être établis. Le attrsset dans la réponse est un sous ensemble du champ attrmask du champ obj_attributes dans l'argument.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.30.4 Mise en œuvre

Si la demande spécifie l'attribut owner à établir, le serveur DEVRAIT permettre que l'opération réussisse si le propriétaire actuel de l'objet correspond à la valeur spécifiée dans la demande. Certains serveurs peuvent être mis en œuvre d'une façon qui interdise l'établissement de l'attribut propriétaire sauf si le demandeur a le privilège de le faire. Si le serveur est indulgent dans ce cas pour confronter les valeurs de propriétaire, la mise en œuvre de client peut être simplifiée dans les cas de création d'un objet (par exemple, une création exclusive via OPEN) suivie par un SETATTR.

L'attribut de taille de fichier est utilisé pour demander des changements de la taille d'un fichier. Une valeur de zéro cause la troncature du fichier, une valeur de moins que la taille courante du fichier cause l'élimination des données à partir de la nouvelle taille jusqu'à la fin du fichier, et une taille supérieure à la taille courante du fichier cause l'ajout d'octets de données logiquement de zéros à la fin du fichier. Les serveurs sont libres de mettre en œuvre cela en utilisant des octets non alloués (trous) ou des octets de données alloués réglés à zéro. Les clients ne devraient pas faire d'hypothèses concernant la mise en œuvre de cette caractéristique par un serveur, au delà de ce que les octets dans la gamme d'octets affectée retournée par READ vont être mis à zéro. Les serveurs DOIVENT prendre en charge d'étendre la taille de fichier via SETATTR.

Il n'est pas garanti que SETATTR soit atomique. Un SETATTR échoué peut partiellement changer les attributs d'un fichier, d'où la raison pour laquelle la réponse inclut toujours l'état et la liste des attributs qui ont été établis.

Si l'objet dont les attributs sont changés a une délégation de fichier qui est détenue par un client autre que celui qui fait le SETATTR, la ou les délégations doivent être rappelées, et l'opération ne peut pas procéder au changement réel d'un attribut avant que chacune de ces délégations soit retournée ou révoquée. Dans tous les cas où des délégations sont rappelées, le serveur va probablement retourner une ou plusieurs erreurs NFS4ERR_DELAY pendant que la délégation reste en instance, bien qu'elle pourrait ne pas le faire si les délégations sont retournées rapidement.

Si l'objet dont les attributs sont en train d'être établis est un répertoire et qu'un autre client détient une délégation de répertoire pour ce répertoire, alors si elles sont activées, des notifications asynchrones vont être générées quand l'ensemble des attributs changés a une intersection non nulle avec l'ensemble des attributs pour lesquels une notification est demandée. Les notifications de type NOTIFY4_CHANGE_DIR_ATTRS vont être envoyées aux clients appropriés, mais le SETATTR n'est pas retardé par l'attente que ces notifications soient envoyées.

Si l'objet dont les attributs sont en train d'être établis est un membre du répertoire pour lequel un autre client détient une délégation de répertoire, alors des notifications asynchrones vont être générées quand l'ensemble d'attributs changés a une intersection non nulle avec l'ensemble d'attributs pour lesquels une notification est demandée. Les notifications de type NOTIFY4_CHANGE_CHILD_ATTRS vont être envoyées aux clients appropriés, mais le SETATTR n'est pas retardé par l'attente que ces notifications soient envoyées.

Changer la taille d'un fichier avec SETATTR change indirectement les attributs time_modify et change. Un client doit tenir compte de cela car les changements de taille peuvent résulter en la suppression de données.

Les attributs time_access_set et time_modify_set sont des attributs en écriture seule construits comme une union commutée de sorte que le client peut diriger le serveur dans le réglage des valeurs de temps. Si l'union commutée spécifie SET_TO_CLIENT_TIME4, le client a fourni un nfstime4 à utiliser pour l'opération. Si l'union commutée ne spécifie pas SET_TO_CLIENT_TIME4, le serveur va utiliser sa propre heure courante pour l'opération SETATTR.

Si les heures du serveur et du client diffèrent, les programmes qui comparent l'heure du client à l'heure du fichier peuvent casser. Un protocole de synchronisation de l'heure devrait être utilisé pour limiter la dérive de l'heure entre client et serveur.

L'utilisation d'un COMPOUND contenant une opération VERIFY qui spécifie seulement l'attribut de changement, suivi immédiatement par un SETATTR, fournit un moyen pour qu'un client puisse spécifier une demande qui émule la fonction de mécanisme de garde de SETATTR de NFSv3. Comme la fonction du mécanisme de garde est d'éviter des changements aux attributs du fichier sur la base d'informations périmées, les délais entre la vérification de la condition de garde et le réglage des attributs ont un potentiel de compromission de cette fonction, comme le ferait le délai correspondant dans l'émulation NFSv4. Donc, les serveurs NFSv4.1 DEVRAIENT veiller à éviter de tels délais, dans la mesure du possible, quand ils exécutent une telle demande.

Si le serveur ne prend pas en charge un attribut comme demandé par le client, le serveur DEVRAIT retourner NFS4ERR_ATTRNOTSUPP.

Un gabarit des attributs actuellement établis est retourné par SETATTR dans tous les cas. Ce gabarit NE DOIT PAS inclure des bits d'attribut dont l'établissement n'est pas demandé par le client. Si le gabarit d'attribut dans la demande et la réponse

sont égaux, le champ status dans la réponse DOIT être NFS4_OK.

18.31 Opération 37 : VERIFY - vérifie les mêmes attributs

18.31.1 Arguments

```
struct VERIFY4args {                                    /* CURRENT_FH : objet */
      fattr4        obj_attributes;
};
```

18.31.2 Résultats

```
struct VERIFY4res {
      nfsstat4      status;
};
```

18.31.3 Description

L'opération VERIFY est utilisée pour vérifier que les attributs ont la valeur supposée par le client avant de procéder aux opérations suivantes dans la demande COMPOUND. Si un des attributs ne correspond pas, alors l'erreur NFS4ERR_NOT_SAME doit être retournée. La bride de fichier courante conserve sa valeur après l'achèvement avec succès de l'opération.

18.31.4 Mise en œuvre

Une utilisation possible de l'opération VERIFY est la série d'opérations suivante. Avec cela, le client tente de vérifier que le fichier en cours de suppression va correspondre à ce que le client s'attend à voir supprimer. Cette série peut aider à empêcher la suppression involontaire d'un fichier.

PUTFH (bride de fichier de répertoire)
LOOKUP (nom de fichier)
VERIFY (bride de fichier == fh)
PUTFH (bride de fichier de répertoire)
REMOVE (nom de fichier)

Cette série n'empêche pas un second client de supprimer et créer un nouveau fichier au milieu de cette séquence, mais elle aide à éviter le résultat involontaire.

Dans le cas où un attribut RECOMMANDÉ est spécifié dans l'opération VERIFY et où le serveur ne prend pas en charge cet attribut pour l'objet de système de fichiers, l'erreur NFS4ERR_ATTRNOTSUPP est retournée au client.

Quand l'attribut rdattr_error ou tout attribut set-only (par exemple, time_modify_set) est spécifié, l'erreur NFS4ERR_INVAL est retournée au client.

18.32 Opération 38 : WRITE - écrit sur le fichier

18.32.1 Arguments

```
enum stable_how4 {
      UNSTABLE4      = 0,
      DATA_SYNC4     = 1,
      FILE_SYNC4     = 2
};

struct WRITE4args {                                     /* CURRENT_FH : fichier */
        stateid4        stateid;
        offset4         offset;
        stable_how4     stable;
```

```
        opaque          data<>;
};
```

18.32.2 Résultats

```
struct WRITE4resok {
        count4          count;
        stable_how4     committed;
        verifier4       writeverf;
};

union WRITE4res switch (nfsstat4 status) {
   cas NFS4_OK : WRITE4resok    resok4;
   défaut : vide
};
```

18.32.3 Description

L'opération WRITE est utilisée pour écrire des données sur un fichier régulier. Le fichier cible est spécifié par la bride de fichier courante. Offset spécifie le décalage où les données devraient être écrites. Un décalage de zéro spécifie que l'écriture devrait commencer au début du fichier. Le compte, codé au titre du paramètre données opaques, représente le nombre d'octets de données à écrire. Si le compte est zéro, le WRITE va réussir et retourner un compte de zéro sous réserve de la vérification des permissions. Le serveur PEUT écrire moins d'octets que demandé par le client.

Le client spécifie avec le paramètre stable la méthode selon laquelle les données sont à traiter par le serveur. Si stable est FILE_SYNC4, le serveur DOIT engager les données écrites plus toutes les métadonnées du système de fichiers sur une mémorisation stable avant de retourner les résultats. Cela correspond à la sémantique du protocole NFSv2. Tout autre comportement constitue une violation du protocole. Si stable est DATA_SYNC4, alors le serveur DOIT engager toutes les données à une mémorisation stable et assez de métadonnées pour restituer les données avant de retourner. La mise en œuvre de serveur est libre d'utiliser DATA_SYNC4 de la même façon que FILE_SYNC4, mais avec une possible chute de performances. Si stable est UNSTABLE4, le serveur est libre d'engager toute partie des données et les métadonnées à une mémorisation stable, incluant tout ou rien, avant de retourner une réponse au client. Il n'y a aucune garantie sur si ou quand des données non engagées vont ensuite être engagées à une mémorisation stable. Les seules garanties du serveur sont qu'il ne va pas détruire de données sans changer la valeur de writeverf et qu'il ne va pas engager les données et métadonnées à un niveau inférieur à celui demandé par le client.

Sauf quand des identifiants d'état spéciaux sont utilisés, la valeur d'identifiant d'état pour une demande WRITE représente une valeur retournée d'une demande précédente LOCK ou OPEN de gamme d'octets ou l'identifiant d'état associé à une délégation. L'identifiant d'état identifie les propriétaires associés si il en est et est utilisé par le serveur pour vérifier que les verrous associés sont encore valides (par exemple, n'ont pas été révoqués).

Quand c'est achevé avec succès, les résultats suivants sont retournés. Le résultat de compte est le nombre d'octets de données écrits sur le fichier. Le serveur peut écrire moins d'octets que demandé. Si il en est ainsi, le nombre réel d'octets écrits en commençant au décalage de localisation est retourné.

Le serveur retourne aussi une indication du niveau d'engagement des données et métadonnées via l'engagement. Selon le Tableau 20,

* Le serveur PEUT engager les données à un niveau supérieur à celui demandé.
* Le serveur DOIT engager les données à un niveau au moins aussi élevé qu'engagé.

Stable	Engagé
UNSTABLE4	FILE_SYNC4, DATA_SYNC4, UNSTABLE4
DATA_SYNC4	FILE_SYNC4, DATA_SYNC4
FILE_SYNC4	FILE_SYNC4

Tableau 20 : Combinaisons valides des champs Stable dans la demande et Engagé dans la réponse

La portion finale du résultat est le champ writeverf. Ce champ est le vérificateur d'écriture et est un mouchard que le client peut utiliser pour déterminer si un serveur a changé l'état d'instance (par exemple, redémarrage de serveur) entre un appel à

WRITE et un appel suivant à WRITE ou COMMIT. Ce mouchard DOIT être inchangé durant une seule instance du serveur NFSv4.1 et DOIT être unique entre les instances du serveur NFSv4.1. Si le mouchard change, alors le client DOIT supposer que toutes les données écrites avec une valeur de UNSTABLE4 pour les engagées et un vieil writeverf dans la réponse ont été perdues et vont devoir être récupérées.

Si un client écrit des données au serveur avec l'argument stable réglé à UNSTABLE4 et si la réponse donne une réponse engagée de DATA_SYNC4 ou UNSTABLE4, le client va suivre un peu après avec une opération COMMIT pour synchroniser les données et métadonnées asynchrones en instance avec la mémorisation stable du serveur, empêchant l'erreur du client. Il est possible qu'une défaillance du client ou une autre erreur empêche un COMMIT suivant d'être reçu par le serveur.

Pour un WRITE avec une valeur d'identifiant d'état de tous les bits égaux à zéro, le serveur PEUT permettre que le WRITE soit servi sous réserve de verrous de gamme d'octets obligatoires ou de modes de refus de partage courants pour le fichier. Pour un WRITE avec une valeur d'identifiant d'état de tous les bits égaux à 1, le serveur NE DOIT PAS permettre que l'opération WRITE outrepasse les vérifications de verrouillage chez le serveur et soit traitée autrement comme si un identifiant d'état de tous les bits égaux à zéro était utilisé.

En cas de succès, la bride de fichier courante conserve sa valeur.

18.32.4 Mise en œuvre

Il est possible que le serveur écrive moins d'octets de données que demandé par le client. Dans ce cas, le serveur NE DEVRAIT PAS retourner une erreur sauf si aucune donnée n'a été écrite. Si le serveur écrit moins que le nombre d'octets spécifié, le client va devoir envoyer un autre WRITE pour écrire les données restantes.

Il est supposé que l'acte d'écrire des données sur un fichier va causer la mise à jour des attributs time_modified et change du fichier. Cependant, ces attributs NE DEVRAIT PAS être changés si le contenu du fichier n'est pas changé. Donc, une demande WRITE avec count réglé à zéro NE DEVRAIT PAS causer la mise à jour des attributs time_modified et change du fichier.

Une mémorisation stable est une mémorisation persistante qui survit à :
1. des pannes d'alimentation électrique répétées,
2. des défaillances du matériel (de tout clavier, alimentation électrique, etc.)
3. de pannes et redémarrages répétés du logiciel.

Cette définition ne vise pas la défaillance du module de mémorisation stable lui-même.

Le vérificateur est défini pour permettre au client de détecter différentes instances d'un serveur de protocole NFSv4.1 sur lesquelles des données en antémémoire non engagées peuvent être perdues. Dans le cas le plus probable, le vérificateur permet au client de détecter les redémarrages du serveur. Cette information est nécessaire afin que le client puisse en toute sécurité déterminer si le serveur pourrait avoir perdu des données mises en antémémoire. Si le serveur échoue de façon inattendue et si le client a des données non engagées provenant de précédentes demandes WRITE (faites avec l'argument stable réglé à UNSTABLE4 et dans lesquelles le résultat engagé a été retourné aussi comme UNSTABLE4) le serveur pourrait ne pas avoir purgé les données mises en antémémoire dans la mémorisation stable. La charge de la récupération pèse sur le client, et le client va avoir besoin de retransmettre les données au serveur.

Un vérificateur suggéré serait d'utiliser l'heure à laquelle le serveur a démarré la dernière fois (si redémarrer le serveur résulte en la perte des mémoires tampons).

Le champ committed de la réponse permet au client de faire une mise en antémémoire plus efficace. Si le serveur engage toutes les demandes WRITE à une mémorisation stable, alors il DEVRAIT retourner avec committed réglé à FILE_SYNC4, sans considération de la valeur du champ stable dans les arguments. Un serveur qui utilise un accélérateur NVRAM peut choisir de mettre en œuvre cette politique. Le client peut utiliser cela pour augmenter l'efficacité de l'antémémoire en y éliminant les données qui ont déjà été engagées sur le serveur.

Certaines mises en œuvre peuvent retourner NFS4ERR_NOSPC au lieu de NFS4ERR_DQUOT quand le quota d'un utilisateur est dépassé.

Dans le cas où la bride de fichier courante est de type NF4DIR, le serveur va retourner NFS4ERR_ISDIR. Si le fichier courant est une liaison symbolique, l'erreur NFS4ERR_SYMLINK va être retournée. Autrement, si la bride de fichier

courante ne désigne pas un fichier ordinaire, le serveur va retourner NFS4ERR_WRONG_TYPE.

Si le verrouillage de gamme d'octets obligatoire est en vigueur pour le fichier, et si la gamme d'octets correspondante de données à écrire sur le fichier est verrouillée par READ_LT ou WRITE_LT par un propriétaire qui n'est pas associé à l'identifiant d'état, le serveur DOIT retourner NFS4ERR_LOCKED. Si il en est ainsi, le client DOIT vérifier si le propriétaire correspondant à l'identifiant d'état utilisé avec l'opération WRITE a un verrou READ_LT en conflit qui se chevauche avec la gamme d'octets qui était à écrire. Si le propriétaire de l'identifiant d'état n'a pas de verrou READ_LT en conflit, alors le client DEVRAIT essayer d'obtenir le verrou d'écriture de gamme d'octets approprié via l'opération LOCK avant de retenter le WRITE. Quand le WRITE s'achève, le client DEVRAIT libérer le verrou de gamme d'octets via LOCKU.

Si le propriétaire de l'identifiant d'état a un verrou READ_LT en conflit, alors le client n'a pas d'autre choix que de retourner une erreur à l'application qui a tenté le WRITE. La raison en est que comme le propriétaire de l'identifiant d'état avait un verrou READ_LT, soit le serveur a tenté de relever temporairement ce verrou READ_LT en un verrou WRITE_LT, soit le serveur n'a pas la capacité de mettre à niveau. Si le serveur a tenté de relever le verrou READ_LT et a échoué, il n'y a pas lieu pour le client de tenter à nouveau la mise à niveau via l'opération LOCK, parce que ce pourrait être qu'un autre client essaye aussi de mettre à niveau. Si deux clients sont bloqués en essayant de mettre à niveau le même verrou, les clients sont bloqués. Si le serveur n'a pas la capacité de mettre à niveau, alors il est vain d'essayer une opération LOCK pour mettre à niveau.

Si un ou plusieurs autre clients ont des délégations pour le fichier sur lequel écrire, ces délégations DOIVENT être rappelées, et l'opération ne peut pas se faire tant que ces délégations ne sont pas retournées ou révoquées. Sauf si cela se fait très rapidement, une ou plusieurs erreurs NFS4ERR_DELAY vont être retournées aux demandes faites pendant que la délégation reste en instance. Normalement, les délégations ne vont pas être rappelées par suite d'une opération WRITE parce que le rappel va se faire par suite d'un OPEN antérieur. Cependant, comme il est possible qu'un WRITE soit fait avec un identifiant d'état spécial, le serveur a besoin de vérifier ce cas même si le client devrait avoir fait un OPEN précédemment.

18.33 Opération 40 : BACKCHANNEL_CTL - contrôle le canal de retour

18.33.1 ARGUMENT

```
typedef opaque gsshandle4_t<>;

struct gss_cb_handles4 {
      rpc_gss_svc_t       gcbp_service;                 /* RFC 2203 */
      gsshandle4_t        gcbp_handle_from_server;
      gsshandle4_t        gcbp_handle_from_client;
};

union callback_sec_parms4 switch (uint32_t cb_secflavor) {
  cas AUTH_NONE : vide
  cas AUTH_SYS :  authsys_parms   cbsp_sys_cred;          /* RFC 5531 */
  cas RPCSEC_GSS :  gss_cb_handles4  cbsp_gss_handles;
};

struct BACKCHANNEL_CTL4args {
      uint32_t                bca_cb_program;
      callback_sec_parms4     bca_sec_parms<>;
};
```

18.33.2 Résultats

```
struct BACKCHANNEL_CTL4res {
      nfsstat4            bcr_status;
};
```

18.33.3 Description

L'opération BACKCHANNEL_CTL remplace le numéro de programme de rappel du canal de retour et ajoute (sans remplacer) les brides RPCSEC_GSS à utiliser par le canal de retour.

Les arguments de l'appel BACKCHANNEL_CTL sont un sous ensemble des paramètres CREATE_SESSION. Dans les arguments de BACKCHANNEL_CTL, le champ bca_cb_program et les champs bca_sec_parms correspondent respectivement aux champs csa_cb_program et csa_sec_parms des arguments de CREATE_SESSION (paragraphe 18.36).

BACKCHANNEL_CTL DOIT apparaître dans un COMPOUND qui commence avec SEQUENCE.

Si la bride RPCSEC_GSS identifiée par gcbp_handle_from_server n'existe pas sur le serveur, le serveur DOIT retourner NFS4ERR_NOENT.

Si une bride RPCSEC_GSS utilise le contexte de SSV (voir le paragraphe 2.10.9) alors parce que chaque bride RPCSEC_GSS de SSV partage un contexte GSS de SSV commun, il y a des considérations de sécurité spécifiques de cette situation qui sont discutées au paragraphe 2.10.10.

18.34 Opération 41 : BIND_CONN_TO_SESSION - associe la connexion à la session

18.34.1 Argument

```
enum channel_dir_from_client4 {
   CDFC4_FORE                = 0x1,
   CDFC4_BACK                = 0x2,
   CDFC4_FORE_OR_BOTH        = 0x3,
   CDFC4_BACK_OR_BOTH        = 0x7
  };

struct BIND_CONN_TO_SESSION4args {
   sessionid4                bctsa_sessid;
   channel_dir_from_client4  bctsa_dir;
   bool                      bctsa_use_conn_in_rdma_mode;
};
```

18.34.2 Résultat

```
enum channel_dir_from_server4 {
   CDFS4_FORE     = 0x1,
   CDFS4_BACK     = 0x2,
   CDFS4_BOTH     = 0x3
  };

struct BIND_CONN_TO_SESSION4resok {
   sessionid4                bctsr_sessid;
   channel_dir_from_server4  bctsr_dir;
   bool                      bctsr_use_conn_in_rdma_mode;
};

union BIND_CONN_TO_SESSION4res switch (nfsstat4 bctsr_status) {
   cas NFS4_OK :  BIND_CONN_TO_SESSION4resok  bctsr_resok4;
   défaut :  vide
};
```

18.34.3 Description

BIND_CONN_TO_SESSION est utilisé pour associer des connexions supplémentaires à une session. Il DOIT être utilisé sur la connexion associée à la session. Il DOIT être la seule opération dans la procédure COMPOUND. Si la protection d'état SP4_NONE (paragraphe 18.35) est utilisée, tout principal, nuance de sécurité, ou contexte RPCSEC_GSS PEUT être

utilisé pour invoquer l'opération. Si SP4_MACH_CRED est utilisé, RPCSEC_GSS DOIT être utilisé avec les services d'intégrité ou de confidentialité, en utilisant le principal qui a créé l'identifiant de client. Si SP4_SSV est utilisé, RPCSEC_GSS avec le mécanisme GSS de SSV (paragraphe 2.10.9) et l'intégrité ou la confidentialité DOIT être utilisé.

Si, quand l'identifiant de client a été créé, le client a opté pour la protection d'état SP4_NONE, le client n'est pas obligé d'utiliser BIND_CONN_TO_SESSION pour associer la connexion à la session, sauf si le client souhaite associer la connexion au canal de retour. Quand la protection SP4_NONE est utilisée, le simple envoi d'une demande COMPOUND avec une opération SEQUENCE est suffisant pour associer la connexion à la session spécifiée dans SEQUENCE.

Le champ bctsa_dir indique si le client veut associer la connexion au canal avant ou au canal de retour ou aux deux canaux. La valeur CDFC4_FORE_OR_BOTH indique que le client veut associer la connexion aux deux canaux avant et de retour, mais va accepter que la connexion soit associée juste au canal avant. La valeur CDFC4_BACK_OR_BOTH indique que le client veut s'associer avec les deux canaux avant et de retour, mais va accepter que la connexion soit juste associée au canal de retour. Le serveur répond dans bctsr_dir à quel canaux la connexion est associée. Si le client a spécifié CDFC4_FORE, le serveur DOIT retourner CDFS4_FORE. Si le client a spécifié CDFC4_BACK, le serveur DOIT retourner CDFS4_BACK. Si le client a spécifié CDFC4_FORE_OR_BOTH, le serveur DOIT retourner CDFS4_FORE ou CDFS4_BOTH. Si le client a spécifié CDFC4_BACK_OR_BOTH, le serveur DOIT retourner CDFS4_BACK ou CDFS4_BOTH.

Voir l'opération CREATE_SESSION (paragraphe 18.36) et la description de l'argument csa_use_conn_in_rdma_mode pour comprendre bctsa_use_conn_in_rdma_mode, et la description de csr_use_conn_in_rdma_mode pour comprendre bctsr_use_conn_in_rdma_mode.

Invoquer BIND_CONN_TO_SESSION sur une connexion déjà associée à la session spécifiée n'a pas d'effet, et le serveur DOIT répondre avec NFS4_OK, sauf si le client demande des changements à l'ensemble de canaux auxquels la connexion est associée. Si il en est ainsi, le serveur DOIT retourner NFS4ERR_INVAL.

18.34.4 Mise en œuvre

Si le canal d'une session perd toutes les connexions, selon la protection d'état sur l'identifiant de client et le type de canal, le client pourrait avoir besoin d'utiliser BIND_CONN_TO_SESSION pour s'associer à une nouvelle connexion. Si le serveur a redémarré et ne conserve pas la mémoire tampon de réponse dans une mémorisation stable, le serveur ne va pas reconnaître l'identifiant de session. Le client va finalement devoir invoquer EXCHANGE_ID pour créer un nouvel identifiant de client et de session.

Supposons que la protection d'état SP4_SSV soit utilisée, et que BIND_CONN_TO_SESSION soit parmi les opérations incluses dans l'ensemble spo_must_enforce quand l'identifiant de client a été créé (paragraphe 18.35). Si il en est ainsi, il y a un problème si SET_SSV est envoyé, qu'aucune réponse n'est retournée, et si la dernière connexion associée à l'identifiant de client cesse. Le client, selon le modèle des sessions, DOIT réessayer le SET_SSV. Mais il a besoin d'une nouvelle connexion pour le faire, et DOIT associer cette connexion à la session via un BIND_CONN_TO_SESSION authentifié avec le mécanisme GSS du SSV. Le problème est que les codes d'intégrité de message RPCSEC_GSS utilisent une sous clé déduite du SSV car la clé et le SSV peuvent avoir changé. Bien qu'il y aient plusieurs stratégies de récupération, une seule stratégie générale est décrite ici.

* Le client se reconnecte.

* Le client suppose que SET_SSV a été exécuté, et donc envoie BIND_CONN_TO_SESSION avec la sous clé (déduite du nouveau SSV, c'est-à-dire, ce à quoi SET_SSV aurait réglé le SSV) utilisée comme clé pour les codes d'intégrité de message de l'accréditif RPCSEC_GSS.

* Si la demande réussit, cela signifie que le SET_SSV original tenté s'est bien exécuté. Le client envoie à nouveau le SET_SSV original, auquel le serveur va répondre via la mémoire tampon de réponse.

* Si le serveur retourne une erreur d'authentification RPC, cela signifie que le SSV courant du serveur n'a pas changé (et le SET_SSV n'a probablement pas été exécuté). Le client essaye alors BIND_CONN_TO_SESSION avec la sous clé déduite du vieux SSV comme clé pour les codes d'intégrité de message RPCSEC_GSS.

* Le BIND_CONN_TO_SESSION tenté avec le vieux SSV devrait réussir. Si il réussit, le client envoie à nouveau le SET_SSV original. Si le SET_SSV original n'a pas été exécuté, alors le serveur l'exécute. Si le SET_SSV original a été exécuté mais a échoué, le serveur va retourner le SET_SSV à partir de la mémoire tampon de réponse.

18.35 Opération 42 : EXCHANGE_ID - instancie l'identifiant de client

L'opération EXCHANGE_ID échange les identifiants de client et serveur à long terme (propriétaires) et fournit l'accès à un identifiant de client, en créant un si nécessaire. Cet identifiant de client devient associé à la connexion sur laquelle l'opération est faite, afin d'être disponible quand un CREATE_SESSION est fait ou quand la connexion est utilisée pour produire une demande sur une session existante associée au client actuel.

18.35.1 Argument

```
const EXCHGID4_FLAG_SUPP_MOVED_REFER      = 0x00000001;
const EXCHGID4_FLAG_SUPP_MOVED_MIGR       = 0x00000002;
const EXCHGID4_FLAG_BIND_PRINC_STATEID    = 0x00000100;
const EXCHGID4_FLAG_USE_NON_PNFS          = 0x00010000;
const EXCHGID4_FLAG_USE_PNFS_MDS          = 0x00020000;
const EXCHGID4_FLAG_USE_PNFS_DS           = 0x00040000;
const EXCHGID4_FLAG_MASK_PNFS             = 0x00070000;
const EXCHGID4_FLAG_UPD_CONFIRMED_REC_A = 0x40000000;
const EXCHGID4_FLAG_CONFIRMED_R           = 0x80000000;

struct state_protect_ops4 {
        bitmap4 spo_must_enforce;
        bitmap4 spo_must_allow;
  };

struct ssv_sp_parms4 {
        state_protect_ops4   ssp_ops;
        sec_oid4             ssp_hash_algs<>;
        sec_oid4             ssp_encr_algs<>;
        uint32_t             ssp_window;
        uint32_t             ssp_num_gss_handles;
};

enum state_protect_how4 {
        SP4_NONE = 0,
        SP4_MACH_CRED = 1,
        SP4_SSV = 2
};

union state_protect4_a switch (state_protect_how4 spa_how) {
      cas SP4_NONE : vide
      cas SP4_MACH_CRED : state_protect_ops4    spa_mach_ops;
      cas SP4_SSV : ssv_sp_parms4               spa_ssv_parms;
};

struct EXCHANGE_ID4args {
      client_owner4        eia_clientowner;
      uint32_t             eia_flags;
      state_protect4_a     eia_state_protect;
      nfs_impl_id4         eia_client_impl_id<1>;
};
```

18.35.2 Résultat

```
struct ssv_prot_info4 {
   state_protect_ops4   spi_ops;
   uint32_t             spi_hash_alg;
   uint32_t             spi_encr_alg;
   uint32_t             spi_ssv_len;
```

```
   uint32_t                spi_window;
   gsshandle4_t            spi_handles<>;
};

union state_protect4_r switch (state_protect_how4 spr_how) {
   cas SP4_NONE : vide
   cas SP4_MACH_CRED : state_protect_ops4     spr_mach_ops;
   cas SP4_SSV : ssv_prot_info4               spr_ssv_info;
  };

struct EXCHANGE_ID4resok {
   clientid4               eir_clientid;
   sequenceid4             eir_sequenceid;
   uint32_t                eir_flags;
   state_protect4_r        eir_state_protect;
   server_owner4           eir_server_owner;
   opaque                  eir_server_scope<NFS4_OPAQUE_LIMIT>;
   nfs_impl_id4            eir_server_impl_id<1>;
  };

union EXCHANGE_ID4res switch (nfsstat4 eir_status) {
  cas NFS4_OK : EXCHANGE_ID4resok      eir_resok4;
  défaut : vide
};
```

18.35.3 Description

Le client utilise l'opération EXCHANGE_ID pour enregistrer une instance particulière de ce client auprès du serveur, comme représenté par un client_owner4. Cependant, quand le client_owner4 a déjà été enregistré par d'autres moyens (par exemple, migration d'état transparente) le client peut encore utiliser EXCHANGE_ID pour obtenir l'identifiant de client alloué précédemment.

L'identifiant de client retourné de cette opération va être associé à la connexion sur laquelle le EXCHANGE_ID est reçu et va servir d'objet parent pour les sessions créées par le client sur cette connexion ou auxquelles la connexion est liée. Par suite de l'utilisation de ces sessions pour faire des demandes impliquant la création d'état, cet état va devenir associé à l'identifiant de client retourné.

Dans les situations dans lesquelles l'enregistrement du client_owner n'a pas été fait précédemment, l'identifiant de client doit d'abord être utilisé, avec le eir_sequenceid retourné, à la création d'une session associée en utilisant CREATE_SESSION.

Si le fanion EXCHGID4_FLAG_CONFIRMED_R est établi dans le résultat, eir_flags, c'est alors l'indication que l'enregistrement du client_owner s'est déjà produit et qu'un autre CREATE_SESSION n'est pas nécessaire pour le confirmer. Bien sûr, des opérations CREATE_SESSION suivantes peuvent être nécessaires pour d'autres raisons.

La valeur eir_sequenceid est utilisée pour établir une valeur de séquence initiale associée à l'identifiant de client retourné. Dans les cas où un CREATE_SESSION a déjà été fait, il n'y a pas besoin de cette valeur, car le séquençage de ces demandes a déjà été établi, et le client n'a pas besoin de cette valeur et va l'ignorer.

EXCHANGE_ID PEUT être envoyé dans une procédure COMPOUND qui commence avec SEQUENCE. Cependant, quand un client communique avec un serveur pour la première fois, il ne va pas avoir de session, donc l'utilisation de SEQUENCE ne va pas être possible. Si EXCHANGE_ID est envoyé sans une SEQUENCE précédente, alors ce DOIT être la seule opération dans la demande de procédure COMPOUND. Si elle ne l'est pas, le serveur DOIT retourner NFS4ERR_NOT_ONLY_OP.

Le champ eia_clientowner est composé d'un champ co_verifier et d'une chaîne co_ownerid. Comme noté au paragraphe 2.4, le co_ownerid identifie le client, et le co_verifier spécifie une incarnation particulière de ce client. Un EXCHANGE_ID envoyé avec une nouvelle incarnation du client va conduire le serveur à supprimer l'état de verrouillage de l'ancienne incarnation. Par ailleurs, quand un EXCHANGE_ID envoyé avec l'incarnation et le co_ownerid courant ne résulte pas en une erreur sans relation, il va potentiellement mettre à jour les propriétés d'un identifiant de client existant ou

simplement retourner des informations sur le client_id existant. Cette dernière situation va se produire quand cette opération est faite sur le même serveur en utilisant des adresses réseau différentes au titre de la création de connexions tronquées.

Un serveur NE DOIT PAS fournir le même identifiant de client à deux différentes incarnations d'un eia_clientowner.

En plus de l'identifiant de client et de l'identifiant de séquence, le serveur retourne un propriétaire de serveur (eir_server_owner) et une portée de serveur (eir_server_scope). Le premier champ est utilisé en connexion avec le multiplexage de réseau comme décrit au paragraphe 2.10.5. Le dernier champ est utilisé pour permettre aux clients de déterminer quand les identifiants de client envoyés par un serveur peuvent être reconnus par un autre en cas de migration de système de fichiers (voir le paragraphe 11.11.9 du présent document).

L'identifiant de client retourné par EXCHANGE_ID est seulement unique par rapport à la combinaison de eir_server_owner.so_major_id et eir_server_scope. Donc, si deux serveurs retournent le même identifiant de client, la charge de distinguer les identifiants de client sur la base de leurs eir_server_owner.so_major_id et eir_server_scope revient au client. Dans le cas de deux différents serveurs qui revendiquent des server_owner.so_major_id et eir_server_scope correspondants, le client peut utiliser les techniques de vérification discutées au paragraphe 2.10.5.1 pour déterminer si les serveurs sont distincts. Si ils sont distincts, alors le client va avoir besoin de noter les adresses réseau de destination des connexions utilisées avec chaque serveur et d'utiliser l'adresse réseau comme discriminant final.

Le serveur, comme défini par l'identité unique exprimée dans le so_major_id du propriétaire de serveur et la portée de serveur, a besoin de retracer plusieurs propriétés de chaque identifiant de client qu'il détient. Les propriétés s'appliquent à l'identifiant de client et à toutes les sessions associées à l'identifiant de client. Les propriétés sont déduites des arguments et résultats de EXCHANGE_ID. Les propriétés d'identifiant de client incluent :

* Les capacités exprimées par les bits suivants, qui viennent des résultats de EXCHANGE_ID :
  - EXCHGID4_FLAG_SUPP_MOVED_REFER
  - EXCHGID4_FLAG_SUPP_MOVED_MIGR
  - EXCHGID4_FLAG_BIND_PRINC_STATEID
  - EXCHGID4_FLAG_USE_NON_PNFS
  - EXCHGID4_FLAG_USE_PNFS_MDS
  - EXCHGID4_FLAG_USE_PNFS_DS

  Ces propriétés peuvent être mises à jour par des opérations EXCHANGE_ID suivantes sur les identifiants de client confirmés bien que le serveur PUISSE refuser de les changer.

* La méthode de protection d'état utilisée, une de SP4_NONE, SP4_MACH_CRED, ou SP4_SSV, comme réglé par le champ spa_how des arguments à EXCHANGE_ID. Une fois que l'identifiant de client est confirmé, cette propriété ne peut pas être mise à jour par des opérations EXCHANGE_ID suivantes.

* Pour la protection d'état SP4_MACH_CRED ou SP4_SSV :
  - La liste des opérations (spo_must_enforce) qui DOIVENT utiliser la protection d'état spécifiée. Cette liste vient des résultats de EXCHANGE_ID.
  - La liste des opérations (spo_must_allow) qui PEUVENT utiliser la protection d'état spécifiée. Cette liste vient des résultats de EXCHANGE_ID.

  Une fois l'identifiant de client confirmé, ces propriétés ne peuvent pas être mises à jour par des demandes EXCHANGE_ID ultérieures.

* Pour la protection SP4_SSV :
  - L'OID de l'algorithme de hachage. Cette propriété est représentée par un des algorithmes dans le champ ssp_hash_algs des arguments de EXCHANGE_ID. Une fois que l'identifiant de client est confirmé, cette propriété ne peut pas être mise à jour par les demandes EXCHANGE_ID suivantes.
  - L'OID de l'algorithme de chiffrement. Cette propriété est représentée par un des algorithmes dans le champ ssp_encr_algs des arguments de EXCHANGE_ID. Une fois que l'identifiant de client est confirmé, cette propriété ne peut pas être mise à jour par les demandes EXCHANGE_ID suivantes.
  - La longueur du SSV. Cette propriété est représentée par le champ spi_ssv_len dans le résultat de EXCHANGE_ID. Une fois que l'identifiant de client est confirmé, cette propriété ne peut pas être mise à jour par les opérations EXCHANGE_ID suivantes. Il y a des relations EXIGÉ et RECOMMANDÉ entre la longueur de la clé de l'algorithme de chiffrement ("longueur de clé") la longueur du résultat de l'algorithme de hachage ("longueur de hachage") et la longueur du SSV ("longueur de SSV").
    - o longueur de clé DOIT être ≤ longueur de hachage. C'est parce que les clés utilisées pour l'algorithme de chiffrement sont en fait des sous clés déduites du SSV, et leur déduction est via l'algorithme de hachage. Le choix

d'un algorithme de chiffrement avec une longueur de clé qui excède la longueur du résultat de l'algorithme de hachage exigerait du bourrage, et donc affaiblirait l'utilisation de l'algorithme de chiffrement.
  - o longueur de hachage DEVRAIT être ≤ longueur de SSV. C'est parce que le SSV est une clé utilisée pour déduire les sous clés via un HMAC, et il est recommandé que la clé utilisée comme entrée à un HMAC soit au moins aussi longue que le résultat de l'algorithme de hachage de HMAC (voir la Section 3 de la [RFC2104]).
  - o longueur de clé DEVRAIT être ≤ longueur de SSV. C'est un résultat transitif des deux invariants ci-dessus.
  - o longueur de clé DEVRAIT être ≥ longueur de hachage / 2. C'est parce que la déduction de sous clé est via un HMAC et il est recommandé que si le HMAC doit être tronqué, il ne devrait pas être tronqué à moins de la moitié de la longueur de hachage (voir la Section 4 de la [RFC2104]).
- Le nombre de versions concurrentes de SSV que le client et le serveur vont prendre en charge (voir le paragraphe 2.10.9). Cette propriété est représentée par spi_window dans le résultat de EXCHANGE_ID. La propriété peut être mise à jour par des opérations EXCHANGE_ID suivantes.

* L'identifiant de mise en œuvre de client comme représenté par le champ eia_client_impl_id des arguments. La propriété peut être mise à jour par des demandes EXCHANGE_ID suivantes.

* L'identifiant de mise en œuvre de serveur comme représenté par le champ eir_server_impl_id de la réponse. La propriété peut être mise à jour par les réponses aux demandes EXCHANGE_ID suivantes.

Les eia_flags passés au titre des arguments et les résultats de eir_flags permettent au client et serveur de s'informer l'un l'autre de leurs capacités ainsi que d'indiquer comment l'identifiant de client va être utilisé. Qu'un bit soit établi ou à zéro sur les fanions d'arguments ne force pas le serveur à établir ou supprimer le même bit sur le côté des résultats. Les bits non définis ci-dessus ne peuvent pas être établis dans le champ eia_flags. Si il le sont, le serveur DOIT rejeter l'opération avec NFS4ERR_INVAL.

Le bit EXCHGID4_FLAG_UPD_CONFIRMED_REC_A peut seulement être établi dans eia_flags ; il est toujours à zéro dans eir_flags. Le bit EXCHGID4_FLAG_CONFIRMED_R peut seulement être établi dans eir_flags ; il est toujours à zéro dans eia_flags. Si le serveur reconnaît les co_ownerid et co_verifier comme des transpositions d'un identifiant de client confirmé, il établit EXCHGID4_FLAG_CONFIRMED_R dans eir_flags. Le fanion EXCHGID4_FLAG_CONFIRMED_R permet à un client de dire si l'identifiant de client qu'il essaye de créer existe déjà et est confirmé.

Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A est établi dans eia_flags, cela signifie que le client tente de mettre à jour les propriétés d'un identifiant de client existant confirmé (si le client veut mettre à jour les propriétés d'un identifiant de client non confirmé, il NE DOIT PAS établir EXCHGID4_FLAG_UPD_CONFIRMED_REC_A). Si il en est ainsi, il est RECOMMANDÉ que le client envoie l'opération de mise à jour de EXCHANGE_ID dans le même COMPOUND qu'un SEQUENCE afin que le EXCHANGE_ID soit exécuté exactement une fois. Si le client peut mettre à jour les propriétés de l'identifiant de client dépend de la protection d'état qu'il a choisi quand l'identifiant de client a été créé, et du principal et de la nuance de sécurité qu'il a utilisé quand il a envoyé l'opération EXCHANGE_ID. Les situations décrites aux points 6, 7, 8, ou 9 de la seconde liste numérotée du paragraphe 18.35.4 ci-dessous vont s'appliquer. Noter que si l'opération réussit et retourne un identifiant de client qui est déjà confirmé, le serveur DOIT établir le bit EXCHGID4_FLAG_CONFIRMED_R dans eir_flags.

Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A n'est pas établi dans eia_flags, cela signifie que le client essaye d'établir un nouvel identifiant de client ; il tente de tronquer la communication de données avec le serveur (voir le paragraphe 2.10.5) ; ou il tente de mettre à jour les propriétés d'un identifiant de client non confirmé. Les situations décrites aux points 1, 2, 3, 4, ou 5 de la seconde liste numérotée du paragraphe 18.35.4 vont s'appliquer. Noter que si l'opération réussit et retourne un identifiant de client qui était précédemment confirmé, le serveur DOIT établir le bit EXCHGID4_FLAG_CONFIRMED_R dans eir_flags.

Quand le bit du fanion EXCHGID4_FLAG_SUPP_MOVED_REFER est établi, le client indique qu'il est capable de traiter une erreur NFS4ERR_MOVED au titre d'une séquence d'orientation documentaire. Quand ce bit n'est pas établi, il est encore légal que le serveur effectue une séquence d'orientation documentaire. Cependant, un serveur peut utiliser le fait que le client est incapable de répondre correctement à une orientation documentaire, en l'évitant pour ce client particulier. Il peut, par exemple, agir comme un mandataire pour ce système de fichiers particulier, au prix de moindres performances, bien qu'il ne soit pas obligé de le faire. Si le serveur va potentiellement effectuer une orientation documentaire, il DOIT établir EXCHGID4_FLAG_SUPP_MOVED_REFER dans eir_flags.

Quand le bit EXCHGID4_FLAG_SUPP_MOVED_MIGR est établi, le client indique qu'il est capable de traiter une erreur NFS4ERR_MOVED au titre d'une séquence de migration d'un système de fichiers. Quand ce bit n'est pas établi, il est encore légal pour le serveur d'indiquer qu'un système de fichiers s'est déplacé, quand cela arrive en fait. Cependant, un serveur peut utiliser le fait que le client est incapable de répondre correctement à une migration dans sa programmation de

migration de systèmes de fichiers de façon à éviter que la migration de systèmes de fichiers soit activement utilisée. Il peut aussi cacher les migrations réelles aux clients incapables de les traiter en agissant comme un mandataire pour un système de fichiers migré pour des clients particuliers, au prix de moindres performances, bien qu'il ne soit pas obligé de faire cela. Si le serveur va potentiellement effectuer une migration, il DOIT établir EXCHGID4_FLAG_SUPP_MOVED_MIGR dans eir_flags.

Quand le bit EXCHGID4_FLAG_BIND_PRINC_STATEID est établi, le client indique qu'il veut que le serveur lie l'identifiant d'état au principal. Cela signifie que quand un principal crée un identifiant d'état, il doit être celui qui utilise l'identifiant d'état. Si le serveur va effectuer un lien, il va retourner EXCHGID4_FLAG_BIND_PRINC_STATEID. Le serveur PEUT retourner EXCHGID4_FLAG_BIND_PRINC_STATEID même si le client ne le demande pas. Si une mise à jour de l'identifiant de client change la valeur de la propriété d'identifiant de client de EXCHGID4_FLAG_BIND_PRINC_STATEID, l'effet s'applique seulement aux nouveaux identifiants d'état. Les identifiants d'état existants (et tous les identifiants d'état avec le même champ "autre") qui ont été créés avec un lien d'identifiant d'état à principal en vigueur vont continuer d'avoir ce lien en vigueur. Les identifiants d'état existants (et tous les identifiants d'état avec le même champ "autre") qui ont été créés sans lien d'identifiant d'état à principal vont continuer de ne pas être liés.

Les bits EXCHGID4_FLAG_USE_NON_PNFS, EXCHGID4_FLAG_USE_PNFS_MDS, et EXCHGID4_FLAG_USE_PNFS_DS sont décrits au paragraphe 13.1 et portent les rôles pour lesquels l'identifiant de client va être utilisé dans un environnement pNFS. Le serveur DOIT établir une des combinaisons acceptables de ces bits (rôles) dans eir_flags, comme spécifié dans ce paragraphe. Note que la même paire de propriétaire de client/propriétaire de serveur peut avoir plusieurs rôles. Plusieurs rôles peuvent être associés au même identifiant de client ou avec des identifiants de client différents. Donc, si un client envoie EXCHANGE_ID à partir du même propriétaire de client au même propriétaire de serveur plusieurs fois, mais spécifie des rôles pNFS différents chaque fois, le serveur pourrait retourner des identifiants de client différents. Étant donné que différents rôles pNFS pourraient avoir des identifiants de client différents, le client peut demander des propriétés différentes pour chaque rôle/identifiant de client.

Le champ spa_how du champ eia_state_protect spécifie comment le client veut protéger ses états de client, verrouillage, et session contre des changements non autorisés (paragraphe 2.10.8.3) :

* SP4_NONE. Le client ne demande pas au serveur NFSv4.1 d'appliquer de protection d'état. Le serveur NFSv4.1 NE DOIT PAS appliquer de protection d'état pour l'identifiant de client retourné.

* SP4_MACH_CRED. Si spa_how est SP4_MACH_CRED, alors le client DOIT envoyer l'opération EXCHANGE_ID avec RPCSEC_GSS comme nuance de sécurité, et avec un service de RPC_GSS_SVC_INTEGRITY ou RPC_GSS_SVC_PRIVACY. Si SP4_MACH_CRED est spécifié, alors le client veut utiliser un accréditif de machine fondé sur RPCSEC_GSS pour protéger son état. Le serveur DOIT noter le principal avec lequel l'opération EXCHANGE_ID a été envoyée, et le mécanisme GSS utilisé. Ces notes comportent collectivement l'accréditif de machine.

  Après que l'identifiant de client est confirmé, tant que le prêt associé à l'identifiant de client n'est pas expiré, une opération EXCHANGE_ID suivante qui utilise le même eia_clientowner.co_owner que la première EXCHANGE_ID DOIT aussi utiliser le même accréditif de machine que la première EXCHANGE_ID. Le serveur retourne pour l'opération EXCHANGE_ID suivante le même identifiant de client que retourné de la première EXCHANGE_ID.

* SP4_SSV. Si spa_how est SP4_SSV, alors le client DOIT envoyer l'opération EXCHANGE_ID avec RPCSEC_GSS comme nuance de sécurité, et avec un service de RPC_GSS_SVC_INTEGRITY ou RPC_GSS_SVC_PRIVACY. Si SP4_SSV est spécifié, alors le client veut utiliser le SSV pour protéger son état. Le serveur enregistre l'accréditif utilisé dans la demande comme accréditif de machine (comme défini ci-dessus) pour le eia_clientowner.co_owner. L'opération CREATE_SESSION qui confirme l'identifiant de client DOIT utiliser le même accréditif de machine.

Quand un client spécifie SP4_MACH_CRED ou SP4_SSV, il fournit aussi deux listes d'opérations (chacune exprimée comme un gabarit binaire). La première liste est spo_must_enforce et consiste en les opérations que le client DOIT envoyer (sous réserve de la confirmation par le serveur de la liste des opérations dans le résultat de EXCHANGE_ID) avec l'accréditif de machine (si la protection SP4_MACH_CRED est spécifiée) ou l'accréditif fondé sur le SSV (si la protection SP4_SSV est utilisée). Le client DOIT envoyer les opérations avec les accréditifs RPCSEC_GSS qui spécifient le service de sécurité RPC_GSS_SVC_INTEGRITY ou RPC_GSS_SVC_PRIVACY. Normalement, la première liste d'opérations inclut EXCHANGE_ID, CREATE_SESSION, DELEGPURGE, DESTROY_SESSION, BIND_CONN_TO_SESSION, et DESTROY_CLIENTID. Le client NE DEVRAIT PAS spécifier dans cette liste d'opérations qui exigent une bride de fichier parce que les politiques d'accès du serveur PEUVENT être en conflit avec le choix du client, et donc le client serait alors incapable d'accéder à un sous ensemble de l'espace de noms du serveur.

Noter que si la protection SP4_SSV est spécifiée, et si le client indique que CREATE_SESSION doit être protégé avec SP4_SSV, parce que le SSV ne peut pas exister sans un identifiant de client confirmé, la première CREATE_SESSION DOIT être envoyée à la place en utilisant l'accréditif de machine, et le serveur DOIT accepter l'accréditif de machine.

Il y a un résultat correspondant, aussi appelé spo_must_enforce, des opérations pour lesquelles le serveur va demander la protection SP4_MACH_CRED ou SP4_SSV. Normalement, le résultat du serveur est égal à l'argument du client, mais le résultat PEUT être différent. Si le client demande une ou plusieurs opérations dans l'ensemble { EXCHANGE_ID, CREATE_SESSION, DELEGPURGE, DESTROY_SESSION, BIND_CONN_TO_SESSION, DESTROY_CLIENTID }, alors le résultat spo_must_enforce DOIT inclure les opérations de cet ensemble que le client a demandées.

Si spo_must_enforce dans le résultat a BIND_CONN_TO_SESSION établi, alors l'application du lien de connexion est activée, et le client DOIT utiliser l'accréditif de machine (si la protection SP4_MACH_CRED est utilisée) ou de SSV (si la protection SP4_SSV est utilisée) sur les appels à BIND_CONN_TO_SESSION.

La seconde liste est spo_must_allow et consiste en les opérations que le client veut avoir l'option d'envoyer avec l'accréditif de machine ou l'accréditif fondé sur SSV, même si l'objet sur lequel les opérations sont effectuées n'est pas la propriété de l'accréditif de machine ou de SSV.

Le résultat correspondant, aussi appelé spo_must_allow, consiste en les opérations que le serveur va permettre au client d'utiliser avec les accréditifs SP4_SSV ou SP4_MACH_CRED. Normalement, le résultat du serveur est égal à l'argument du client, mais le résultat PEUT être différent.

L'objet de spo_must_allow est de permettre aux clients de résoudre l'énigme suivante. Supposons que l'identifiant de client soit confirmé avec EXCHGID4_FLAG_BIND_PRINC_STATEID, et qu'il invoque OPEN avec les accréditifs RPCSEC_GSS d'un utilisateur normal. Maintenant supposons que les accréditifs de l'utilisateur expirent, et ne puissent pas être renouvelés (par exemple, un ticket Kerberos accordant un ticket expire, et l'utilisateur s'est déconnecté et ne va pas acquérir un nouveau ticket d'accord de ticket). Le client va être incapable d'envoyer un CLOSE sans les accréditifs de l'utilisateur, ce qui revient à dire que le client doit soit quitter l'état sur le serveur, soit renvoyer un EXCHANGE_ID avec un nouveau vérificateur pour supprimer tout état, c'est-à-dire sauf si le client inclut CLOSE dans la liste d'opérations dans spo_must_allow et si le serveur l'accepte.

Les paramètres de protection SP4_SSV ont aussi :

ssp_hash_algs : c'est l'ensemble d'algorithmes que le client prend en charge pour les besoins du calcul des résumés nécessaires pour le mécanisme GSS de SSV interne et pour l'opération SET_SSV. Chaque algorithme est spécifié comme un identifiant d'objet (OID). Les algorithmes EXIGÉS pour un serveur sont id-sha1, id-sha224, id-sha256, id-sha384, et id-sha512 [RFC4055]. Du fait des faiblesses connues de id-sha1, il est RECOMMANDÉ que le client spécifie au moins un algorithme dans ssp_hash_algs autre que id-sha1. L'algorithme que choisit le serveur dans l'ensemble est indiqué dans spi_hash_alg, un champ de spr_ssv_prot_info. Le champ spi_hash_alg est un index dans le dispositif ssp_hash_algs. À cause des faiblesses connues de id-sha1, il est RECOMMANDÉ qu'il ne soit pas choisi par le serveur tant que ssp_hash_algs contient d'autres algorithmes pris en charge. Si le serveur ne prend en charge aucun des algorithmes offerts, il retourne NFS4ERR_HASH_ALG_UNSUPP. Si ssp_hash_algs est vide, le serveur DOIT retourner NFS4ERR_INVAL.

ssp_encr_algs : c'est l'ensemble d'algorithmes que le client prend en charge pour les besoins de la protection de la confidentialité pour le mécanisme GSS de SSV interne. Chaque algorithme est spécifié comme un OID. L'algorithme EXIGÉ pour un serveur est id-aes256-CBC. Les algorithmes RECOMMANDÉS sont id-aes192-CBC et id-aes128-CBC [NIST]. L'algorithme choisi est retourné dans spi_encr_alg, un index dans ssp_encr_algs. Si le serveur ne prend en charge aucun des algorithmes offerts, il retourne NFS4ERR_ENCR_ALG_UNSUPP. Si ssp_encr_algs est vide, le serveur DOIT retourner NFS4ERR_INVAL. Noter que du fait des exigences et recommandations précédemment mentionnées sur les relations entre longueur de clé et longueur de hachage, certaines combinaisons d'algorithme de chiffrement et algorithme de hachage RECOMMANDÉS et EXIGÉS NE DEVRAIENT PAS ou NE DOIVENT PAS être utilisées. Le Tableau 21 résume les combinaisons illégales et déconseillées.

ssp_window : c'est le nombre de versions de SSV que le client veut que maintienne le serveur (c'est-à-dire, chaque appel réussi à SET_SSV produit une nouvelle version du SSV). Si ssp_window est zéro, le serveur DOIT retourner NFS4ERR_INVAL. Le serveur répond avec spi_window, qui NE DOIT PAS excéder ssp_window et DOIT être au moins un. Toutes les demandes sur le canal de retour ou le canal avant qui utilisent une version du SSV en dehors de la fenêtre vont échouer avec une erreur d'authentification ONC RPC, et le demandeur va devoir les réessayer avec le même identifiant de créneau et identifiant de séquence.

ssp_num_gss_handles : c'est le nombre de brides RPCSEC_GSS que le serveur devrait créer fondées sur le mécanisme de GSS de SSV (voir le paragraphe 2.10.9). Ce n'est pas le nombre total de brides RPCSEC_GSS pour l'identifiant de client. Bien sûr, les appels suivants à EXCHANGE_ID vont ajouter des brides RPCSEC_GSS. Le serveur répond avec une liste des brides dans spi_handles. Si le client demande au moins une bride et si le serveur ne peut pas la créer, le serveur DOIT retourner une erreur. Les brides dans spi_handles ne sont pas disponibles à l'utilisation tant que l'identifiant de client n'est pas confirmé, ce qui pourrait être immédiatement si EXCHANGE_ID retourne EXCHGID4_FLAG_CONFIRMED_R, ou sur la confirmation de succès de CREATE_SESSION.

Alors qu'un identifiant de client peut s'étendre sur toutes les connexions qui sont connectées à un serveur partageant le même eir_server_owner.so_major_id, les brides RPCSEC_GSS retournées dans spi_handles peuvent seulement être utilisées sur des connexions connectées à un serveur qui retourne le même eir_server_owner.so_major_id et eir_server_owner.so_minor_id sur chaque connexion. Il est permis au client de régler ssp_num_gss_handles à zéro ; le client peut créer plus de brides avec un autre appel EXCHANGE_ID.

Parce que chaque bride RPCSEC_GSS de SSV partage un contexte GSS de SSV commun, il y a des considérations de sécurité spécifiques de cette situation discutées au paragraphe 2.10.10.

La seq_window (voir le paragraphe 5.2.3.1 de la [RFC2203]) de chaque bride RPCSEC_GSS dans spi_handle DOIT être la même que la seq_window de la bride RPCSEC_GSS utilisée pour l'accréditif de la demande RPC d'où l'opération EXCHANGE_ID a été envoyée.

Algorithme de chiffrement	NE DOIT PAS être combiné avec	NE DEVRAIT PAS être combiné avec
id-aes128-CBC		id-sha384, id-sha512
id-aes192-CBC	id-sha1	id-sha512
id-aes256-CBC	id-sha1, id-sha224	

Tableau 21

Les arguments incluent un dispositif de jusqu'à un élément de longueur appelé eia_client_impl_id. Si eia_client_impl_id est présent, il contient les informations d'identification de la mise en œuvre de client. De même, le résultat inclut un dispositif de jusqu'à un élément de longueur appelé eir_server_impl_id qui identifie la mise en œuvre du serveur. Les serveurs DOIVENT accepter un dispositif eia_client_impl_id de longueur zéro, et les clients DOIVENT accepter un dispositif eir_server_impl_id array de longueur zéro.

Une utilisation possible des identifiants de mise en œuvre serait dans un logiciel de diagnostic qui extrait ces informations en tentant d'identifier les problèmes d'interopérabilité, les comportements de performance de la charge de travail, ou des statistiques générales d'usage. Comme l'intention d'avoir accès à ces informations est seulement pour la planification ou le diagnostic général, le client et le serveur NE DOIVENT PAS interpréter ces informations d'identité de mise en œuvre d'une façon qui affecte l'interaction de la mise en œuvre avec son homologue. Le client et le serveur ne doivent pas dépendre de ce que l'homologue manifeste un comportement particulier permis sur la base d'un identifiant de mise en œuvre mais il est exigé qu'ils inter-opèrent comme spécifié ailleurs dans la spécification du protocole.

Parce qu'il est possible que certaines mises en œuvre violent la spécification de protocole et interprètent les informations d'identité, les mises en œuvre DOIVENT fournir des facilités pour permettre au client et serveur NFSv4 d'être configurés à régler le contenu des structures nfs_impl_id envoyées à toute valeur spécifiée.

18.35.4 Mise en œuvre

L'enregistrement de client d'un serveur est un quintuplé :

1. co_ownerid : chaîne d'identifiant de client, provenant de la structure eia_clientowner de la structure EXCHANGE_ID4args.
2. co_verifier : valeur spécifique du client utilisée pour indiquer les incarnations (où un redémarrage de client représente une nouvelle incarnation) provenant de la structure eia_clientowner de la structure EXCHANGE_ID4args.
3. principal : le principal qui était défini dans l'accréditif et/ou vérificateur de l'en-tête RPC au moment où l'enregistrement de client a été établi.
4. identifiant de client : abrégé de l'identifiant de client, généré par le serveur et retourné via le champ eir_clientid dans la structure EXCHANGE_ID4resok.
5. confirmed : champ privé sur le serveur qui indique si un enregistrement de client a été confirmé ou non. Un enregistrement de client est confirmé si il y a eu une opération CREATE_SESSION réussie pour le confirmer.

Autrement, il est non confirmé. Un enregistrement non confirmé est établi par un appel EXCHANGE_ID. Tout enregistrement non confirmé qui n'est pas confirmé sur une période de prêt DEVRAIT être supprimé.

Les identifiants suivants représentent des valeurs spéciales pour les champs dans les enregistrements.

ownerid_arg : valeur du sous champ eia_clientowner.co_ownerid de la structure EXCHANGE_ID4args de la demande en cours.

verifier_arg : valeur du sous champ eia_clientowner.co_verifier de la structure EXCHANGE_ID4args de la demande en cours.

old_verifier_arg : valeur du champ eia_clientowner.co_verifier d'un enregistrement de client reçu dans une demande précédente ; c'est distinct de verifier_arg.

principal_arg : valeur du principal RPCSEC_GSS pour la demande en cours.

old_principal_arg : valeur du principal d'un enregistrement de client comme défini par l'accréditif ou vérificateur de l'en-tête RPC d'une demande précédente. C'est distinct de principal_arg.

clientid_ret : valeur du champ eir_clientid que le serveur va retourner dans la structure EXCHANGE_ID4resok pour la demande en cours.

old_clientid_ret : valeur du champ eir_clientid que le serveur a retourné dans la strucutre EXCHANGE_ID4resok pour une demande précédente. C'est distinct de clientid_ret.

confirmed : l'identifiant de client a été confirmé.

unconfirmed : l'identifiant de client n'a pas été confirmé.

Comme EXCHANGE_ID est une opération non idempotente, on doit considérer la possibilité que un nouvel essai se produise par suite d'un redémarrage de client, d'une partition de réseau, d'un mauvais fonctionnement de routeur, etc. Les réessais sont identifiés par la valeur du champ eia_clientowner de EXCHANGE_ID4args, et la méthode pour les traiter est présentée dans les scénarios qui suivent.

Les scénarios sont décrits en termes d'enregistrement de client qu'a un serveur pour un certain co_ownerid. Noter que si l'identifiant de client a été créé en spécifiant la protection d'état SP4_SSV et un EXCHANGE_ID comme celui des opérations dans spo_must_allow, alors le serveur DOIT autoriser les EXCHANGE_ID avec le principal de SSV en plus du principal qui a créé l'identifiant de client.

1. Nouvel identifiant de propriétaire
   Si le serveur n'a pas d'enregistrement de client avec un eia_clientowner.co_ownerid correspondant à ownerid_arg, et si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A n'est pas établi dans EXCHANGE_ID, alors un nouvel identifiant de client abrégé (appelons le clientid_ret) est généré, et l'enregistrement non confirmé suivant est ajouté à l'état du serveur.

   { ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

   Ensuite, le serveur retourne clientid_ret.

2. Non mise à jour sur l'identifiant de client existant
   Si le serveur a l'enregistrement confirmé suivant, et si la demande n'a pas établi EXCHGID4_FLAG_UPD_CONFIRMED_REC_A, alors la demande est le résultat d'une demande réessayée due à un routeur fautif ou à une connexion perdue, ou le client essaye de déterminer si il peut effectuer un multiplexage.

   { ownerid_arg, verifier_arg, principal_arg, clientid_ret, confirmed }

   Comme l'enregistrement a été confirmé, le client doit avoir reçu la réponse du serveur à la demande EXCHANGE_ID initiale. Comme le serveur a un enregistrement confirmé, et comme EXCHGID4_FLAG_UPD_CONFIRMED_REC_A n'est pas établi, avec l'exception possible de eir_server_owner.so_minor_id, le serveur retourne le même résultat que quand les propriétés de l'identifiant de client ont été mises à jour pour la dernière fois (ou si elles n'ont jamais été mises à jour, le résultat quand l'identifiant de client a été créé). L'enregistrement confirmé est inchangé.

3. Collision de client
   Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A n'est pas établi, et si le serveur a l'enregistrement suivant confirmé, alors cette demande est probablement le résultat d'une collision par hasard entre les valeurs du sous champ eia_clientowner.co_ownerid de EXCHANGE_ID4args pour deux clients différents.

   { ownerid_arg, *, old_principal_arg, old_clientid_ret, confirmed }

   Si il n'y a actuellement aucun état associé à old_clientid_ret, ou si il y a un état mais le prêt a expiré, alors ce cas est effectivement équivalent au cas Nouvel identifiant de propriétaire du paragraphe 18.35.4, alinéa 7, point 1. L'enregistrement confirmé est supprimé, le old_clientid_ret et son état de verrouillage sont supprimés, un nouvel identifiant de client abrégé est généré, et l'enregistrement non confirmé suivant est ajouté à l'état du serveur.

   { ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

   Ensuite, le serveur retourne clientid_ret.

   Si old_clientid_ret a un prêt non expiré avec état, alors aucun état de old_clientid_ret n'est changé ni supprimé. Le serveur retourne NFS4ERR_CLID_INUSE pour indiquer que le client devrait réessayer avec une valeur différente pour le sous champ eia_clientowner.co_ownerid de EXCHANGE_ID4args. L'enregistrement de client n'est pas changé.

4. Remplacement d'enregistrement non confirmé
   Si le fanion EXCHGID4_FLAG_UPD_CONFIRMED_REC_A n'est pas établi, et si le serveur a l'enregistrement non confirmé suivant, alors le client tente à nouveau EXCHANGE_ID sur un identifiant de client non confirmé, peut-être du fait d'une nouvelle tentative, d'un redémarrage de client avant la confirmation de l'identifiant de client (c'est-à-dire, avant que CREATE_SESSION soit invoqué) ou quelque autre raison.

   { ownerid_arg, *, *, old_clientid_ret, unconfirmed }

   Il est possible que les propriétés de old_clientid_ret soient différentes de celles spécifiées dans le EXCHANGE_ID courant. Que les propriétés soient ou non mises à jour, pour éliminer toute ambiguïté, le serveur supprime l'enregistrement non confirmé, génère un nouvel identifiant de client (clientid_ret) et établit l'enregistrement non confirmé suivant :

   { ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

5. Redémarrage du client
   Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A n'est pas établi, et si le serveur a l'enregistrement de client confirmé suivant, alors cette demande provient probablement d'un client précédemment confirmé qui a redémarré.

   { ownerid_arg, old_verifier_arg, principal_arg, old_clientid_ret, confirmed }

   Comme la précédente incarnation du même client ne va plus faire de demandes, une fois que le nouvel identifiant de client est confirmé par CREATE_SESSION, les verrous de gamme d'octets et de partage de réservation devraient être libérés immédiatement plutôt que de forcer la nouvelle incarnation à attendre l'expiration du prêt sur la précédente incarnation. De plus, l'état de session devrait être supprimé car si le client avait maintenu ces informations à travers le redémarrage, cette demande n'aurait pas été envoyée. Si le serveur ne prend en charge ni le type de réclamation CLAIM_DELEGATE_PREV ni le type CLAIM_DELEG_PREV_FH, les délégations associées devraient aussi être purgées ; autrement, les délégations sont conservées et la récupération se fait conformément au paragraphe 10.2.1.

   Après le traitement, le clientid_ret est retourné au client et cet enregistrement de client est ajouté:

   { ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

   L'enregistrement confirmé décrit précédemment continue d'exister, et donc le même ownerid_arg existe en même temps dans les deux états confirmé et non confirmé. Le nombre d'états peut tomber à un quand le serveur reçoit un CREATE_SESSION ou EXCHANGE_ID applicable.

   * Si le serveur reçoit ensuite un CREATE_SESSION réussi qui confirme clientid_ret, alors le serveur détruit atomiquement l'enregistrement confirmé et confirme l'enregistrement non confirmé comme décrit au paragraphe 18.36.3.

* Si le serveur reçoit plutôt ensuite un EXCHANGE_ID avec le propriétaire de client égal à ownerid_arg, une stratégie est de simplement supprimer l'enregistrement non confirmé, et traiter le EXCHANGE_ID comme décrit au paragraphe 18.35.4.

6. Mise à jour
   Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A est établi, et si le serveur a l'enregistrement confirmé suivant, alors cette demande est une tentative de mise à jour.

   { ownerid_arg, verifier_arg, principal_arg, clientid_ret, confirmed }

   Comme l'enregistrement a été confirmé, le client doit avoir reçu la réponse du serveur provenant de la demande EXCHANGE_ID initiale. Le serveur permet la mise à jour, et l'enregistrement de client est laissé intact.

7. Mise à jour mais pas d'enregistrement confirmé
   Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A est établi, et si le serveur n'a pas d'enregistrement ownerid_arg non confirmé correspondant, alors le serveur retourne NFS4ERR_NOENT et laisse intact tout enregistrement non confirmé.

8. Mise à jour mais mauvais vérificateur
   Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A est établi, et si le serveur a l'enregistrement suivant confirmé, alors cette demande est une tentative illégale de mise à jour, peut-être parce c'est un ré-essai d'une précédente incarnation de client.

   { ownerid_arg, old_verifier_arg, *, clientid_ret, confirmed }

   Le serveur retourne NFS4ERR_NOT_SAME et laisse intact l'enregistrement de client.

9. Mise à jour mais mauvais principal
   Si EXCHGID4_FLAG_UPD_CONFIRMED_REC_A est établi, et si le serveur a l'enregistrement suivant confirmé, alors cette demande est une tentative illégale de mise à jour par un principal non autorisé.

   { ownerid_arg, verifier_arg, old_principal_arg, clientid_ret, confirmed }

   Le serveur retourne NFS4ERR_PERM et laisse intact l'enregistrement de client.

18.36 Opération 43 : CREATE_SESSION - Crée une nouvelle session et confirme l'identifiant de client

18.36.1 Argument

```
struct canal_attrs4 {
      count4          ca_headerpadsize;
      count4          ca_maxrequestsize;
      count4          ca_maxresponsesize;
      count4          ca_maxresponsesize_cached;
      count4          ca_maxoperations;
      count4          ca_maxrequests;
      uint32_t        ca_rdma_ird<1>;
};

const CREATE_SESSION4_FLAG_PERSIST                  = 0x00000001;
const CREATE_SESSION4_FLAG_CONN_BACK_CHAN   = 0x00000002;
const CREATE_SESSION4_FLAG_CONN_RDMA            = 0x00000004;

struct CREATE_SESSION4args {
      clientid4               csa_clientid;
      sequenceid4             csa_sequence;
      uint32_t                csa_flags;
      channel_attrs4          csa_fore_chan_attrs;
      channel_attrs4          csa_back_chan_attrs;
```

```
        uint32_t                csa_cb_program;
        rappel_sec_parms4       csa_sec_parms<>;
};
```

18.36.2 Résultat

```
struct CREATE_SESSION4resok {
        sessionid4          csr_sessionid;
        sequenceid4         csr_sequence;
        uint32_t            csr_flags;
        channel_attrs4      csr_fore_chan_attrs;
        channel_attrs4      csr_back_chan_attrs;
  };

union CREATE_SESSION4res switch (nfsstat4 csr_status) {
  cas NFS4_OK : CREATE_SESSION4resok    csr_resok4;
  défaut : vide
};
```

18.36.3 Description

Cette opération est utilisée par le client pour créer de nouveaux objets de session sur le serveur.

CREATE_SESSION peut être envoyé avec ou sans une opération SEQUENCE précédente dans la même procédure COMPOUND. Si CREATE_SESSION est envoyé avec une opération SEQUENCE précédente, toute session créée par CREATE_SESSION n'a pas de relation directe à la session spécifiée dans l'opération SEQUENCE, bien que les deux sessions pourraient être associées au même identifiant de client. Si CREATE_SESSION est envoyé sans une SEQUENCE précédente, alors elle DOIT être la seule opération dans la demande de procédure COMPOUND. Si elle ne l'est pas, le serveur DOIT retourner NFS4ERR_NOT_ONLY_OP.

En plus de créer une session, CREATE_SESSION a les effets suivants :

* La première session créée avec un nouvel identifiant de client sert à confirmer la création de l'état de ce client sur le serveur. Le serveur retourne les valeurs de paramètres pour la nouvelle session.
* La connexion CREATE_SESSION qui est envoyée est associée au canal avant de la session.

Les arguments et résultats de CREATE_SESSION sont décrits comme suit :

csa_clientid : c'est l'identifiant de client auquel la nouvelle session va être associée. Le résultat correspondant est csr_sessionid, l'identifiant de session de la nouvelle session.

csa_sequence : chaque identifiant de client met en série CREATE_SESSION via un numéro de séquence par identifiant de client (voir le paragraphe 18.36.4). Le résultat correspondant est csr_sequence, qui DOIT être égal à csa_sequence.

Dans les trois arguments suivants, le client offre une valeur qui doit être une propriété de la session. Sauf mention contraire, il est RECOMMANDÉ que le serveur accepte la valeur. Si elle n'est pas acceptable, le serveur PEUT utiliser une valeur différente. Néanmoins, le serveur DOIT retourner au client la valeur que la session va utiliser (qui va être soit ce que le client a offert, soit ce sur quoi le serveur insiste).

csa_flags : le champ csa_flags contient une liste des bits fanions suivants :

CREATE_SESSION4_FLAG_PERSIST : si CREATE_SESSION4_FLAG_PERSIST est établi, le client veut que le serveur fournisse une mémoire tampon de réponse persistante. Pour les sessions dans lesquelles seulement des opérations idempotentes vont être utilisées (par exemple, une session en lecture seule) les clients NE DEVRAIENT PAS établir CREATE_SESSION4_FLAG_PERSIST. Si le serveur ne veut ou peut pas fournir une mémoire tampon de réponse persistante, le serveur NE DOIT PAS établir CREATE_SESSION4_FLAG_PERSIST dans le champ csr_flags. Si le serveur est un serveur de métadonnées pNFS, pour les raisons décrites au paragraphe 12.5.2, il DEVRAIT prendre en charge CREATE_SESSION4_FLAG_PERSIST si il prend en charge l'attribut layout_hint (paragraphe 5.12.4).

CREATE_SESSION4_FLAG_CONN_BACK_CHAN : si CREATE_SESSION4_FLAG_CONN_BACK_CHAN est établi

dans csa_flags, le client demande que la connexion sur laquelle l'opération CREATE_SESSION est arrivée soit associée au canal de retour de la session en plus de son canal avant. Si le serveur l'accepte, il établit CREATE_SESSION4_FLAG_CONN_BACK_CHAN dans le champ de résultat csr_flags. Si CREATE_SESSION4_FLAG_CONN_BACK_CHAN n'est pas établi dans csa_flags, alors CREATE_SESSION4_FLAG_CONN_BACK_CHAN NE DOIT PAS être établi dans csr_flags.

CREATE_SESSION4_FLAG_CONN_RDMA : si CREATE_SESSION4_FLAG_CONN_RDMA est établi dans csa_flags, et si la connexion sur laquelle l'opération CREATE_SESSION est arrivée n'est actuellement pas en mode RDMA mais a la capacité d'opérer en mode RDMA, alors le client demande que le serveur passe au mode RDMA sur la connexion. Si le serveur l'accepte, il établit CREATE_SESSION4_FLAG_CONN_RDMA dans le champ de résultat csr_flags. Si CREATE_SESSION4_FLAG_CONN_RDMA n'est pas établi dans csa_flags, alors CREATE_SESSION4_FLAG_CONN_RDMA NE DOIT PAS être établi dans csr_flags. Noter qu'une fois que le serveur a accepté de passer au mode RDMA, lui et le client DOIVENT échanger tout le trafic futur sur la connexion avec le tramage RPC RDMA et non plus le marquage d'enregistrement ([RFC8166]).

csa_fore_chan_attrs, csa_back_chan_attrs : les champs csa_fore_chan_attrs et csa_back_chan_attrs s'appliquent aux attributs du canal avant (qui porte les demandes originaires du client au serveur) et du canal de retour (le canal qui porte les demandes de rappel originaires du serveur au client) respectivement. Les résultats sont dans des structures correspondantes appelées csr_fore_chan_attrs et csr_back_chan_attrs. Les résultats établissent des attributs pour chaque canal, et sur toutes les utilisations suivantes de chaque canal de la session. Chaque structure a les champs suivants :

ca_headerpadsize : quantité maximum de bourrage que le demandeur veut appliquer pour s'assurer que les charges utiles d'écriture sont alignées sur une certaine frontière chez le répondeur. Pour chaque canal, le serveur va répondre dans ca_headerpadsize avec sa valeur préférée, ou zéro si le bourrage n'est pas utilisé, et PEUT diminuer cette valeur mais NE DOIT PAS l'augmenter.

ca_maxrequestsize : taille maximum d'une demande COMPOUND ou CB_COMPOUND qui va être envoyée. Cette taille représente la taille de la demande codée en XDR, incluant les en-têtes RPC (incluant les accréditifs et vérificateurs de nuance de sécurité) mais excluant tous les en-têtes de tramage de transport RPC. Imaginons une demande venant sur une connexion TCP/IP non RDMA, et qu'elle ait un seul en-tête de marquage d'enregistrement qui la précède. Le compte maximum permis codé dans l'en-tête va être ca_maxrequestsize. Si un demandeur envoie une demande qui excède ca_maxrequestsize, l'erreur NFS4ERR_REQ_TOO_BIG va être retournée conformément à la description du paragraphe 2.10.6.4. Pour chaque canal, le serveur PEUT diminuer cette valeur mais NE DOIT PAS l'augmenter.

ca_maxresponsesize : taille maximum d'une réponse COMPOUND ou CB_COMPOUND que le demandeur va accepter du répondeur incluant les en-têtes RPC (voir la définition de ca_maxrequestsize). Pour chaque canal, le serveur PEUT diminuer cette valeur, mais NE DOIT PAS l'augmenter. Cependant, si le client choisit une valeur pour ca_maxresponsesize telle qu'un répondeur sur un canal ne pourrait jamais envoyer de réponse, le serveur DEVRAIT retourner NFS4ERR_TOOSMALL dans la réponse CREATE_SESSION. Après la création d'une session, si un demandeur envoie une demande pour laquelle la taille de la réponse excéderait cette valeur, le répondeur va retourner NFS4ERR_REP_TOO_BIG, conformément à la description du paragraphe 2.10.6.4.

ca_maxresponsesize_cached : comme ca_maxresponsesize, mais c'est la taille maximum d'une réponse qui va être mémorisée dans la mémoire tampon de réponse (paragraphe 2.10.6.1). Pour chaque canal, le serveur PEUT diminuer cette valeur, mais NE DOIT PAS l'augmenter. Si, dans la réponse à CREATE_SESSION, la valeur de ca_maxresponsesize_cached d'un canal est moindre que la valeur de ca_maxresponsesize du même canal, alors c'est une indication au demandeur qu'il doit être sélectif quant à la réponse qu'il demande au répondeur de mettre en antémémoire ; par exemple, de grandes réponses provenant d'opérations non idempotentes (par exemple, des demandes COMPOUND avec une opération READ) ne devraient pas être mises en antémémoire. Le demandeur décide de quelles réponses mettre en antémémoire via un argument aux opérations SEQUENCE (le champ sa_cachethis, voir le paragraphe 18.46) ou CB_SEQUENCE (le champ csa_cachethis, voir le paragraphe 20.9). Après la création d'une session, si un demandeur envoie une demande pour laquelle la taille de la réponse excéderait ca_maxresponsesize_cached, le répondeur va retourner NFS4ERR_REP_TOO_BIG_TO_CACHE, conformément à la description du paragraphe 2.10.6.4.

ca_maxoperations : nombre maximum d'opérations que le répondeur va accepter dans un COMPOUND ou CB_COMPOUND. Pour le canal de retour, le serveur NE DOIT PAS changer la valeur qu'offre le client. Pour le canal avant, le serveur PEUT changer la valeur demandée. Après la création de la session, si un demandeur envoie un COMPOUND ou CB_COMPOUND avec plus d'opérations que ca_maxoperations, le répondeur DOIT retourner NFS4ERR_TOO_MANY_OPS.

ca_maxrequests : nombre maximum de demandes COMPOUND ou CB_COMPOUND concurrentes que le demandeur va envoyer sur la session. Les demandes suivantes vont chacune recevoir un identifiant de créneau par le demandeur dans la gamme de zéro à ca_maxrequests - 1 inclus. Pour le canal de retour, le serveur NE DOIT PAS changer la valeur qu'offre le client. Pour le canal avant, le serveur PEUT changer la valeur demandée.

ca_rdma_ird : ce dispositif a un maximum de un élément. Si ce dispositif a un élément, alors l'élément contient la profondeur de file d'attente de lecture RDMA entrante (IRD, *inbound RDMA read queue depth*). Pour chaque canal, le serveur PEUT diminuer cette valeur, mais NE DOIT PAS l'augmenter.

csa_cb_program : c'est le numéro de programme ONC RPC que le serveur DOIT utiliser dans tous les rappels envoyés à travers le canal de retour au client. Le serveur DOIT spécifier un numéro de programme ONC RPC égal à csa_cb_program et un numéro de version ONC RPC égal à 4 dans les rappels envoyés au client. Si un CB_COMPOUND est envoyé au client, le serveur DOIT utiliser un numéro de version mineure de 1. Il n'y a pas de résultat correspondant.

csa_sec_parms : le champ csa_sec_parms est un dispositif des accréditifs de sécurité acceptables que le serveur peut utiliser sur le canal de retour de la session. Trois nuances de sécurité sont prises en charge : AUTH_NOE, AUTH_SYS, et RPCSEC_GSS. Si AUTH_NONE est spécifié pour un accréditif, alors cela dit que le client autorise le serveur à utiliser AUTH_NONE sur tous les rappels pour la session. Si AUTH_SYS est spécifié, alors le client autorise le serveur à utiliser AUTH_SYS sur tous les rappels, en utilisant l'accréditif spécifié cbsp_sys_cred. Si RPCSEC_GSS est spécifié, alors il est permis au serveur d'utiliser le contexte RPCSEC_GSS spécifié dans cbsp_gss_parms comme contexte RPCSEC_GSS dans l'accréditif de l'en-tête RPC des rappels au client. Il n'y a pas de résultat correspondant.

Le contexte RPCSEC_GSS pour le canal de retour est spécifié via une paire de valeurs de type de données gsshandle4_t. Le type de données gsshandle4_t représente une bride RPCSEC_GSS, et est précisément la même que le type de données du champ "handle" du type de données rpc_gss_init_res défini au paragraphe 5.2.3.1 "Réponse de création de contexte - acceptation réussie", de la [RFC2203].

La première bride RPCSEC_GSS, gcbp_handle_from_server, est la bride avant que le serveur a retournée au client (soit dans la bride de champ de type de données rpc_gss_init_res, soit comme un des éléments du champ spi_handles retourné dans la réponse à EXCHANGE_ID) quand le contexte RPCSEC_GSS a été créé sur le serveur. La seconde bride, gcbp_handle_from_client, est la bride arrière à laquelle le client va transposer le contexte RPCSEC_GSS. Le serveur peut utiliser immédiatement la valeur de gcbp_handle_from_client dans l'accréditif RPCSEC_GSS dans les rappels RPC. C'est-à-dire que la valeur dans gcbp_handle_from_client peut être utilisée comme valeur du champ "handle" dans le type de données rpc_gss_cred_t (voir "Éléments du protocole de sécurité RPCSEC_GSS", à la Section 5 de la [RFC2203]) dans les rappels RPC. Le serveur DOIT utiliser le service de sécurité RPCSEC_GSS spécifié dans gcbp_service, c'est-à-dire, il DOIT régler le champ "service" du type de données rpc_gss_cred_t dans l'accréditif RPCSEC_GSS à la valeur de gcbp_service (voir "En-tête de demande RPC", au paragraphe 5.3.1 de la [RFC2203]).

Si la bride RPCSEC_GSS identifiée par gcbp_handle_from_server n'existe pas sur le serveur, le serveur va retourner NFS4ERR_NOENT.

Dans chaque élément de csa_sec_parms, les contextes RPCSEC_GSS avant et arrière DOIVENT partager le même contexte GSS et DOIVENT avoir la même seq_window (voir le paragraphe 5.2.3.1 de la [RFC2203]). L'état du contexte avant et arrière RPCSEC_GSS est indépendant de chaque autre autant que le numéro de séquence RPCSEC_GSS (voir le champ seq_num dans le type de données rpc_gss_cred_t de la Section 5 et du paragraphe 5.3.1 de la [RFC2203]).

Si une bride RPCSEC_GSS utilise le contexte de SSV (voir le paragraphe 2.10.9) alors parce que chaque bride RPCSEC_GSS de SSV partage un contexte GSS commun de SSV, il y a des considérations de sécurité spécifiques de cette situation discutées au paragraphe 2.10.10.

Une fois la session créée, la première SEQUENCE ou CB_SEQUENCE reçue sur un créneau DOIT avoir un identifiant de séquence égal à 1 ; sinon, le répondeur DOIT retourner NFS4ERR_SEQ_MISORDERED.

18.36.4 Mise en œuvre

Pour décrire une mise en œuvre possible, la même notation des enregistrements de client introduite dans la description de EXCHANGE_ID est utilisé avec l'ajout suivant :

clientid_arg : valeur du champ csa_clientid de la structure CREATE_SESSION4args de la demande en cours.

Comme CREATE_SESSION est une opération non idempotente, on doit considérer la possibilité qu'un ré-essai puisse survenir par suite d'un redémarrage du client, d'une partition de réseau, d'un mauvais fonctionnement du routeur, etc. Pour chaque identifiant de client créé par EXCHANGE_ID, le serveur tient une mémoire tampon de réponse séparée (appelée la mémoire tampon de réponse CREATE_SESSION) similaire à la mémoire tampon de réponse de session utilisée pour les opérations SEQUENCE, avec deux distinctions :

* d'abord, c'est une mémoire tampon de réponse juste pour détecter et traiter les demandes CREATE_SESSION pour un identifiant de client donné,
* ensuite, la taille de la mémoire tampon de réponse d'identifiant de client est d'un créneau (et par suite, la demande CREATE_SESSION ne porte pas de numéro de créneau). Cela signifie que au plus une demande CREATE_SESSION pour un identifiant de client donné peut être en instance.

Comme déclaré précédemment, CREATE_SESSION peut être envoyé avec ou sans une opération SEQUENCE précédente. Même si une SEQUENCE précède CREATE_SESSION, le serveur DOIT maintenir la mémoire tampon de réponse de CREATE_SESSION, qui est séparée de la mémoire tampon de réponse pour la session associée à une SEQUENCE. Si CREATE_SESSION a été à l'origine envoyé par lui-même, le client PEUT envoyer un nouvel essai de l'opération CREATE_SESSION au sein d'un COMPOUND précédé d'un SEQUENCE. Si CREATE_SESSION était à l'origine envoyé dans un COMPOUND qui commençait par un SEQUENCE, alors le client DEVRAIT envoyer un nouvel essai dans un COMPOUND qui commence par un SEQUENCE avec le même identifiant de session que le SEQUENCE de la demande d'origine. Cependant, le client PEUT envoyer un nouvel essai dans un COMPOUND qui n'a pas de SEQUENCE précédent, ou a un SEQUENCE précédent qui se réfère à une session différente du CREATE_SESSION original. Cela pourrait être nécessaire si le client envoie un CREATE_SESSION dans un COMPOUND précédé d'un SEQUENCE avec l'identifiant de session X, et que la session X n'existe plus. Néanmoins, tout ré-essai de CREATE_SESSION, avec ou sans un SEQUENCE précédent, DOIT utiliser la même valeur de csa_sequence que l'original.

Après que le client a reçu une réponse à une opération EXCHANGE_ID qui contient un nouvel identifiant de client non confirmé, le serveur s'attend à ce que le client poursuive avec une opération CREATE_SESSION pour confirmer l'identifiant de client. Le serveur s'attend à ce que la valeur de csa_sequenceid dans les arguments à cette CREATE_SESSION soient égaux à la valeur du champ eir_sequenceid qui a été retourné dans les résultats du EXCHANGE_ID qui a retourné l'identifiant de client non confirmé. Avant que le serveur réponde à cette opération EXCHANGE_ID, il initialise le créneau d'identifiant de client à être égal à eir_sequenceid - 1 (tenant compte d'un dépassement de capacité négatif) et enregistre un résultat de CREATE_SESSION arrangé avec un résultat "en antémémoire" de NFS4ERR_SEQ_MISORDERED. Avec le créneau d'identifiant de client ainsi initialisé, le traitement de l'opération CREATE_SESSION est divisé en quatre phases :

1. Recherche de l'enregistrement de client. Le serveur cherche l'identifiant de client dans son tableau d'enregistrements de clients. Si le serveur ne contient aucun enregistrement avec l'identifiant de client égal à clientid_arg, alors il est probable que l'état du client a été purgé durant une période d'inactivité, éventuellement due à une perte de connexité. NFS4ERR_STALE_CLIENTID est retourné, et aucun changement n'est fait aux enregistrements de client sur le serveur. Autrement, le serveur passe à la phase 2.

2. Traitement de l'identifiant de séquence. Si csa_sequenceid est égal à l'identifiant de séquence dans le créneau d'identifiant de client, alors c'est une répétition de la demande CREATE_SESSION précédente, et le serveur retourne le résultat en antémémoire. Si csa_sequenceid n'est pas égal à l'identifiant de séquence dans le créneau, et est supérieur de plus de un (en tenant compte du retour à zéro) alors le serveur retourne l'erreur NFS4ERR_SEQ_MISORDERED, et ne change pas le créneau. Si csa_sequenceid est égal à l'identifiant de séquence du créneau + 1 (en tenant compte du retour à zéro) alors l'identifiant de séquence du créneau est réglé à csa_sequenceid, et le traitement de CREATE_SESSION passe à la prochaine phase. Un nouvel appel CREATE_SESSION suivant sur le même identifiant de client DOIT utiliser un csa_sequenceid qui est supérieur de un à l'identifiant de séquence dans le créneau.

3. Confirmation d'identifiant de client. Si cela va être la première session pour l'identifiant de client, l'opération CREATE_SESSION sert à confirmer l'identifiant de client. Autrement, la phase de confirmation d'identifiant de client est sautée et seule la phase de création de session se produit. Tout cas dans lequel il y a plus d'un enregistrement avec des valeurs identiques pour l'identifiant de client représente une erreur de mise en œuvre de serveur. L'opération dans les potentiels cas valides est résumée comme suit :

   * Confirmation réussie

   Si le serveur a l'enregistrement suivant non confirmé, alors c'est la confirmation attendue d'un enregistrement non confirmé.

   { ownerid, verifier, principal_arg, clientid_arg, unconfirmed }

Comme noté au paragraphe 18.35.4, le serveur pourrait aussi avoir l'enregistrement confirmé suivant :

{ ownerid, old_verifier, principal_arg, old_clientid, confirmed }

Le serveur programme le remplacement des deux enregistrements par :

{ ownerid, verifier, principal_arg, clientid_arg, confirmed }

Le traitement de CREATE_SESSION continue sur la création de session. Une fois que la session est bien créée, le remplacement d'enregistrement de client programmé est engagé. Si la création de la session échoue, alors aucun changement n'est fait aux enregistrements de client sur le serveur.

* Échec de confirmation
Si le serveur a l'enregistrement suivant, alors le client a changé de principal après la précédente demande EXCHANGE_ID, ou il y a eu une collision par hasard entre des identifiants abrégés de client.

{ *, *, old_principal_arg, clientid_arg, * }

Aucun de ces cas n'est admissible. Le traitement cesse et NFS4ERR_CLID_INUSE est retourné au client. Aucun changement n'est fait à un enregistrement de client sur le serveur.

4. Création de session. Le serveur a confirmé l'identifiant de client, soit dans cette opération CREATE_SESSION, soit dans une précédente opération CREATE_SESSION. Le serveur examine les champs restants des arguments.

   Le serveur crée la session en enregistrant les valeurs de paramètre utilisées (y compris si le fanion CREATE_SESSION4_FLAG_PERSIST est établi et a été accepté par le serveur) et en allouant de l'espace pour la mémoire tampon de réponse de la session (si il n'y a pas assez d'espace, le serveur retourne NFS4ERR_NOSPC). Pour chaque créneau dans la mémoire tampon de réponse, le serveur règle l'identifiant de séquence à zéro, et enregistre une entrée contenant une réponse COMPOUND avec zéro opération et l'erreur NFS4ERR_SEQ_MISORDERED. De cette façon, si la première demande SEQUENCE envoyée avait un identifiant de séquence égal à zéro, le serveur peut simplement retourner ce qui est dans la mémoire tampon de réponse : NFS4ERR_SEQ_MISORDERED. Le client initialise sa mémoire tampon de réponse pour recevoir des rappels de la même façon, et de même, la première opération CB_SEQUENCE sur un créneau après la création de session DOIT avoir un identifiant de séquence de un.

   Si l'état de session est créé avec succès, le serveur associe la session à l'identifiant de client fourni par le client.

   Quand une demande qui avait CREATE_SESSION4_FLAG_CONN_RDMA établi doit être réessayée, le ré-essai DOIT être fait sur une nouvelle connexion qui n'est pas en mode RDMA. Si les propriétés de la nouvelle connexion sont assez différentes pour que les arguments à CREATE_SESSION doivent être changés, alors un non ré-essai DOIT être envoyé. Le serveur va finalement disposer de toute session qui avait été créée sur la connexion d'origine.

Sur le canal de retour, le client et le serveur pourraient souhaiter avoir de nombreux créneaux, dans certains cas peut-être plus que le canal avant, afin de traiter les situations où la liaison réseau a une forte latence et est le principal goulet d'étranglement pour la réponse aux rappels. Si il en est ainsi, et si le client fournit trop peu de créneaux au canal de retour, le serveur pourrait limiter le nombre d'objets rappelables qu'il donne au client.

Mettre en œuvre la prise en charge du rappel RPCSEC_GSS exige des changements aux deux mises en œuvre de RPCSEC_GSS du client et du serveur. Un ensemble possible de changements inclut :

* d'ajouter une structure de données qui enveloppe le contexte GSS-API avec un compte de référence,
* de nouvelles fonctions pour incrémenter et décrémenter le compte de référence. Si le compte de référence est décrémenté à zéro, la structure de données enveloppante et le contexte GSS-API à laquelle il réfère vont être libérés ;
* de changer RPCSEC_GSS pour créer la structure de données enveloppante à réception du contexte GSS-API provenant de gss_accept_sec_context() et gss_init_sec_context(). Le compte de référence serait initialisé à 1 ;
* d'ajouter une fonction pour transposer une bride RPCSEC_GSS existante en un pointeur sur la structure de données enveloppante. Le compte de référence serait incrémenté.
* d'ajouter une fonction pour créer une nouvelle bride RPCSEC_GSS à partir d'un pointeur sur la structure de données enveloppante. Le compte de référence serait incrémenté.
* de remplacer les appels provenant de RPCSEC_GSS qui libèrent les contextes GSS-API, par des appels pour décrémenter le compte de référence sur la structure de données enveloppante.

18.37 Opération 44 : DESTROY_SESSION - détruit une session

18.37.1 Argument

```
struct DESTROY_SESSION4args {
        sessionid4      dsa_sessionid;
};
```

18.37.2 Résultat

```
struct DESTROY_SESSION4res {
        nfsstat4        dsr_status;
};
```

18.37.3 Description

L'opération DESTROY_SESSION clôt la session et élimine la mémoire tampon de réponse de la session, si il en est. Toutes les connexions restantes associées à la session sont immédiatement dissociées. Si la connexion n'a pas de session associée restante, la connexion PEUT être close par le serveur. Les verrous, délégations, dispositions, veut, et le prêt, qui sont tous liés à l'identifiant de client, ne sont pas affectés par DESTROY_SESSION.

DESTROY_SESSION DOIT être invoqué sur une connexion associée à la session à détruire. De plus, si la protection d'état SP4_MACH_CRED était spécifiée quand l'identifiant de client a été créé, le principal RPCSEC_GSS qui a créé la session DOIT être celui qui détruit la session, en utilisant la confidentialité ou intégrité RPCSEC_GSS. Si la protection d'état SP4_SSV était spécifiée quand l'identifiant de client a été créé, RPCSEC_GSS utilisant le mécanisme de SSV (paragraphe 2.10.9) DOIT être utilisé, avec l'intégrité ou la confidentialité.

Si la demande COMPOUND commence avec SEQUENCE, et si les identifiants de session spécifiés dans SEQUENCE et DESTROY_SESSION sont les mêmes, alors :

* DESTROY_SESSION DOIT être l'opération finale dans la demande COMPOUND ;
* Il est conseillé d'éviter de placer DESTROY_SESSION dans une demande COMPOUND avec d'autres opérations de modification d'état, parce que le DESTROY_SESSION va détruire la mémoire tampon de réponse ;
* parce que la session et sa mémoire tampon de réponse sont détruites, un client qui réessaye la demande peut recevoir une erreur en réponse au ré-essai, même si la demande d'origine a réussi.

Si la demande COMPOUND commence avec SEQUENCE, et si les identifiants de session spécifiés dans SEQUENCE et DESTROY_SESSION sont différents, alors DESTROY_SESSION peut apparaître dans n'importe quelle position dans la demande COMPOUND (sauf la première position). Les deux identifiants de session peuvent appartenir à des identifiants de client différents.

Si la demande COMPOUND ne commence pas avec SEQUENCE, et si DESTROY_SESSION n'est pas la seule opération, alors serveur DOIT retourner NFS4ERR_NOT_ONLY_OP.

Si il y a un canal de retour sur la session et si le serveur a en instance des opérations CB_COMPOUND pour la session à qui il n'a pas encore été répondu, alors le serveur PEUT refuser de détruire la session et retourner une erreur. Si il en est ainsi, alors au cas où le canal de retour serait mort, le serveur DEVRAIT retourner NFS4ERR_CB_PATH_DOWN pour informer le client que le canal de retour doit être réparé avant que le serveur permette la destruction de la session. Autrement, l'erreur CB_BACK_CHAN_BUSY DEVRAIT être retournée pour indiquer qu'il y a des CB_COMPOUND auxquels il faut répondre. Le client DEVRAIT répondre à tous les CB_COMPOUND en instance avant d'envoyer DESTROY_SESSION à nouveau.

18.38 Opération 45 : FREE_STATEID - libère l'identifiant d'état sans verrous

18.38.1 Argument

```
struct FREE_STATEID4args {
        stateid4        fsa_stateid;
};
```

18.38.2 Résultat

```
struct FREE_STATEID4res {
      nfsstat4       fsr_status;
};
```

18.38.3 Description

L'opération FREE_STATEID est utilisée pour libérer un identifiant d'état qui n'a plus de verrou associé (y compris ouvertures, verrous de gamme d'octets, délégations, et dispositions). Ce peut être à cause d'opérations LOCKU de client ou parce que le serveur les a révoqués. Si il y a des verrous valides (de n'importe quelle sorte) associés à l'identifiant d'état en question, l'erreur NFS4ERR_LOCKS_HELD va être retournée, et l'identifiant d'état associé ne va pas être libéré.

Quand un identifiant d'état est libéré qui avait été associé à des verrous révoqués, par l'envoi de l'opération FREE_STATEID, le client accuse réception de la perte de ces verrous. Cela permet au serveur, une fois que tout cet état révoqué est acquitté, d'admettre que le client réclame à nouveau des verrous, sans rencontrer les conditions limites discutées au paragraphe 8.4.2.

Une fois qu'un FREE_STATEID réussi est fait pour un identifiant d'état donné, toute utilisation ultérieure de cet identifiant d'état va résulter en une erreur NFS4ERR_BAD_STATEID.

18.39 Opération 46 : GET_DIR_DELEGATION - obtient une délégation de répertoire

18.39.1 Argument

```
typedef nfstime4 attr_notice4;

struct GET_DIR_DELEGATION4args {     /* CURRENT_FH : répertoire délégué */
      bool             gdda_signal_deleg_avail;
      bitmap4          gdda_notification_types;
      attr_notice4     gdda_child_attr_delay;
      attr_notice4     gdda_dir_attr_delay;
      bitmap4          gdda_child_attributes;
      bitmap4          gdda_dir_attributes;
};
```

18.39.2 Résultat

```
struct GET_DIR_DELEGATION4resok {
      verifier4        gddr_cookieverf;
      stateid4         gddr_stateid;                  /* identifiant d'état pour get_dir_delegation */
      bitmap4          gddr_notification;             /* quelles notifications le serveur peut prendre en charge */
      bitmap4          gddr_child_attributes;
      bitmap4          gddr_dir_attributes;
};

enum gddrnf4_status {
      GDD4_OK             = 0,
      GDD4_UNAVAIL        = 1
};

union GET_DIR_DELEGATION4res_non_fatal switch (gddrnf4_status gddrnf_status) {
  cas GDD4_OK :   GET_DIR_DELEGATION4resok      gddrnf_resok4;
  cas GDD4_UNAVAIL :  bool                      gddrnf_will_signal_deleg_avail;
};

union GET_DIR_DELEGATION4res switch (nfsstat4 gddr_status) {
  cas NFS4_OK :  GET_DIR_DELEGATION4res_non_fatal      gddr_res_non_fatal4;
  défaut : vide
```

};

18.39.3 Description

L'opération GET_DIR_DELEGATION est utilisée par un client pour demander une délégation de répertoire. Le répertoire est représenté par la bride de fichier courante. Le client spécifie aussi si il veut que le serveur lui notifie quand le répertoire change d'une certaine façon en établissant un ou plusieurs bits dans un gabarit binaire. Le serveur peut refuser d'accorder la délégation. Dans ce cas, le serveur va retourner NFS4ERR_DIRDELEG_UNAVAIL. Si le serveur décide remettre la délégation, il va retourner un vérificateur de mouchard pour ce répertoire. Si le vérificateur de mouchard change quand le client détient la délégation, la délégation va être rappelée sauf si le client a demandé la notification de cet événement.

Le serveur va aussi retourner un identifiant d'état de délégation de répertoire, gddr_stateid, par suite de l'opération GET_DIR_DELEGATION. Cet identifiant d'état va apparaître dans les messages de rappel relatifs à la délégation, comme des notifications et rappels de délégation. Le client va utiliser cet identifiant d'état pour retourner la délégation volontairement ou sur rappel. Une délégation est retournée en invoquant l'opération DELEGRETURN.

Le serveur pourrait n'être pas capable de prendre en charge les notifications de certains événements. Si le client demande de telles notifications, le serveur DOIT informer le client de son incapacité à le faire au titre de la réponse GET_DIR_DELEGATION en n'établissant pas les bits appropriés dans le gabarit de notifications de prise en charge, gddr_notification, contenus dans la réponse. Le serveur NE DOIT PAS ajouter de bits à gddr_notification que le client n'a pas demandés.

L'opération GET_DIR_DELEGATION peut être utilisée pour les répertoires d'attributs normaux et nommés.

Si le client règle gdda_signal_deleg_avail à VRAI, alors il enregistre avec le client un "want" pour une délégation de répertoire. Si la délégation n'est pas disponible, et si le serveur prend en charge et va honorer le "want", le résultat aura gddrnf_will_signal_deleg_avail réglé à VRAI et aucune erreur ne sera indiquée en retour. Si il en est ainsi, le client devrait s'attendre à une future opération CB_RECALLABLE_OBJ_AVAIL pour indiquer qu'une délégation de répertoire est disponible. Si le serveur ne souhaite pas honorer le "want" ou n'est pas capable de le faire, il retourne l'erreur NFS4ERR_DIRDELEG_UNAVAIL. Si la délégation est immédiatement disponible, le serveur DEVRAIT la retourner avec la réponse à l'opération, plutôt que via un rappel.

Quand un client fait une demande pour une délégation de répertoire alors qu'il détient déjà une délégation de répertoire pour ce répertoire (y compris le cas où elle a été rappelée mais pas encore retournée par le client ou révoquée par le serveur) le serveur DOIT répondre avec la valeur de gddr_status réglée à NFS4_OK, la valeur de gddrnf_status réglée à GDD4_UNAVAIL, et la valeur de gddrnf_will_signal_deleg_avail réglée à FAUX. La délégation que le client détenait avant la demande reste intacte, et son état est inchangé. L'identifiant d'état courant n'est pas changé (voir au paragraphe 16.2.3.1.2 la description de l'identifiant d'état courant).

18.39.4 Mise en œuvre

Les délégations de répertoire fournissent l'avantage d'améliorer la cohérence des informations d'espace de noms dans l'antémémoire. C'est fait par des rappels synchrones. Un serveur doit prendre en charge les rappels synchrones afin de prendre en charge les délégations de répertoire. De plus, les notifications asynchrones fournissent un moyen pour réduire le trafic du réseau ainsi que d'améliorer les performances de client dans certaines conditions.

Les notifications sont spécifiées en termes de changements potentiels au répertoire. Un client peut demander que des événements lui soient notifiés en établissant un ou plusieurs bits dans gdda_notification_types. Le client peut demander des notifications d'ajouts d'entrées à un répertoire (en établissant NOTIFY4_ADD_ENTRY dans gdda_notification_types) des notifications de suppression d'entrées (NOTIFY4_REMOVE_ENTRY) des changements de dénomination (NOTIFY4_RENAME_ENTRY) des changements d'attribut de répertoire (NOTIFY4_CHANGE_DIR_ATTRIBUTES) et des changements de vérificateur de mouchard (NOTIFY4_CHANGE_COOKIE_VERIFIER) en établissant un ou plusieurs bits correspondants dans le champ gdda_notification_types.

Le client peut aussi demander des notifications de changement des entrées d'attributs de répertoire (NOTIFY4_CHANGE_CHILD_ATTRIBUTES) afin de garder à jour cette mémoire tampon d'attributs. Cependant, tout changement fait aux attributs fils ne cause pas le rappel de la délégation. Si un client est intéressé par la mise en antémémoire d'entrée de répertoire ou la mise en antémémoire négative de noms, il peut régler le gdda_notification_types de façon appropriée pour son besoin particulier et le serveur lui notifier tous les changements qui autrement invalideraient

son antémémoire de noms. La sorte de notification qu'un client demande peut dépendre de la taille du répertoire, de son taux de changement, et des applications utilisées pour accéder à ce répertoire. L'énumération des conditions dans lesquelles un client pourrait demander des notifications sort du domaine d'application de cette spécification.

Pour les notifications d'attribut, le client va établir des bits dans le gabarit binaire gdda_dir_attributes pour indiquer de quels attributs il veut être notifié. Si le serveur ne prend pas en charge les notifications pour les changements d'un certain attribut, il NE DEVRAIT PAS établir cet attribut dans le gabarit binaire des attributs pris en charge spécifié dans la réponse (gddr_dir_attributes). Le client va aussi établir dans le gabarit binaire gdda_child_attributes les attributs d'entrées de répertoire dont il veut être notifié, et le serveur va indiquer dans gddr_child_attributes quels attributs d'entrées de répertoire il va notifier au client.

Le client va aussi faire savoir au serveur si il veut obtenir la notification aussitôt que le changement d'attribut se produit ou après un certain délai en réglant un facteur de délai ; gdda_child_attr_delay est pour les changements d'attribut des entrées de répertoire et gdda_dir_attr_delay est pour les changements d'attribut au répertoire. Si ce facteur de délai est réglé à zéro, cela indique au serveur que le client veut avoir notification de tout changement d'attribut aussitôt qu'il se produit. Si le facteur de délai est réglé à N secondes, le serveur va faire de son mieux pour garantir que les mises à jour d'attribut sont synchronisées sur N secondes. Si le client demande un facteur de délai que le serveur ne prend pas en charge ou qui peut causer une consommation significative de ressources chez le serveur en causant l'envoi par le serveur de nombreuses notifications, le serveur ne devrait pas s'engager à envoyer des notifications pour les attributs et donc ne doit pas établir le bit approprié dans les gabarits binaires gddr_child_attributes et gddr_dir_attributes dans la réponse.

Le client DOIT utiliser un couple de sécurité (paragraphe 2.6.1) avec lequel le répertoire ou son ancêtre applicable (paragraphe 2.6) est exporté. Sinon, le serveur DOIT retourner NFS4ERR_WRONGSEC à l'opération qui à la fois précède GET_DIR_DELEGATION et établit la bride de fichier courante (voir le paragraphe 2.6.3.1).

La délégation de répertoire couvre toutes les entrées dans le répertoire sauf l'entrée parente. Cela signifie que si un répertoire et son parent détiennent tous deux des délégations de répertoire, un changement au parent ne causera pas l'envoi d'une notification pour l'enfant même si l'entrée parente de l'enfant pointe sur le répertoire parent.

18.40 Opération 47 : GETDEVICEINFO - obtient les informations d'appareil

18.40.1 Argument

```
struct GETDEVICEINFO4args {
      deviceid4       gdia_device_id;
      layouttype4     gdia_layout_type;
      count4          gdia_maxcount;
      bitmap4         gdia_notify_types;
};
```

18.40.2 Résultat

```
struct GETDEVICEINFO4resok {
      device_addr4    gdir_device_addr;
      bitmap4         gdir_notification;
};

union GETDEVICEINFO4res switch (nfsstat4 gdir_status) {
  cas NFS4_OK : GETDEVICEINFO4resok     gdir_resok4;
  cas NFS4ERR_TOOSMALL : count4         gdir_mincount;
  défaut : vide
};
```

18.40.3 Description

L'opération GETDEVICEINFO retourne les informations d'adresse d'appareil de mémorisation pNFS pour l'identifiant d'appareil spécifié. Le client identifie les informations d'appareil à retourner en fournissant le gdia_device_id et gdia_layout_type qui identifie l'appareil de façon univoque. Le client fournit gdia_maxcount pour limiter le nombre d'octets pour le résultat. Cette taille maximum représente toutes les données retournées dans la structure GETDEVICEINFO4resok

et inclut les frais généraux de XDR. Le serveur peut retourner moins de données. Si le serveur est incapable de retourner des informations dans la limite de gdia_maxcount, l'erreur NFS4ERR_TOOSMALL va être retournée. Cependant, si gdia_maxcount est zéro, NFS4ERR_TOOSMALL NE DOIT PAS être retourné.

Le champ da_layout_type de la gdir_device_addr retournée par le serveur DOIT être égal au gdia_layout_type spécifié par le client. Si il n'est pas égal, le client DEVRAIT ignorer la réponse comme invalide et se comporter comme si le serveur avait retourné une erreur, même si le client prend bien en charge le type de disposition retourné.

Le client fournit aussi un gabarit binaire de notification, gdia_notify_types, pour la notification de transposition d'identifiant d'appareil qu'il est intéressé à recevoir ; le serveur doit prendre en charge les notifications d'identifiant d'appareil pour que la demande de notification prenne effet. Le gabarit de notification est composé de la même manière que le gabarit binaire pour les attributs de fichier (paragraphe 3.3.7). Les numéros de positions de bits sont énumérés dans le type d'énumération notify_device_type4 (paragraphe 20.12). Seules deux valeurs énumérées de notify_device_type4 s'appliquent actuellement à GETDEVICEINFO : NOTIFY_DEVICEID4_CHANGE et NOTIFY_DEVICEID4_DELETE (voir le paragraphe 20.12).

Le gabarit binaire de notification s'applique seulement à l'identifiant d'appareil spécifié. Si un client envoie une opération GETDEVICEINFO plusieurs fois sur un deviceID, le dernier gabarit binaire de notification est utilisé par le serveur pour les notifications suivantes. Si le gabarit binaire est zéro ou vide, alors les notifications d'identifiant d'appareil sont désactivées.

Si le client veut juste mettre à jour ou désactiver les notifications, il PEUT envoyer une opération GETDEVICEINFO avec gdia_maxcount réglé à zéro. Dans ce cas, si l'identifiant d'appareil est valide, le champ da_addr_body du champ gdir_device_addr de la réponse va être de longueur zéro.

Si un identifiant d'appareil inconnu est donné dans gdia_device_id, le serveur retourne NFS4ERR_NOENT. Autrement, les informations d'adresse d'appareil sont retournées dans gdir_device_addr. Finalement, si le serveur prend en charge les notifications pour les transpositions d'identifiant d'appareil, le résultat de gdir_notification va contenir un gabarit binaire des notifications qu'il va réellement envoyer au client (via CB_NOTIFY_DEVICEID, voir le paragraphe 20.12).

Si NFS4ERR_TOOSMALL est retourné, le résultat contient aussi gdir_mincount. La valeur de gdir_mincount représente la taille minimum nécessaire pour obtenir les informations d'appareil.

18.40.4 Mise en œuvre

À part de mettre à jour ou de désactiver les notifications, un autre usage de gdia_maxcount réglé à zéro est de valider un identifiant d'appareil.

Le client DEVRAIT demander une notification des changements ou de la suppression d'un identifiant d'appareil en transposition d'adresse d'appareil afin que le serveur puisse permettre au client d'utiliser en douceur une nouvelle transposition, sans avoir d'entrée/sortie en cours qui échoue de façon abrupte, ou forcer le rappel ou la révocation des dispositions qui utilisent l'identifiant d'appareil.

Il est possible que GETDEVICEINFO (et GETDEVICELIST) entre en concurrence avec CB_NOTIFY_DEVICEID, c'est-à-dire, CB_NOTIFY_DEVICEID arrive avant que le client obtienne et traite la réponse à GETDEVICEINFO ou GETDEVICELIST. L'analyse de la concurrence souligne le fait que le serveur NE DOIT PAS supprimer un identifiant d'appareil à qui se réfère une disposition qu'a le client.

* CB_NOTIFY_DEVICEID supprime un identifiant d'appareil. Si le client croit qu'il a des dispositions qui se réfèrent à l'identifiant d'appareil, alors il est possible que des dispositions se référant à l'identifiant d'appareil supprimé aient été révoquées. Le client devrait envoyer une demande TEST_STATEID utilisant l'identifiant d'état pour chaque disposition qui pourrait avoir été révoquée. Si TEST_STATEID indique que des dispositions ont été révoquées, le client doit récupérer de la révocation de disposition comme décrit au paragraphe 12.5.6. Si TEST_STATEID indique qu'au moins une disposition n'a pas été révoquée, le client devrait envoyer une opération GETDEVICEINFO sur l'identifiant d'appareil supposé supprimé pour vérifier que l'identifiant d'appareil a été supprimé.

  Si GETDEVICEINFO indique que l'identifiant d'appareil n'existe pas, alors le client suppose que le serveur est fautif et récupère en envoyant une opération EXCHANGE_ID. Si GETDEVICEINFO indique que l'identifiant d'appareil existe bien, alors le serveur est fautif en envoyant une notification erronée de suppression d'identifiant d'appareil, le degré de faute n'exige pas que le client crée un nouvel identifiant de client.

Si le client n'a pas de disposition qui se réfère à l'identifiant d'appareil, il n'y a aucun dommage. Le client devrait marquer l'identifiant d'appareil comme supprimé, et quand les résultats de GETDEVICEINFO ou GETDEVICELIST sont reçus qui indiquent que l'identifiant d'appareil a bien été supprimé, l'identifiant d'appareil devrait être supprimé de l'antémémoire du client.

* CB_NOTIFY_DEVICEID indique que les transpositions d'adressage d'appareil d'un identifiant d'appareil ont changé. Le client devrait supposer que le résultat du GETDEVICEINFO en cours va être périmé pour l'identifiant d'appareil une fois reçu, et donc il devrait envoyer un autre GETDEVICEINFO sur l'identifiant d'appareil.

18.41 Opération 48 : GETDEVICELIST - obtient toutes les transpositions d'appareils pour un système de fichiers

18.41.1 Argument

```
struct GETDEVICELIST4args {                    /* CURRENT_FH : objet appartenant au système de fichiers */
      layouttype4     gdla_layout_type;
      count4          gdla_maxdevices;         /* nombre d'identifiants d'appareil à retourner */
      nfs_cookie4     gdla_cookie;
      verifier4       gdla_cookieverf;
};
```

18.41.2 Résultat

```
struct GETDEVICELIST4resok {
      nfs_cookie4       gdlr_cookie;
      verifier4         gdlr_cookieverf;
      deviceid4         gdlr_deviceid_list<>;
      bool              gdlr_eof;
};

union GETDEVICELIST4res switch (nfsstat4 gdlr_status) {
  cas NFS4_OK : GETDEVICELIST4resok     gdlr_resok4;
  défaut : vide
};
```

18.41.3 Description

Cette opération est utilisé par le client pour énumérer tous les identifiants d'appareil que le système de fichiers d'un serveur utilise.

Le client fournit une bride de fichier courante d'un objet de fichier qui appartient au système de fichiers (c'est-à-dire, tous les objets fichier qui partagent le même fsid que celui de la bride de fichier courante) et le type de disposition dans gdia_layout_type. Comme cette opération pourrait exiger plusieurs appels pour énumérer tous les identifiants d'appareil (et est donc similaire à l'opération READDIR (paragraphe 18.23)) le client fournit aussi gdia_cookie et gdia_cookieverf pour spécifier la position actuelle du curseur dans la liste. Quand le client veut lire à partir du début des transpositions d'appareil du système de fichiers, il règle gdla_cookie à zéro. Le champ gdla_cookieverf DOIT être ignoré par le serveur quand gdla_cookie est zéro. Le client fournit gdla_maxdevices pour limiter le nombre d'identifiants d'appareil dans le résultat. Si gdla_maxdevices est zéro, le serveur DOIT retourner NFS4ERR_INVAL. Le serveur PEUT retourner moins d'identifiants d'appareil.

La réponse de succès à l'opération va contenir le mouchard, gdlr_cookie, et le vérificateur de mouchard, gdlr_cookieverf, à utiliser sur le GETDEVICELIST suivant. Une valeur de gdlr_eof de VRAI signifie qu'il ne reste pas d'entrée dans la liste d'appareils du serveur. Chaque élément de gdlr_deviceid_list contient un identifiant d'appareil.

18.41.4 Mise en œuvre

Un exemple de l'utilisation de cette opération est pour les clients et serveurs pNFS qui utilisent des dispositions LAYOUT4_BLOCK_VOLUME. Dans ces environnements il peut être utile qu'un client détermine l'accessibilité d'un appareil au premier accès au système de fichiers.

18.42 Opération 49 : LAYOUTCOMMIT - engage les écritures faites en utilisant une disposition

18.42.1 Argument

```
union newtime4 switch (bool nt_timechanged) {
  cas VRAI : nfstime4          nt_time;
  cas FAUX : vide
};

union newoffset4 switch (bool no_newoffset) {
  cas VRAI : offset4          no_offset;
  cas FAUX : vide
};

struct LAYOUTCOMMIT4args {                              /* CURRENT_FH : fichier */
      offset4              loca_offset;
      length4              loca_length;
      bool                 loca_reclaim;
      stateid4             loca_stateid;
      newoffset4           loca_last_write_offset;
      newtime4             loca_time_modify;
      layoutupdate4        loca_layoutupdate;
};
```

18.42.2 Résultat

```
union newsize4 switch (bool ns_sizechanged) {
  cas VRAI : length4        ns_size;
  cas FAUX : vide
};

struct LAYOUTCOMMIT4resok {
      newsize4              locr_newsize;
};

union LAYOUTCOMMIT4res switch (nfsstat4 locr_status) {
  cas NFS4_OK : LAYOUTCOMMIT4resok      locr_resok4;
  défaut : vide
};
```

18.42.3 Description

L'opération LAYOUTCOMMIT engage les changements dans la disposition représentée par la bride de fichier courante, l'identifiant de client (déduit de l'identifiant de session dans la précédente opération SEQUENCE) la gamme d'octets, et l'identifiant d'état. Comme les dispositions sont subdivisibles, une plus petite portion d'une disposition, restituée via LAYOUTGET, peut être engagée. La gamme d'octets engagée est spécifiée par la gamme d'octets (loca_offset et loca_length). Cette gamme d'octets DOIT chevaucher une ou plusieurs dispositions existantes accordées précédemment via LAYOUTGET (paragraphe 18.43) chacune avec un iomode de LAYOUTIOMODE4_RW. Dans le cas où le iomode de tout segment de disposition détenu n'est pas LAYOUTIOMODE4_RW, le serveur devrait retourner l'erreur NFS4ERR_BAD_IOMODE. Pour le cas où le client ne détient pas de segment de disposition correspondant pour la gamme d'octets définie, le serveur devrait retourner l'erreur NFS4ERR_BAD_LAYOUT.

L'opération LAYOUTCOMMIT indique que le client a terminé les écritures en utilisant une disposition obtenue par un

précédent LAYOUTGET. Le client peut avoir seulement écrit un sous ensemble de la gamme de données qu'il demandait précédemment. LAYOUTCOMMIT lui permet d'engager ou d'éliminer de l'espace alloué provisoirement et de mettre à jour le serveur avec une nouvelle fin de fichier. La disposition référencée par LAYOUTCOMMIT est encore valide après l'achèvement de l'opération et sa référence peut être continuée par l'identifiant de client, la bride de fichier, la gamme d'octets, le type de disposition, et l'identifiant d'état.

Si le champ loca_reclaim est réglé à VRAI, cela indique que le client tente d'engager des changements à une disposition après le redémarrage du serveur de métadonnées durant la période de grâce de récupération du serveur de métadonnées (voir le paragraphe 12.7.4). Ce type de demande peut être nécessaire quand le client a des écritures non engagées sur des gammes d'octets allouées provisoirement d'un fichier qui ont été envoyées aux appareils de mémorisation avant le redémarrage du serveur de métadonnées. Dans ce cas, la disposition fournie par le client DOIT être un sous ensemble d'une disposition écrivable que le client détenait immédiatement avant le redémarrage du serveur de métadonnées. La valeur du champ loca_stateid DOIT être une valeur que le serveur de métadonnées avait retournée avant son redémarrage. Le serveur de métadonnées est libre d'accepter ou rejeter cette demande sur la base de ses propres vérifications de cohérence de métadonnées internes. Si le serveur de métadonnées trouve que la disposition fournie par le client ne satisfait pas à ses vérifications de cohérence, il DOIT rejeter la demande avec l'état NFS4ERR_RECLAIM_BAD. L'achèvement réussi de la demande LAYOUTCOMMIT avec loca_reclaim réglé à VRAI NE fournit PAS au client une disposition pour le fichier. Cela engage simplement les changements à la disposition spécifiée dans le champ loca_layoutupdate. Pour obtenir une disposition pour le fichier, le client doit envoyer une demande LAYOUTGET au serveur après l'expiration de la période de grâce du serveur. Si le serveur de métadonnées reçoit une demande LAYOUTCOMMIT avec loca_reclaim réglé à VRAI quand le serveur de métadonnées n'est pas dans sa période de grâce de récupération, il DOIT rejeter la demande avec l'état NFS4ERR_NO_GRACE.

Régler le champ loca_reclaim à VRAI n'est exigé que si et seulement si la disposition engagée a été acquise avant le redémarrage du serveur de métadonnées. Si le client engage une disposition qui a été acquise durant la période de grâce du serveur de métadonnées, il DOIT régler le champ "reclaim" à FAUX.

Le loca_stateid est une valeur d'identifiant d'état de disposition comme retournée par des opérations de disposition précédemment réussies (voir le paragraphe 12.5.3).

Le champ loca_last_write_offset spécifie le décalage du dernier octet écrit par le client avant l'opération LAYOUTCOMMIT. Noter que cette valeur n'est jamais égale à la taille du fichier (au plus elle est inférieure d'un octet à la taille du fichier) et DOIT être inférieure ou égale à NFS4_MAXFILEOFF. Aussi, loca_last_write_offset DOIT chevaucher la gamme décrite par loca_offset et loca_length. Le serveur de métadonnées peut utiliser ces informations pour déterminer si la taille du fichier doit être mise à jour. Si le serveur de métadonnées met à jour la taille du fichier par suite d'une opération LAYOUTCOMMIT, il doit retourner la nouvelle taille (locr_newsize.ns_size) au titre des résultats.

Le champ loca_time_modify permet au client de suggérer au serveur de métadonnées une heure de modification qu'il aimerait qu'il établisse. Le serveur de métadonnées peut utiliser la suggestion ou il peut utiliser l'heure de l'opération LAYOUTCOMMIT pour régler l'heure de modification. Si le serveur de métadonnées utilise l'heure de modification fournie par le client, il devrait s'assurer que ce temps ne va pas à reculons. Si le client veut forcer le serveur de métadonnées à régler une heure exacte, le client devrait utiliser une opération SETATTR dans un COMPOUND juste après LAYOUTCOMMIT. Voir les détails au paragraphe 12.5.4. Si le client désire l'heure de modification résultante, il devrait construire le COMPOUND de telle façon que un GETATTR suive le LAYOUTCOMMIT.

L'argument loca_layoutupdate à LAYOUTCOMMIT fournit un mécanisme pour qu'un client fournisse des mises à jour spécifique de disposition au serveur de métadonnées. Par exemple, la mise à jour de disposition peut décrire quelles gammes d'octets de la disposition originale ont été utilisées et quelles gammes d'octets peuvent être désallouées. Il n'y a pas de structure NFSv4.1 layoutupdate4 spécifique de disposition de fichier.

Les informations de disposition sont plus prolixes pour les appareils de blocs que pour les objets et fichiers parce que les deux derniers cachent les détails de l'allocation de blocs derrière leurs protocoles de mémorisation. Au minimum, le client a besoin de communiquer les changements de la localisation de fin de fichier au serveur, et, si désiré, sa vue de l'heure de modification du fichier. Pour les dispositions de bloc/volume, il a besoin de spécifier précisément quels blocs ont été utilisés.

Si la disposition identifiée dans les arguments n'existe pas, l'erreur NFS4ERR_BADLAYOUT est retournée. La disposition engagée peut aussi être rejetée si elle ne correspond pas à une disposition existante avec un iomode de LAYOUTIOMODE4_RW.

En cas de succès, la bride de fichier courante conserve sa valeur et l'identifiant d'état courant conserve sa valeur.

18.42.4 Mise en œuvre

Le client PEUT aussi utiliser LAYOUTCOMMIT avec le champ loca_reclaim réglé à VRAI pour porter des indications sur les attributs de fichier modifiés ou pour rapporter des informations spécifiques du type de disposition telles que des erreurs d'entrée/sortie pour des dispositions de mémorisation fondées sur l'objet, comme fait normalement durant le fonctionnement normal. Faire ainsi peut aider le serveur de métadonnées à récupérer des fichiers plus efficacement après un redémarrage. Par exemple, certaines mises en œuvre de système de fichiers peuvent exiger une récupération coûteuse d'objets de système de fichiers si le serveur de métadonnées n'a pas une indication positive de la part de tous les clients qui détiennent une disposition LAYOUTIOMODE4_RW qu'ils ont terminé avec succès toutes leurs écritures. Envoyer un LAYOUTCOMMIT (si nécessaire) et ensuite suivre avec LAYOUTRETURN peut fournir une telle indication et permettre une récupération en douceur et efficace.

Si loca_reclaim est VRAI, le serveur de métadonnées est libre d'examiner ou ignorer la valeur dans le champ loca_stateid. La mise en œuvre de serveur de métadonnées pourrait ou non coder dans ses informations d'identifiant d'état de disposition qu'elle permet au serveur de métadonnées d'effectuer une vérification de cohérence sur la demande LAYOUTCOMMIT.

18.43 Opération 50 : LAYOUTGET - obtient les informations de disposition

18.43.1 Argument

```
struct LAYOUTGET4args {                                  /* CURRENT_FH : fichier */
      bool                loga_signal_layout_avail;
      layouttype4         loga_layout_type;
      layoutiomode4       loga_iomode;
      offset4             loga_offset;
      length4             loga_length;
      length4             loga_minlength;
      stateid4            loga_stateid;
      count4              loga_maxcount;
};
```

18.43.2 Résultat

```
struct LAYOUTGET4resok {
      bool            logr_return_on_close;
      stateid4        logr_stateid;
      layout4         logr_layout<>;
};

union LAYOUTGET4res switch (nfsstat4 logr_status) {
  cas NFS4_OK : LAYOUTGET4resok              logr_resok4;
  cas NFS4ERR_LAYOUTTRYLATER : bool     logr_will_signal_layout_avail;
  défaut : vide
};
```

18.43.3 Description

L'opération LAYOUTGET demande une disposition de la part du serveur de métadonnées pour lire ou écrire le fichier donné par la bride de fichier à la gamme d'octets spécifiée par le décalage et la longueur. Les dispositions sont identifiées par l'identifiant de client (déduit de l'identifiant de session dans la précédente opération SEQUENCE) la bride de fichier courante, le type de disposition (loga_layout_type) et l'identifiant d'état de disposition (loga_stateid). L'utilisation du champ loga_iomode dépend du type de disposition, mais devrait refléter l'intention d'accès aux données du client.

Si le serveur de métadonnées est dans une période de grâce, et ne conserve pas les transpositions de dispositions et identifiant d'appareil en adresse d'appareil, alors il DOIT retourner NFS4ERR_GRACE (voir le paragraphe 8.4.2.1).

L'opération LAYOUTGET retourne les informations de disposition pour la gamme d'octets spécifiée : une disposition. Le

client spécifie en fait deux gammes, toutes deux commençant au décalage dans le champ loga_offset. La première gamme est entre loga_offset et loga_offset + loga_length - 1 inclus. Cette gamme indique la gamme désirée que le client veut que la disposition couvre. La seconde gamme est entre loga_offset et loga_offset + loga_minlength - 1 inclus. Cette gamme indique la gamme requise dont le client a besoin que la disposition couvre. Donc, loga_minlength DOIT être inférieur ou égal à loga_length.

Quand un champ de longueur est réglé à NFS4_UINT64_MAX, cela indique le désir (quand loga_length est NFS4_UINT64_MAX) ou l'exigence (quand loga_minlength est NFS4_UINT64_MAX) d'obtenir une disposition à partir de loga_offset jusqu'à la fin du fichier, sans considération de la longueur du fichier.

Les règles suivantes gouvernent les relations entre loga_length, loga_minlength, et loga_offsetong, et leurs minimums.

* Si loga_length est inférieur à loga_minlength, le serveur de métadonnées DOIT retourner NFS4ERR_INVAL.

* Si loga_minlength est zéro, c'est l'indication au serveur de métadonnées que le client désire une disposition au décalage loga_offset ou moins que le serveur de métadonnées a "directement disponible". Directement est subjectif, et dépend du type de disposition et de la mise en œuvre de serveur pNFS. Par exemple, certains serveurs de métadonnées pourraient avoir une mémorisation stable pré-allouée quand ils reçoivent une demande pour une gamme d'un fichier qui va au delà de la longueur actuelle du fichier. Si loga_minlength est zéro et si loga_length est supérieur à zéro, cela dit au serveur de métadonnées quelle gamme de la disposition le client préférerait avoir. Si loga_length et loga_minlength sont tous les deux à zéro, alors le client indique qu'il désire une disposition de n'importe quelle longueur avec la fin du décalage de la gamme non inférieur à la valeur spécifiée de loga_offset, et le début du décalage à ou en dessous de loga_offset. Si le serveur de métadonnées n'a pas une disposition directement disponible, alors il DOIT retourner NFS4ERR_LAYOUTTRYLATER.

* Si la somme de loga_offset et de loga_minlength excède NFS4_UINT64_MAX, et si loga_minlength n'est pas NFS4_UINT64_MAX, l'erreur NFS4ERR_INVAL DOIT résulter.

* Si la somme de loga_offset et de loga_length excède NFS4_UINT64_MAX, et si loga_length n'est pas NFS4_UINT64_MAX, l'erreur NFS4ERR_INVAL DOIT résulter.

Après que le serveur de métadonnées a effectué les vérifications ci-dessus sur loga_offset, loga_minlength, et loga_offset, le serveur de métadonnées DOIT retourner une disposition conformément aux règles du Tableau 22.

Dispositions acceptables fondées sur loga_minlength.
Note : u64m = NFS4_UINT64_MAX ; a_off = loga_offset ; a_minlen = loga_minlength.

iomode de disposition de demande	**a_minlen de disposition de demande**	**iomode de disposition de réponse**	**Décalage de disposition de réponse**	**Longueur de disposition de réponse**
_READ	u64m	PEUT être _READ	DOIT être ≤ a_off	DOIT être ≥ longueur de fichier - décalage de disposition
_READ	u64m	PEUT être _RW	DOIT être ≤ a_off	DOIT être u64m
_READ	> 0 et < u64m	PEUT être _READ	DOIT être ≤ a_off	DOIT être ≥ MIN(longueur de fichier, a_minlen + a_off) - décalage de disposition
_READ	> 0 et < u64m	PEUT être _RW	DOIT être ≤ a_off	DOIT être ≥ a_off - décalage de disposition + a_minlen
_READ	0	PEUT être _READ	DOIT être ≤ a_off	DOIT être > 0
_READ	0	PEUT être _RW	DOIT être ≤ a_off	DOIT être > 0
_RW	u64m	DOIT être _RW	DOIT être ≤ a_off	DOIT être u64m
_RW	> 0 et < u64m	DOIT être _RW	DOIT être ≤ a_off	DOIT être ≥ a_off - décalage de disposition + a_minlen
_RW	0	DOIT être _RW	DOIT être ≤ a_off	DOIT être > 0

Tableau 22

Si loga_minlength n'est pas zéro et si le serveur de métadonnées ne peut pas retourner une disposition en accord avec les règles du Tableau 22, alors le serveur de métadonnées DOIT retourner l'erreur NFS4ERR_BADLAYOUT. Si loga_minlength est zéro et si le serveur de métadonnées ne peut pas ou ne veut pas retourner une disposition en accord avec les règles du Tableau 22, alors le serveur de métadonnées DOIT retourner l'erreur NFS4ERR_LAYOUTTRYLATER. En

supposant que loga_length est supérieur à loga_minlength ou égal à zéro, le serveur de métadonnées DEVRAIT retourner une disposition selon les règles du Tableau 23.

Dispositions désirées fondées sur loga_length. Les règles du Tableau 22 DOIVENT être appliquées en premier.
Note : u64m = NFS4_UINT64_MAX ; a_off = loga_offset ; a_len = loga_length.

iomode de disposition de demande	**a_len de disposition de demande**	**iomode de disposition de réponse**	**Décalage de disposition de réponse**	**Longueur de disposition de réponse**
_READ	u64m	PEUT être _READ	DOIT être ≤ a_off	DEVRAIT être u64m
_READ	u64m	PEUT être _RW	DOIT être ≤ a_off	DEVRAIT être u64m
_READ	> 0 et < u64m	PEUT être _READ	DOIT être ≤ a_off	DEVRAIT être ≥ a_off - décalage de disposition + a_len
_READ	> 0 et < u64m	PEUT être _RW	DOIT être ≤ a_off	DEVRAIT être ≥ a_off - décalage de disposition
_READ	0	PEUT être _READ	DOIT être ≤ a_off	DEVRAIT être > a_off - décalage de disposition
_READ	0	PEUT être _READ	DOIT être ≤ a_off	DEVRAIT être > a_off - décalage de disposition
_RW	u64m	DOIT être _RW	DOIT être ≤ a_off	DEVRAIT être u64m
_RW	> 0 et < u64m	DOIT être _RW	DOIT être ≤ a_off	DEVRAIT être ≥ a_off - décalage de disposition + a_len
_RW	0	DOIT être _RW	DOIT être ≤ a_off	DEVRAIT être ≥ a_off - décalage de disposition

Tableau 23

Le champ loga_stateid spécifie un identifiant d'état valide. Si une disposition n'est pas détenue actuellement par le client, le champ loga_stateid représente un identifiant d'état qui reflète l'ouverture, verrou de gamme d'octets, ou identifiant d'état de délégation valides correspondants. Une fois qu'une disposition est détenue sur le fichier par le client, le champ loga_stateid DOIT être un identifiant d'état comme retourné d'une précédente opération LAYOUTGET ou LAYOUTRETURN ou fournie par une opération CB_LAYOUTRECALL (voir le paragraphe 12.5.3).

Le champ loga_maxcount spécifie la taille maximum de disposition (in octets) que le client peut traiter. Si la taille de la structure de disposition excède la taille spécifiée par maxcount, le serveur de métadonnées va retourner l'erreur NFS4ERR_TOOSMALL.

La disposition retournée est exprimée comme un dispositif, logr_layout, avec chaque élément de type layout4. Si un fichier a un seul schéma de découpage en bandes, alors logr_layout DEVRAIT contenir juste une entrée. Autrement, si la gamme demandée chevauche plus d'un schéma de découpage en bandes, logr_layout va contenir le nombre d'entrées requis. Les éléments de logr_layour DOIVENT être triés en ordre ascendant de la valeur du champ lo_offset de chaque élément. Il NE DOIT PAS y avoir de trous ou chevauchements dans la gamme entre deux éléments successifs de logr_layout. Le champ lo_iomode dans chaque élément de logr_layoutn DOIT être le même.

Les tableaux 22 et 23 se réfèrent tous deux à l'iomode, décalage, et longueur de disposition retournée. Parce que la disposition retournée est codée dans le dispositif logr_layout, plus de description est nécessaire.

iomode : la valeur de l'iomode de disposition retournée mentionnée dans les tableaux 22 et 23 est égale à la valeur du champ lo_iomode dans chaque élément de logr_layout. Comme le montrent les tableaux 22 et 23, le serveur de métadonnées PEUT retourner une disposition avec un lo_iomode différent de l'iomode demandé (champ loga_iomode de la demande). Si il fait cela, il DOIT s'assurer que le lo_iomode est plus permissif que le loga_iomode demandé. Par exemple, ce comportement permet à une mise en œuvre de relever les demandes LAYOUTIOMODE4_READ en demandes LAYOUTIOMODE4_RW à sa discrétion, dans les limites du protocole spécifique du type de la disposition. Un lo_iomode de LAYOUTIOMODE4_READ ou LAYOUTIOMODE4_RW DOIT être retourné.

offset : la valeur du décalage de disposition retournée mentionnée dans les tableaux 22 et 23 est toujours égale au champ lo_offset du premier élément logr_layout.

length : quand on règle la valeur de la longueur de disposition retournée, la situation est compliquée par la possibilité que la valeur spéciale de longueur de disposition NFS4_UINT64_MAX soit impliquée. Pour un dispositif de logr_layout de N éléments, le champ lo_length dans les N-1 premiers éléments NE DOIT PAS être NFS4_UINT64_MAX. Le champ

lo_length du dernier élément de logr_layout peut être NFS4_UINT64_MAX dans certaines conditions comme décrit dans la liste suivante :

* Si une règle applicable du Tableau 22 déclare que le serveur de métadonnées DOIT retourner une disposition de longueur NFS4_UINT64_MAX, alors le champ lo_length du dernier élément de logr_layout DOIT être NFS4_UINT64_MAX.

* Si une règle applicable du Tableau 22 déclare que le serveur de métadonnées NE DOIT PAS retourner une disposition de longueur NFS4_UINT64_MAX, alors le champ lo_length du dernier élément de logr_layout NE DOIT PAS être NFS4_UINT64_MAX.

* Si une règle applicable du Tableau 23 déclare que le serveur de métadonnées DEVRAIT retourner une disposition de longueur NFS4_UINT64_MAX, alors le champ lo_length du dernier élément de logr_layout DEVRAIT être NFS4_UINT64_MAX.

* Quand la valeur de la longueur de la disposition retournée du Tableau 22 et du Tableau 23 n'est pas NFS4_UINT64_MAX, alors la longueur de la disposition retournée est égale à la somme des champs lo_length de chaque élément de logr_layout.

Le champ de résultat logr_return_on_close est une directive de retourner la disposition avant de clore le fichier. Quand le serveur de métadonnées règle cette valeur de retour à VRAI, il DOIT être prêt à rappeler la disposition dans le cas où le client échoue à retourner la disposition avant la clôture. Pour le serveur de métadonnées qui sait qu'une disposition doit être retournée avant la clôture du fichier, cette valeur de retour peut être utilisée pour communiquer le comportement désiré au client et donc supprimer une étape de l'interaction entre le client et le serveur de métadonnées.

L'identifiant d'état logr_stateid est retourné au client pour être utilisé dans les opérations suivantes relatives à la disposition. Voir aux paragraphes 8.2, 12.5.3, et 12.5.5.2 la discussion et les exigences.

Le format de la disposition retournée (lo_content) est spécifique du type de disposition. La valeur du type de disposition (lo_content.loc_type) pour chaque élément du dispositif de dispositions retournées par le serveur de métadonnées (logr_layout) DOIT être égal au loga_layout_type spécifié par le client. Si il n'est pas égal, le client DEVRAIT ignorer la réponse comme invalide et se comporter comme si le serveur de métadonnées avait retourné une erreur, même si le client prend en charge le type de disposition retourné.

Si ni le fichier demandé ni son système de fichiers contenant ne prennent en charge les dispositions, le serveur de métadonnées DOIT retourner NFS4ERR_LAYOUTUNAVAILABLE. Si le type de disposition n'est pas pris en charge, le serveur de métadonnées DOIT retourner NFS4ERR_UNKNOWN_LAYOUTTYPE. Si les dispositions sont prises en charge mais aucune disposition ne correspond à l'identification de disposition fournie par le client, le serveur de métadonnées DOIT retourner NFS4ERR_BADLAYOUT. Si un loga_iomode invalide est spécifié, ou si un loga_iomode de LAYOUTIOMODE4_ANY est spécifié, le serveur de métadonnées DOIT retourner NFS4ERR_BADIOMODE.

Si la disposition pour le fichier est indisponible du fait de conditions temporaires, par exemple, le partage de fichiers interdit les dispositions, le serveur de métadonnées DOIT retourner NFS4ERR_LAYOUTTRYLATER.

Si la demande de disposition est rejetée à cause d'un chevauchement de rappel de dispositions, le serveur de métadonnées DOIT retourner NFS4ERR_RECALLCONFLICT. Voir les détails au paragraphe 12.5.5.2.

Si la disposition est en conflit avec un verrou de gamme d'octets obligatoire détenu sur le fichier, et si les appareils de mémorisation n'ont pas de méthode pour appliquer les verrous obligatoires, autrement que par la restriction de dispositions, le serveur de métadonnées DEVRAIT retourner NFS4ERR_LOCKED.

Si le client règle loga_signal_layout_avail à VRAI, alors il enregistre avec le client un "want" pour une disposition dans le cas où la disposition ne peut pas être obtenue à cause de l'épuisement des ressources. Si le serveur de métadonnées prend en charge et honore le "want", le résultat va avoir logr_will_signal_layout_avail réglé à VRAI. Si il en est ainsi, le client devrait s'attendre à une opération CB_RECALLABLE_OBJ_AVAIL pour indiquer qu'une disposition est disponible.

En cas de succès, la bride de fichier courante conserve sa valeur et l'identifiant d'état courant est mis à jour pour correspondre à la valeur retournée dans le résultat.

18.43.4 Mise en œuvre

Normalement, LAYOUTGET va être invoqué au titre d'une demande COMPOUND après une opération OPEN et résulte en ce que le client a les informations de localisation pour le fichier. Cela exige que loga_stateid soit réglé à l'identifiant d'état spécial qui dit au serveur de métadonnées d'utiliser l'identifiant d'état courant, qui est établi par OPEN (voir le paragraphe 16.2.3.1.2). Un client peut aussi détenir une disposition à travers plusieurs OPEN. Le client spécifie un type de disposition qui limite quelle sorte de disposition le serveur de métadonnées va retourner. Cela empêche les serveurs de métadonnées d'accorder des dispositions inutilisables par le client.

Comme indiqué par les tableaux 22 et 23, la spécification de LAYOUTGET permet à un client et serveur pNFS une considérable souplesse. Un client pNFS peut avoir plusieurs stratégies d'envoi de LAYOUTGET. Des exemples suivent :

* Si LAYOUTGET est précédé par OPEN dans la même demande COMPOUND et si le OPEN demande l'accès OPEN4_SHARE_ACCESS_READ, le client pourrait opter pour demander une disposition _READ avec loga_offset réglé à zéro, loga_minlength réglé à zéro, et loga_length réglé à NFS4_UINT64_MAX. Si le fichier a l'espace alloué, cet espace est divisé en bandes sur un ou plusieurs appareils de mémorisation, et il n'y a pas de disposition en conflit ou le concept de disposition en conflit ne s'applique pas au type de disposition ou mise en œuvre de serveur pNFS, alors le serveur de métadonnées pourrait retourner une disposition avec un décalage de départ de zéro, et une longueur égale à la longueur du fichier, sinon NFS4_UINT64_MAX. Si la longueur du fichier n'est pas un multiple de la largeur de bande du serveur pNFS (voir au paragraphe 13.2 la définition formelle) le serveur de métadonnées pourrait arrondir la longueur de la disposition retournée.

* Si LAYOUTGET est précédé d'un OPEN dans la même demande COMPOUND, et si le OPEN demande l'accès OPEN4_SHARE_ACCESS_WRITE et ne tronque pas le fichier, le client pourrait opter pour demander une disposition _RW avec loga_offset réglé à zéro, loga_minlength réglé à zéro, et loga_length réglé à la longueur courante du fichier (si elle est connue) ou NFS4_UINT64_MAX. Comme avec le cas précédent, dans certaines conditions le serveur de métadonnées pourrait retourner une disposition qui couvre la longueur entière du fichier ou au delà.

* Cette stratégie est comme ci-dessus, mais le OPEN tronque le fichier. Dans ce cas, le client pourrait anticiper qu'il va écrire sur le fichier à partir du décalage zéro, et donc loga_offset et loga_minlength sont réglés à zéro, et loga_length est réglé à la valeur de threshold4_write_iosize. Le serveur de métadonnées pourrait retourner une disposition à partir du décalage zéro avec une longueur d'au moins threshold4_write_iosize.

* Un processus sur le client invoque une demande de lire à partir du décalage 10000 pou la longueur 50000. Le client utilise des entrées/sorties en antémémoire, et a des tailles d'antémémoire de 4096 octets. Le client entend transposer la demande de traitement en une série de demandes READ commençant au décalage 8192. La fin du décalage doit être supérieure à 10000 + 50000 = 60000, et le prochain décalage qui est un multiple de 4096 est 61440. La différence entre 61440 et ce décalage de début de la disposition est 53248 (qui est le produit de 4096 et 15). La valeur de threshold4_read_iosize est inférieure à 53248, donc le client envoie une demande LAYOUTGET avec loga_offset réglé à 8192, loga_minlength réglé à 53248, et loga_length réglé à la longueur du fichier (si elle est connue) moins 8192 ou NFS4_UINT64_MAX (si la longueur du fichier n'est pas connue). Comme cette demande LAYOUTGET excède le seuil du serveur de métadonnées, il accorde la disposition, éventuellement avec un décalage initial de zéro, avec une fin de décalage de au moins 8192 + 53248 - 1 = 61439, mais de préférence une disposition avec un décalage aligné sur la largeur de bande et une longueur qui est un multiple de la largeur de bande.

* Cette stratégie est comme ci-dessus, mais le client n'utilise pas l'entrée/sortie en antémémoire, et à la place toutes les demandes internes d'entrée/sortie sont envoyées directement au serveur. La demande LAYOUTGET a loga_offset égal à 10000 et loga_minlength réglé à 50000. La valeur de loga_length est réglée à la longueur du fichier. Le serveur de métadonnées est libre de retourner une disposition qui chevauche entièrement la gamme demandée, avec un décalage de début et une longueur alignées sur la largeur de bande.

* Là encore, un processus sur le client invoque une demande de lire à partir du décalage 10000 pour la longueur 50000 (c'est-à-dire une gamme avec un décalage de début de 10000 et un décalage de fin de 69999) et l'entrée/sortie en antémémoire est utilisée. Le client s'attend à ce que le serveur pourrait n'être pas capable de retourner la disposition pour la pleine gamme d'entrée/sortie. Le client entend transposer la demande de traitement en une série de treize demandes READ commençant au décalage 8192, chacune avec une longueur de 4096, avec une longueur totale de 53248 (ce qui est égal à 13 * 4096) qui contient entièrement la gamme que le processus de client veut lire. Parce que la valeur de threshold4_read_iosize est égale à 4096, il est praticable et raisonnable que le client utilise plusieurs opérations LAYOUTGET pour achever la série de READ. Le client envoie une demande LAYOUTGET avec loga_offset réglé à 8192, loga_minlength réglé à 4096, et loga_length réglé à 53248 ou plus. Le serveur va accorder une disposition éventuellement avec un décalage initial de zéro, avec un décalage de fin d'au moins 8192 + 4096 - 1 =

12287, mais de préférence une disposition avec un décalage aligné sur la largeur de bande et une longueur qui soit un multiple de la largeur de bande. Cela va permettre au client de progresser vers l'avant, éventuellement en envoyant plus d'opérations LAYOUTGET pour le reste de la gamme.

* Un client NFS détecte un schéma de lecture séquentiel, et envoie donc une opération LAYOUTGET qui va bien au-delà de toute demande de lecture courante ou en instance au serveur. Le serveur pourrait de même détecter ce schéma, et accorder la demande LAYOUTGET. Une fois que le client lit à partir d'un décalage du fichier qui représente 50 % du chemin à travers la gamme de la dernière disposition qu'il a reçu, afin d'éviter de bloquer les entrées/sorties qui attendraient une disposition, le client envoie plus d'opérations à partir d'un décalage du fichier qui représente 50 % du chemin à travers la dernière disposition reçue. Le client continue de demander des dispositions avec des gammes d'octets qui sont très en avance des gammes d'octets des demandes de traitement et/ou de lecture récentes en cours chez le client.

* Cette stratégie est la même que ci-dessus, mais le client échoue à détecter le schéma, mais le serveur le fait. La prochaine fois que le serveur de métadonnées obtient un LAYOUTGET, il retourne une disposition avec une longueur qui est bien au delà de loga_minlength.

* Un client utilise une entrée/sortie en antémémoire, et a une longue file d'attente d'écritures derrière à traiter et aussi détecte un schéma séquentiel d'écriture. Il envoie un LAYOUTGET pour une disposition qui s'étend sur la gamme des écritures derrière en file d'attente et bien au-delà, incluant des gammes au delà de la longueur actuelle du fichier. Le client continue d'envoyer des opérations LAYOUTGET une fois que la file d'attente d'écriture derrière atteint 50 % de la longueur maximum de file d'attente.

Une fois que le client a obtenu une disposition se référant à un identifiant d'appareil particulier, le serveur de métadonnées NE DOIT PAS supprimer l'identifiant d'appareil avant que la disposition soit retournée ou révoquée.

CB_NOTIFY_DEVICEID peut entrer en concurrence avec LAYOUTGET. Un scénario de concurrence est que LAYOUTGET retourne un identifiant d'appareil pour lequel le client n'a pas de transposition d'adresse d'appareil, et où le serveur de métadonnées envoie un CB_NOTIFY_DEVICEID pour ajouter l'identifiant d'appareil aux connaissances du client et pendant ce temps le client envoie GETDEVICEINFO sur l'identifiant d'appareil. Ce scénario est discuté au paragraphe 18.40.4. Un autre scénario est que le CB_NOTIFY_DEVICEID est traité par le client avant qu'il traite les résultats du LAYOUTGET. Le client va envoyer un GETDEVICEINFO sur l'identifiant d'appareil. Si le résultat du GETDEVICEINFO est reçu avant que le client obtienne le résultat du LAYOUTGET, alors il n'y a plus de concurrence. Si le résultat du LAYOUTGET est reçu avant celui du GETDEVICEINFO, le client peut soit attendre le résultat de GETDEVICEINFO, soit en envoyer un autre pour obtenir éventuellement des transpositions d'adresse d'appareil plus à jour pour l'identifiant d'appareil.

18.44 Opération 51 : LAYOUTRETURN - libère les informations de disposition

18.44.1 Argument

```
/* Constantes utilisées pour LAYOUTRETURN et CB_LAYOUTRECALL */
  const LAYOUT4_RET_REC_FILE      = 1;
  const LAYOUT4_RET_REC_FSID      = 2;
  const LAYOUT4_RET_REC_ALL       = 3;

enum layoutreturn_type4 {
        LAYOUTRETURN4_FILE = LAYOUT4_RET_REC_FILE,
        LAYOUTRETURN4_FSID = LAYOUT4_RET_REC_FSID,
        LAYOUTRETURN4_ALL  = LAYOUT4_RET_REC_ALL
};

struct layoutreturn_file4 {
        offset4         lrf_offset;
        length4         lrf_length;
        stateid4        lrf_stateid;
        /* données spécifiques de layouttype4 */
        opaque          lrf_body<>;
};
```

```
union layoutreturn4 switch(layoutreturn_type4 lr_returntype) {
      cas LAYOUTRETURN4_FILE : layoutreturn_file4      lr_layout;
      défaut : vide
};

struct LAYOUTRETURN4args {                                  /* CURRENT_FH : fichier */
      bool                  lora_reclaim;
      layouttype4           lora_layout_type;
      layoutiomode4         lora_iomode;
      layoutreturn4         lora_layoutreturn;
};
```

18.44.2 Résultat

```
union layoutreturn_stateid switch (bool lrs_present) {
  cas VRAI : stateid4               lrs_stateid;
  cas FAUX : vide
};

union LAYOUTRETURN4res switch (nfsstat4 lorr_status) {
  cas NFS4_OK : layoutreturn_stateid    lorr_stateid;
  défaut : vide
  };
```

18.44.3 Description

Cette opération retourne du client au serveur une ou plusieurs dispositions représentées par l'identifiant de client (déduites de l'identifiant de session dans la précédente opération SEQUENCE) lora_layout_type, et lora_iomode. Quand lr_returntype est LAYOUTRETURN4_FILE, la disposition retournée est encore identifiée par la bride de fichier courante, lrf_offset, lrf_length, et lrf_stateid. Si le champ lrf_length est NFS4_UINT64_MAX, tous les octets de la disposition, commençant à lrf_offset, sont retournés. Quand lr_returntype est LAYOUTRETURN4_FSID, la bride de fichier courante est utilisée pour identifier le système de fichiers et toutes les dispositions correspondant à l'identifiant de client, le fsid du système de fichiers, lora_layout_type, et lora_iomode sont retournés. Quand lr_returntype est LAYOUTRETURN4_ALL, toutes les dispositions correspondant à l'identifiant de client, lora_layout_type, et lora_iomode sont retournées et la bride de fichier courante n'est pas utilisée. Après cet appel, le client NE DOIT PAS utiliser la ou les dispositions retournées et le protocole de mémorisation associé pour accéder aux données du fichier.

Si l'ensemble de dispositions désignées dans le cas de LAYOUTRETURN4_FSID ou LAYOUTRETURN4_ALL est vide, alors aucune erreur ne résulte. Dans le cas de LAYOUTRETURN4_FILE, la gamme d'octets spécifiée est retournée même si elle est une subdivision d'une disposition précédemment obtenue avec LAYOUTGET, une combinaison de plusieurs dispositions précédemment obtenues avec LAYOUTGET, ou une combinaison incluant des dispositions précédemment obtenues avec LAYOUTGET, et une ou plusieurs subdivisions de telles dispositions. Quand la gamme d'octets ne désigne aucun octet pour lequel est détenue une disposition pour le fichier, identifiant de client, type de disposition et mode spécifiés, aucune erreur ne résulte. Voir au paragraphe 12.5.5.2.1.5 les considérations sur le "retour en vrac" de dispositions.

La disposition retournée peut être un sous ensemble ou un sur ensemble d'une disposition spécifiée par CB_LAYOUTRECALL. Cependant, si elle est un sous ensemble, le rappel n'est pas achevé tant que toute la portée rappelée n'a pas été retournée. Portée rappelée se réfère à la gamme d'octets dans le case de LAYOUTRETURN4_FILE, de l'utilisation de LAYOUTRETURN4_FSID, ou de l'utilisation de LAYOUTRETURN4_ALL. Il doit y avoir un LAYOUTRETURN avec une portée correspondante pour achever le retour même si toutes les gammes de disposition courantes ont été précédemment retournées individuellement.

Pour toutes les valeurs de lr_returntype, un iomode de LAYOUTIOMODE4_ANY spécifie que toutes les dispositions qui correspondent aux autres arguments de LAYOUTRETURN (c'est-à-dire, identifiant de client, lora_layout_type, et une de bride de fichier courante et gamme, fsid déduit de la bride de fichier courante, ou LAYOUTRETURN4_ALL) sont retournées.

Dans le cas où lr_returntype est LAYOUTRETURN4_FILE, le lrf_stateid fourni par le client est un identifiant d'état de

disposition comme retourné des opérations de disposition précédentes. Noter que le champ "seqid" de lrf_stateid NE DOIT PAS être zéro. Voir les paragraphes 8.2, 12.5.3, et 12.5.5.2 pour la discussion et les exigences.

Le retour d'une disposition ou de toutes les dispositions n'invalide pas la transposition d'identifiant d'appareil de mémorisation en adresse d'appareil de mémorisation. La transposition reste en vigueur jusqu'à ce qu'elle soit spécifiquement changée ou supprimée via des rappels de notification d'identifiant d'appareil. Bien sûr, si il ne reste pas de disposition qui se réfère à un identifiant d'appareil précédemment utilisé, le serveur est libre de supprimer un identifiant d'appareil sans un rappel de notification, ce qui va être le cas quand les notifications ne sont pas en vigueur.

Si le champ lora_reclaim est réglé à VRAI, le client tente de retourner une disposition qui a été acquise avant le redémarrage du serveur de métadonnées durant la période de grâce du serveur de métadonnées. Quand il retourne des dispositions qui ont été acquises durant la période de grâce du serveur de métadonnées, le client DOIT régler le champ lora_reclaim à FAUX. Le champ lora_reclaim DOIT être réglé à FAUX aussi quand lr_layoutreturn est LAYOUTRETURN4_FSID ou LAYOUTRETURN4_ALL. Voir LAYOUTCOMMIT (paragraphe 18.42) pour les détails.

Les dispositions peuvent être retournées quand elles sont rappelées ou volontairement (c'est-à-dire, avant que le serveur les aient rappelées). Dans l'un et l'autre cas, le client doit correctement propager l'état changé dans le contexte de la disposition à l'appareil de mémorisation ou au serveur de métadonnées avant de retourner la disposition.

Si le client retourne la disposition en réponse à un CB_LAYOUTRECALL lorsque le champ lor_recalltype du champ clora_recall était LAYOUTRECALL4_FILE, le client devrait utiliser la valeur de lor_stateid provenant de CB_LAYOUTRECALL comme valeur pour lrf_stateid. Autrement, il devrait utiliser logr_stateid (provenant d'un précédent résultat de LAYOUTGET) ou lorr_stateid (provenant d'un précédent résultat de LAYRETURN). Ceci est fait pour indiquer le moment (en termes de transitions d'identifiant d'état de disposition) où le rappel a été envoyé. Le client utilise la valeur précise lora_recallstateid et NE DOIT PAS régler le seqid de l'identifiant d'état à zéro; autrement, NFS4ERR_BAD_STATEID DOIT être retourné. NFS4ERR_OLD_STATEID peut être retourné si le client utilise un vieux seqid, et si le serveur sait que le client ne devrait pas utiliser le vieux seqid. Par exemple, le client utilise le seqid sur le créneau 1 de la session, reçoit la réponse avec le nouveau seqid, et utilise le créneau pour envoyer une autre demande avec le vieux seqid.

Si un client échoue à retourner une disposition en temps utile, alors le serveur de métadonnées DEVRAIT utiliser son protocole de contrôle avec les appareils de mémorisation pour barricader l'accès du client aux données référencées par la disposition. Voir au paragraphe 12.5.5 les détails.

Si la demande LAYOUTRETURN règle le champ lora_reclaim à VRAI après la période de grâce du serveur de métadonnées, NFS4ERR_NO_GRACE est retourné.

Si la demande LAYOUTRETURN règle le champ lora_reclaim à VRAI et si lr_returntype est réglé à LAYOUTRETURN4_FSID ou LAYOUTRETURN4_ALL, NFS4ERR_INVAL est retourné.

Si le client règle le champ lr_returntype à LAYOUTRETURN4_FILE, alors le champ lrs_stateid va représenter l'identifiant d'état de disposition comme mis à jour pour le traitement de cette opération ; l'identifiant d'état courant va aussi être mis à jour pour correspondre à la valeur retournée. Si le dernier octet de toute disposition pour le fichier courant, l'identifiant de client, et le type de disposition est retourné et si il n'y a pas d'opération CB_LAYOUTRECALL restante en instance pour laquelle une opération LAYOUTRETURN doive être faite, lrs_present DOIT être FAUX, et aucun identifiant d'état ne va être retourné. De plus, l'identifiant d'état courant de demande COMPOUND va être réglé à l'identifiant d'état spécial tout de zéros (voir le paragraphe 16.2.3.1.2). Le serveur DOIT rejeter avec NFS4ERR_BAD_STATEID tout autre utilisation de l'identifiant d'état courant dans cette COMPOUND jusqu'à ce que l'identifiant d'état courant soit rétabli par une opération ultérieure de retour d'identifiant d'état.

En cas de succès, la bride de fichier courante conserve sa valeur.

Si la capacité EXCHGID4_FLAG_BIND_PRINC_STATEID est établie sur l'identifiant de client (voir le paragraphe 18.35) le serveur va exiger que la combinaison de principal, nuance de sécurité, et si applicable, mécanisme GSS, qui a acquis la disposition soit aussi celle qui envoie LAYOUTRETURN. Cela pourrait n'être pas possible si les accréditifs pour le principal ne sont plus disponibles. Le serveur va permettre que l'accréditif de machine ou l'accréditif de SSV (voir le paragraphe 18.35) envoie le LAYOUTRETURN si le code d'opération du LAYOUTRETURN était établi dans le résultat spo_must_allow de EXCHANGE_ID.

18.44.4 Mise en œuvre

L'opération LAYOUTRETURN finale en réponse à un rappel CB_LAYOUTRECALL DOIT être mise en série avec toute opération LAYOUTRETURN en instance qui la coupe. Noter qu'il est possible qu'alors qu'un client retourne la disposition pour une gamme rappelée, le serveur puisse rappeler un surensemble de cette gamme (par exemple, LAYOUTRECALL4_ALL) ; l'opération de retour finale pour cette dernière doit bloquer jusqu'à ce que le rappel de la première disposition soit fait.

Retourner toutes les dispositions dans un système de fichiers utilisant LAYOUTRETURN4_FSID est normalement fait en réponse à un CB_LAYOUTRECALL pour ce système de fichiers comme opération de retour finale. De même, LAYOUTRETURN4_ALL est utilisé en réponse à un rappel pour toutes les dispositions. Il est possible que le client ait déjà retourné des dispositions en instance via des appels LAYOUTRETURN individuels et que l'invocation de LAYOUTRETURN4_FSID ou LAYOUTRETURN4_ALL marque la fin de la séquence de LAYOUTRETURN. Voir au paragraphe 12.5.5.1 les détails.

Une fois que le client a retourné toutes les dispositions se référant à un identifiant d'appareil particulier, le serveur PEUT supprimer l'identifiant d'appareil.

18.45 Opération 52 : SECINFO_NO_NAME - obtient la sécurité sur un objet non nommé

18.45.1 Argument

```
enum secinfo_style4 {
      SECINFO_STYLE4_CURRENT_FH      = 0,
      SECINFO_STYLE4_PARENT          = 1.
};

/* CURRENT_FH : répertoire objet ou fils */
typedef secinfo_style4 SECINFO_NO_NAME4args;
```

18.45.2 Résultat

```
/* CURRENTFH : consommé si l'état est NFS4_OK */
typedef SECINFO4res SECINFO_NO_NAME4res;
```

18.45.3 Description

Comme l'opération SECINFO, SECINFO_NO_NAME est utilisé par le client pour obtenir une liste de nuances valides d'authentification RPC pour un objet de fichier spécifique. À la différence de SECINFO, SECINFO_NO_NAME fonctionne seulement avec les objets qui sont accédés par une bride de fichier.

Il y a deux styles de SECINFO_NO_NAME, comme déterminé par la valeur de l'énumération secinfo_style4. Si SECINFO_STYLE4_CURRENT_FH est passé, alors SECINFO_NO_NAME interroge sur la sécurité demandée pour la bride de fichier courante. Si SECINFO_STYLE4_PARENT est passé, alors SECINFO_NO_NAME interroge sur la sécurité demandée sur le parent de la bride de fichier courante. Si le style choisi est SECINFO_STYLE4_PARENT, alors SECINFO devrait appliquer la même méthodologie d'accès qu'utilisée pour LOOKUPP dans l'évaluation de la traversée au répertoire parent. Donc, si le demandeur n'a pas l'accès approprié au LOOKUPP parent, alors SECINFO_NO_NAME doit se comporter de la même façon et retourner NFS4ERR_ACCESS.

Si PUTFH, PUTPUBFH, PUTROOTFH, ou RESTOREFH retourne NFS4ERR_WRONGSEC, alors le client résout la situation en envoyant une demande COMPOUND qui consiste en PUTFH, PUTPUBFH, ou PUTROOTFH immédiatement suivi de SECINFO_NO_NAME, style SECINFO_STYLE4_CURRENT_FH. Voir au paragraphe 2.6 les instructions pour traiter l'erreur NFS4ERR_WRONGSEC retournée de PUTFH, PUTROOTFH, PUTPUBFH, ou RESTOREFH.

Si SECINFO_STYLE4_PARENT est spécifié et qu'il n'y a pas de répertoire parent, SECINFO_NO_NAME DOIT retourner NFS4ERR_NOENT.

En cas de succès, la bride de fichier courante est consommée (voir le paragraphe 2.6.3.1.1.8) et si la prochaine opération après SECINFO_NO_NAME essaye d'utiliser la bride de fichier courante, cette opération va échouer avec l'état

NFS4ERR_NOFILEHANDLE.

Tout le reste sur SECINFO_NO_NAME est le même que pour SECINFO. Voir la discussion sur SECINFO (paragraphe 18.29.3).

18.45.4 Mise en œuvre

Voir la discussion sur SECINFO (paragraphe 18.29.4).

18.46 Opération 53 : SEQUENCE - fournit le séquençage et le contrôle par procédure

18.46.1 Argument

```
struct SEQUENCE4args {
        sessionid4      sa_sessionid;
        sequenceid4     sa_sequenceid;
        slotid4         sa_slotid;
        slotid4         sa_highest_slotid;
        bool            sa_cachethis;
};
```

18.46.2 Résultat

```
const SEQ4_STATUS_CB_PATH_DOWN                  = 0x00000001;
const SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRING      = 0x00000002;
const SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRED       = 0x00000004;
const SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED     = 0x00000008;
const SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED    = 0x00000010;
const SEQ4_STATUS_ADMIN_STATE_REVOKED           = 0x00000020;
const SEQ4_STATUS_RECALLABLE_STATE_REVOKED      = 0x00000040;
const SEQ4_STATUS_LEASE_MOVED                   = 0x00000080;
const SEQ4_STATUS_RESTART_RECLAIM_NEEDED        = 0x00000100;
const SEQ4_STATUS_CB_PATH_DOWN_SESSION          = 0x00000200;
const SEQ4_STATUS_BACKCHANNEL_FAULT             = 0x00000400;
const SEQ4_STATUS_DEVID_CHANGED                 = 0x00000800;
const SEQ4_STATUS_DEVID_DELETED                 = 0x00001000;

struct SEQUENCE4resok {
        sessionid4      sr_sessionid;
        sequenceid4     sr_sequenceid;
        slotid4         sr_slotid;
        slotid4         sr_highest_slotid;
        slotid4         sr_target_highest_slotid;
        uint32_t        sr_status_flags;
};

union SEQUENCE4res switch (nfsstat4 sr_status) {
  cas NFS4_OK : SEQUENCE4resok  sr_resok4;
  défaut : vide
};
```

18.46.3 Description

L'opération SEQUENCE est utilisée par le serveur pour mettre en œuvre ie contrôle de demande de session et la sémantique de mémoire tampon de réponse.

SEQUENCE DOIT apparaître comme la première opération de tout COMPOUND dans lequel elle apparaît. L'erreur NFS4ERR_SEQUENCE_POS va être retournée quand elle est trouvée dans toute position dans un COMPOUND au delà

de la première. Les opérations autres que SEQUENCE, BIND_CONN_TO_SESSION, EXCHANGE_ID, CREATE_SESSION, et DESTROY_SESSION, NE DOIVENT PAS apparaître comme première opération dans un COMPOUND. De telles opérations DOIVENT donner l'erreur NFS4ERR_OP_NOT_IN_SESSION si elles apparaissent au début d'un COMPOUND.

Si SEQUENCE est reçu sur une connexion non associée à la session via CREATE_SESSION ou BIND_CONN_TO_SESSION, et si l'application d'association de connexion est activée (voir le paragraphe 18.35) alors le serveur retourne NFS4ERR_CONN_NOT_BOUND_TO_SESSION.

L'argument sa_sessionid identifie la session à laquelle cette demande s'applique. Le résultat sr_sessionid DOIT être égal à sa_sessionid.

L'argument sa_slotid est l'indice dans la mémoire tampon de réponse pour la demande. Le champ sa_sequenceid est le numéro de séquence de la demande pour l'entrée de mémoire tampon de réponse (créneau). Le résultat sr_slotid DOIT être égal à sa_slotid. Le résultat sr_sequenceid DOIT être égal à sa_sequenceid.

L'argument sa_highest_slotid est le plus fort identifiant de créneau pour lequel le client a une demande en instance ; il pourrait être égal à sa_slotid. Le serveur retourne deux valeurs de "highest_slotid" : sr_highest_slotid et sr_target_highest_slotid. La première est le plus fort identifiant de créneau que le serveur va accepter dans une future opération SEQUENCE, et NE DEVRAIT PAS être moins que la valeur de sa_highest_slotid (mais voir une exception au paragraphe 2.10.6.1). La dernière est le plus fort identifiant de créneau que le serveur préférerait qu'utilise le client sur une future opération SEQUENCE.

Si sa_cachethis est VRAI, alors le client demande que le serveur mette en antémémoire la réponse entière dans la mémoire tampon de réponse du serveur ; donc, le serveur DOIT mettre la réponse en antémémoire (voir le paragraphe 2.10.6.1.3). Le serveur PEUT mettre la réponse en antémémoire si sa_cachethis est FAUX. Si le serveur ne met pas la réponse entière en antémémoire, il DOIT quand même enregistrer qu'il a exécuté la demande au créneau et identifiant de séquence spécifiés.

La réponse à l'opération SEQUENCE contient un mot de fanions d'état (sr_status_flags) qui peuvent fournir au client des informations sur l'état de l'état de verrouillage et des chemins de communications du client. Noter que tous les bits d'état relatifs à l'état de verrouillage PEUVENT être réinitialisés quand l'état de verrouillage est perdu à cause d'un redémarrage de serveur (même si la session est persistante à travers le redémarrage ; la persistance de session n'implique pas de persistance d'état de verrouillage) ni l'établissement d'une nouvelle instance de client.

SEQ4_STATUS_CB_PATH_DOWN : quand il est établi, il indique que le client n'a pas de chemin de canal de retour opérationnel pour une session associée à l'identifiant de client, rendant nécessaire que le client en rétablisse un. Ce bit reste établi sur toutes les réponses SEQUENCE sur toutes les sessions associées à l'identifiant de client jusqu'à ce que au moins un canal de retour soit disponible sur une session associée à l'identifiant de client. Si le client échoue à rétablir un canal de retour pour l'identifiant de client, il est susceptible d'avoir un état rappelable révoqué.

SEQ4_STATUS_CB_PATH_DOWN_SESSION : quand il est établi, il indique que la session n'a pas de canal de retour opérationnel. Il y a deux raisons pour lesquelles SEQ4_STATUS_CB_PATH_DOWN_SESSION peut être établi et pas SEQ4_STATUS_CB_PATH_DOWN. D'abord c'est une opération de rappel qui s'applique spécifiquement à la session (par exemple, CB_RECALL_SLOT, voir le paragraphe 20.8) qui a besoin d'être envoyée. La seconde est que le serveur a envoyé une opération de rappel, mais la connexion a été perdue avant la réponse. Le serveur ne peut pas être sûr de si le client a reçu ou non l'opération de rappel, et donc, selon les règles de ré-essai de demande, le serveur DOIT réessayer l'opération de rappel sur la même session. Le bit SEQ4_STATUS_CB_PATH_DOWN_SESSION est l'indication au client qu'il doit associer une connexion au canal de retour de la session. Ce bit reste établi sur toutes les réponses à SEQUENCE de la session jusqu'à ce qu'une connexion soit associée à un canal de retour de la session. Si le client échoue à rétablir un canal de retour pour la session, il est susceptible d'avoir l'état rappelable révoqué.

SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRING : quand il est établi, il indique que toutes les brides de contexte GSS ou RPCSEC_GSS allouées au canal de retour de la session vont expirer dans une période égale au temps de prêt. Ce bit reste établi sur toutes les réponses SEQUENCE jusqu'à ce que au moins une des conditions suivantes soit vraie :

* Toutes les brides SSV RPCSEC_GSS sur le canal de retour de la session ont été détruites et tous les contextes GSS non SSV ont expiré.
* Au moins une bride SSV RPCSEC_GSS supplémentaire a été ajoutée au canal de retour.
* Le temps d'expiration de au moins un contexte GSS non SSV d'une bride RPCSEC_GSS est au delà de la période de prêt à partir de l'instant présent (par rapport à l'instant d'envoi de la réponse SEQUENCE).

SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRED : quand il est établi, il indique que tous les contextes GSS non SSV et

toutes les brides RPCSEC_GSS de SSV alloués au canal de retour de la session ont expiré ou ont été détruits. Ce bit reste établi sur toutes les réponses SEQUENCE jusqu'à ce qu'au moins un contexte GSS non SSV non expiré pour le canal de retour de la session ait été établi ou au moins une bride SSV RPCSEC_GSS ait été allouée au canal de retour.

SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED : quand il est établi, il indique que le prêt a expiré et que par suite le serveur a libéré tout l'état de verrouillage du client. Ce bit d'état reste établi sur toutes les réponses SEQUENCE jusqu'à ce que la perte de tous ces verrous ait été acquittée par l'utilisation de FREE_STATEID (voir le paragraphe 18.38) ou en établissant une nouvelle instance de client en détruisant toutes les sessions (via DESTROY_SESSION) l'identifiant de client (via DESTROY_CLIENTID) et en invoquant alors EXCHANGE_ID et CREATE_SESSION pour établir un nouvel identifiant de client.

SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED : quand il est établi, il indique qu'un certain sous ensemble des verrous du client ont été révoqués à cause de l'expiration de la période de prêt suivie par une autre opération LOCK en conflit du client. Ce bit d'état reste établi sur toutes les réponses SEQUENCE jusqu'à ce que la perte de tous ces verrous ait été acquittée par l'utilisation de FREE_STATEID.

SEQ4_STATUS_ADMIN_STATE_REVOKED : quand il est établi, il indique que un ou plusieurs verrous ont été révoqués sans expiration de la période de prêt, du fait d'une action administrative. Ce bit d'état reste établi sur toutes les réponses SEQUENCE jusqu'à ce que la perte de tous ces verrous ait été acquittée par l'utilisation de FREE_STATEID.

SEQ4_STATUS_RECALLABLE_STATE_REVOKED : quand il est établi, il indique que un ou plusieurs objets rappelables ont été révoqués sans expiration de la période de prêt, du fait d'une défaillance du client à les retourner quand ils ont été rappelés, qui peut être une conséquence de l'absence de canal de retour opérationnel et de l'échec du client à rétablir un canal de retour conformément aux fanions d'état SEQ4_STATUS_CB_PATH_DOWN, SEQ4_STATUS_CB_PATH_DOWN_SESSION, ou SEQ4_STATUS_CB_GSS_CONTEXTS_EXPIRED. Ce bit d'état reste établi sur toutes les réponses SEQUENCE jusqu'à ce que la perte de tous ces verrous ait été acquittée par l'utilisation de FREE_STATEID.

SEQ4_STATUS_LEASE_MOVED : quand il est établi, il indique que la responsabilité du renouvellement du prêt a été transférée à un ou plusieurs nouveaux serveurs. Cette condition va continuer jusqu'à ce que le client reçoive une erreur NFS4ERR_MOVED et que le serveur reçoive le GETATTR suivant pour l'attribut fs_locations ou fs_locations_info pour un accès à chaque système de fichiers pour lequel un prêt a été déplacé à un nouveau serveur. Voir au paragraphe 11.11.9.2.

SEQ4_STATUS_RESTART_RECLAIM_NEEDED : quand il est établi, il indique que du fait du redémarrage du serveur, le client doit réclamer l'état de verrouillage. Jusqu'à ce que le client envoie un RECLAIM_COMPLETE global (paragraphe 18.51) chaque opération SEQUENCE va retourner SEQ4_STATUS_RESTART_RECLAIM_NEEDED.

SEQ4_STATUS_BACKCHANNEL_FAULT : le serveur a rencontré une faute irrécupérable avec le canal de retour (par exemple, il a perdu la trace de l'identifiant de séquence pour un créneau dans le canal de retour). Le client DOIT cesser d'envoyer d'autres demandes sur le canal avant de la session, attendre que toutes les demandes en instance s'achèvent sur le canal avant et de retour, et alors détruire la session.

SEQ4_STATUS_DEVID_CHANGED : le client utilise des notifications d'identifiant d'appareil et le serveur a changé une transposition d'identifiant d'appareil détenu par le client. Ce fanion va rester présent jusqu'à ce que le client ait obtenu la nouvelle transposition avec GETDEVICEINFO.

SEQ4_STATUS_DEVID_DELETED : le client utilise des notifications d'identifiant d'appareil et le serveur a supprimé une transposition d'identifiant d'appareil détenue par le client. Ce fanion va rester en vigueur jusqu'à ce que le client envoie un GETDEVICEINFO sur l'identifiant d'appareil avec une valeur nulle dans l'argument gdia_notify_types.

La valeur de l'argument sa_sequenceid relative à l'identifiant de séquence en antémémoire sur le créneau entre dans un des trois cas suivants :

* Si la différence entre sa_sequenceid et l'identifiant de séquence en antémémoire du serveur à l'identifiant de créneau est deux (2) ou plus, ou si sa_sequenceid est moins que l'identifiant de séquence en antémémoire (en tenant compte du retour à zéro de la valeur non signée de l'identifiant de séquence) alors le serveur DOIT retourner NFS4ERR_SEQ_MISORDERED.

* Si sa_sequenceid et l'identifiant de séquence en antémémoire sont le même, c'est un ré-essai, et le serveur répond avec ce qui est enregistré dans la mémoire tampon de réponse. Le prêt est éventuellement renouvelé comme décrit ci-

dessous.

* Si sa_sequenceid est supérieur de un (en tenant compte du retour à zéro) à l'identifiant de séquence en antémémoire, alors c'est une nouvelle demande, et l'identifiant de séquence du créneau est incrémenté. Les opérations suivantes de SEQUENCE, si il en est, sont traitées. Si il n'y a pas d'autre opération, les seuls autres effets sont de mettre en antémémoire la réponse de SEQUENCE dans le créneau, de maintenir l'activité de la session, et éventuellement de renouveler le prêt.

Si le client réutilise un identifiant de créneau et un identifiant de séquence pour une demande complètement différente, le serveur PEUT traiter la demande comme si c'était un ré-essai de ce qu'il a déjà exécuté. Le serveur PEUT cependant détecter la réutilisation illégale du client et retourner NFS4ERR_SEQ_FAUX_RETRY.

Si SEQUENCE retourne une erreur, alors l'état du créneau (identifiant de séquence, réponse en antémémoire) NE DOIT PAS changer, et le prêt associé NE DOIT PAS être renouvelé.

Si SEQUENCE retourne NFS4_OK, alors le prêt associé DOIT être renouvelé (voir le paragraphe 8.3) sauf si SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED est retourné dans sr_status_flags.

18.46.4 Mise en œuvre

Le serveur DOIT maintenir une transposition d'identifiant de session à identifiant de client afin de valider toutes les opérations qui suivent SEQUENCE et prennent un identifiant d'état comme argument et/ou résultat.

Si le client établit une session persistante, alors un SEQUENCE reçu après un redémarrage de serveur pourrait rencontrer des demandes effectuées et enregistrées dans une mémoire tampon de réponse persistante avant le redémarrage de serveur. Dans ce cas, le SEQUENCE va être traité avec succès, tandis que les demandes qui n'étaient pas précédemment effectuées et enregistrées sont rejetées avec NFS4ERR_DEADSESSION.

Selon les opérations dans le COMPOUND qui ont été effectuées avec succès avant le redémarrage de serveur, ces opérations vont aussi avoir des réponses envoyées depuis la mémoire tampon de réponse du serveur. Noter que quand ces opérations établissent un état de verrouillage, c'est l'état de verrouillage qui s'applique à la précédente instance de serveur et au précédent identifiant de client, même si le redémarrage de serveur, qui est arrivé logiquement après ces opérations, a éliminé cet état. Dans le cas d'un COMPOUND partiellement exécuté, le traitement peut atteindre une opération non traitée durant l'instance précédente de serveur, faisant de cette opération une nouvelle opération et non exécutable sur la session existante. Dans ce cas, NFS4ERR_DEADSESSION va être retourné de cette opération.

18.47 Opération 54 : SET_SSV - met à jour le SSV pour un identifiant de client

18.47.1 Argument

```
struct ssa_digest_input4 {
        SEQUENCE4args sdi_seqargs;
};

struct SET_SSV4args {
      opaque        ssa_ssv<>;
      opaque        ssa_digest<>;
};
```

18.47.2 Résultat

```
struct ssr_digest_input4 {
        SEQUENCE4res sdi_seqres;
};

struct SET_SSV4resok {
      opaque        ssr_digest<>;
};
```

```
union SET_SSV4res switch (nfsstat4 ssr_status) {
  cas NFS4_OK : SET_SSV4resok   ssr_resok4;
  défaut : vide
};
```

18.47.3 Description

Cette opération est utilisée pour mettre à jour le SSV pour un identifiant de client. Avant que SET_SSV soit invoqué la première fois sur un identifiant de client, le SSV est zéro. Le SSV est la clé utilisée pour le mécanisme GSS de SSV (paragraphe 2.10.9)

SET_SSV DOIT être précédé d'une opération SEQUENCE dans le même COMPOUND. Il NE DOIT PAS être utilisé si le client n'a pas opté pour la protection d'état SP4_SSV quand l'identifiant de client a été créé (voir le paragraphe 18.35) ; le serveur retourne NFS4ERR_INVAL dans ce cas.

Le champ ssa_digest est calculé comme le résultat du HMAC [RFC2104] en utilisant la sous clé déduite de SSV4_SUBKEY_MIC_I2T et le SSV courant comme clé (voir le paragraphe 2.10.9 pour la description des sous clés) et une valeur de type de données ssa_digest_input4 codée en XDR. Le champ sdi_seqargs est égal aux arguments de l'opération SEQUENCE pour la procédure COMPOUND dans laquelle est SET_SSV.

L'argument ssa_ssv est OUXé avec le SSV courant pour produire le nouveau SSV. L'argument ssa_ssv DEVRAIT être généré au hasard.

Dans la réponse, ssr_digest est le résultat du HMAC en utilisant la sous clé déduite de SSV4_SUBKEY_MIC_T2I et du nouveau SSV comme clé, et une valeur codée en XDR de type de données ssr_digest_input4. Le champ sdi_seqres est égal au résultat de l'opération SEQUENCE pour la procédure COMPOUND dans laquelle est SET_SSV.

Comme noté au paragraphe 18.35, le client et le serveur peuvent maintenir plusieurs versions concurrentes du SSV. Le client et le serveur DOIVENT chacun maintenir un numéro de version de SSV interne, qui est réglé à un la première fois que SET_SSV s'exécute sur le serveur et que le client reçoit la première réponse SET_SSV. Chaque SET_SSV suivant augmente de un le numéro interne de version de SSV. La valeur de ce numéro de version correspond aux champs smpt_ssv_seq, smt_ssv_seq, sspt_ssv_seq, et ssct_ssv_seq des jetons de mécanisme GSS de SSV (voir le paragraphe 2.10.9).

18.47.4 Mise en œuvre

Quand le serveur reçoit ssa_digest, il DOIT vérifier le résumé en calculant le résumé de la même façon que l'a fait le client et en le comparant avec ssa_digest. Si le serveur obtient un résultat différent, c'est l'erreur NFS4ERR_BAD_SESSION_DIGEST. Cette erreur pourrait être le résultat d'un autre SET_SSV provenant du même identifiant de client qui change le SSV. Si il en est ainsi, le client récupère en envoyant à nouveau une opération SET_SSV avec un résumé recalculé sur la base de la sous clé du nouveau SSV. Si la connexion de transport est éliminée après l'envoi de la demande SET_SSV, mais avant que la réponse SET_SSV soit reçue, alors il y a des considérations particulières pour la récupération si le client n'a plus de connexion associée à des sessions associées à l'identifiant de client du SSV. Voir au paragraphe 18.34.4.

Les clients NE DEVRAIENT PAS envoyer un ssa_ssv égal à un précédent ssa_ssv, ni égal à un SSV précédent ou courant (y compris un ssa_ssv égal à zéro car le SSV est initialisé à zéro quand l'identifiant de client est créé).

Les clients DEVRAIENT envoyer SET_SSV avec la confidentialité RPCSEC_GSS. Les serveurs DOIVENT prendre en charge RPCSEC_GSS avec confidentialité pour tout COMPOUND qui a { SEQUENCE, SET_SSV }.

Un client NE DEVRAIT PAS envoyer SET_SSV avec l'accréditif du mécanisme GSS de SSV parce que l'objet du SET_SSV est de faire naître le SSV à partir d'accréditifs non SSV. À la place, SET_SSV DEVRAIT être envoyé avec l'accréditif d'un utilisateur qui accède à l'identifiant de client pour la première fois (paragraphe 2.10.8.3). Cependant, si le client n'envoie pas SET_SSV avec les accréditifs de SSV, le résumé qui protège les arguments utilise la valeur du SSV avant que ssa_ssv soit OUXé, et le résumé qui protège le résultat utilise la valeur de SSV après que ssa_ssv soit OUXé.

18.48 Opération 55 : TEST_STATEID - essai de validité des identifiants d'état

18.48.1 Argument

```
struct TEST_STATEID4args {
      stateid4       ts_stateid<>;
};
```

18.48.2 Résultat

```
struct TEST_STATEID4resok {
      nfsstat4       tsr_status_codes<>;
};

union TEST_STATEID4res switch (nfsstat4 tsr_status) {
   cas NFS4_OK : TEST_STATEID4resok tsr_resok4;
   défaut : vide
};
```

18.48.3 Description

L'opération TEST_STATEID est utilisée pour vérifier la validité d'un ensemble d'identifiants d'état. Elle peut être utilisée à tout moment, mais le client devrait définitivement l'utiliser quand il reçoit l'indication que un ou plusieurs de ses identifiants d'état ont été invalidés du fait de la révocation du verrou. Cela se produit quand l'opération SEQUENCE retourne un des fanions sr_status_flags suivants établis :

* SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED
* SEQ4_STATUS_EXPIRED_ADMIN_STATE_REVOKED
* SEQ4_STATUS_EXPIRED_RECALLABLE_STATE_REVOKED

Le client peut utiliser TEST_STATEID une ou plusieurs fois pour vérifier la validité de ses identifiants d'état. Chaque utilisation de TEST_STATEID permet de vérifier un large ensemble d'identifiants d'état et évite des problèmes d'interférence avec les identifiants d'état antérieurs dans une demande COMPOUND en vérifiant les identifiants d'état suivants, comme on ferait si les identifiants d'état individuels étaient testés par une série d'opérations correspondantes dans une demande COMPOUND.

Pour chaque identifiant d'état, le serveur retourne le code d'état qui serait retourné si cet identifiant d'état devait être utilisé en fonctionnement normal. Retourner une telle indication d'état n'est pas une erreur et ne cause pas la fin du traitement COMPOUND. Les vérifications de validité de l'identifiant d'état procèdent comme pour les opérations normales avec quelques exceptions :

* Il n'y a pas de vérification du type de l'objet d'identifiant d'état, comme ce serait le cas pour l'utilisation normale d'un identifiant d'état.
* Il n'y a pas de référence à la bride de fichier courante.
* Les identifiants d'état spéciaux sont toujours considérés invalides (ils résultent en le code d'erreur NFS4ERR_BAD_STATEID).

Tous les identifiants d'état sont interprétés comme étant associés au client pour la session courante. Toute association possible avec une précédente instance du client (comme des identifiants d'état périmés) n'est pas considérée.

Les valeurs d'état valides dans le dispositif status_code retourné sont NFS4ERR_OK, NFS4ERR_BAD_STATEID, NFS4ERR_OLD_STATEID, NFS4ERR_EXPIRED, NFS4ERR_ADMIN_REVOKED, et NFS4ERR_DELEG_REVOKED.

18.48.4 Mise en œuvre

Voir aux paragraphes 8.2.2 et 8.2.4 la discussion de la structure, durée de vie, et validation de l'identifiant d'état.

18.49 Opération 56 : WANT_DELEGATION - demande de délégation

18.49.1 Argument

```
union deleg_claim4 switch (open_claim_type4 dc_claim) {
/* * Aucun droit spécial sur l'objet. Demande ordinaire de délégation de l'objet spécifié. Objet identifié par bride de fichier.
*/
  cas CLAIM_FH :   vide                                   /* nouveau pour v4.1 */
                                                          /* CURRENT_FH : objet délégué */

/* * Droit au fichier fondé sur une délégation accordée à une instance d'amorçage précédente du client. Le fichier est
spécifié par la bride de fichier. */
  cas CLAIM_DELEG_PREV_FH :    vide                       /* nouveau pour v4.1 */
                                                          /* CURRENT_FH : objet délégué */

/* * Droit au fichier établi par une ouverture précédant le réamorçage du serveur. Fichier identifié par bride de fichier.
Utilisé durant la période de grâce de réclamation du serveur. */
  cas CLAIM_PREVIOUS :  open_delegation_type4   dc_delegate_type;
                                                          /* CURRENT_FH : objet réclamé */
};

struct WANT_DELEGATION4args {
      uint32_t          wda_want;
      deleg_claim4    wda_claim;
};
```

18.49.2 Résultat

```
union WANT_DELEGATION4res switch (nfsstat4 wdr_status) {
  cas NFS4_OK : open_delegation4   wdr_resok4;
  défaut : vide
};
```

18.49.3 Description

Lorsque cette description rend obligatoire le retour d'un code d'erreur spécifique pour une condition spécifique, et lorsque plusieurs conditions s'appliquent, le serveur PEUT retourner un des codes d'erreur obligatoires.

Cette opération permet au client de :

* Obtenir une délégation sur tous les types de fichiers sauf les répertoires.
* D'enregistrer un "want" pour une délégation pour l'objet de fichier spécifié, et d'être notifié via un rappel quand la délégation est disponible. Le serveur PEUT prendre en charge les notifications de disponibilité via des rappels. Si le serveur ne prend pas en charge l'enregistrement de "want", il NE DOIT PAS retourner une erreur pour l'indiquer, et DOIT à la place retourner ond_why réglé à WND4_CONTENTION ou WND4_RESOURCE et ond_server_will_push_deleg ou ond_server_will_signal_avail réglé à FAUX. Quand le serveur indique qu'il va notifier au client au moyen d'un rappel, il va fournir la délégation en utilisant une opération CB_PUSH_DELEG ou annuler sa promesse en envoyant une opération CB_WANTS_CANCELLED.
* D'annuler un "want" pour une délégation.

Le client NE DEVRAIT PAS établir OPEN4_SHARE_ACCESS_READ et NE DEVRAIT PAS établir OPEN4_SHARE_ACCESS_WRITE dans wda_want. Si il le fait, le serveur DOIT les ignorer.

La signification des fanions suivants dans wda_want est la même que dans OPEN, sauf comme noté ci-dessous.

* OPEN4_SHARE_ACCESS_WANT_READ_DELEG
* OPEN4_SHARE_ACCESS_WANT_WRITE_DELEG
* OPEN4_SHARE_ACCESS_WANT_ANY_DELEG
* OPEN4_SHARE_ACCESS_WANT_NO_DELEG. À la différence de l'opération OPEN, ce fanion NE DEVRAIT PAS être établi par le client dans les arguments à WANT_DELEGATION, et DOIT être ignoré par le serveur.
* OPEN4_SHARE_ACCESS_WANT_CANCEL
* OPEN4_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_RESRC_AVAIL

* OPEN4_SHARE_ACCESS_WANT_PUSH_DELEG_WHEN_UNCONTENDED

Le traitement des fanions ci-dessus dans WANT_DELEGATION est le même que dans OPEN. Les informations sur la délégation et/ou les promesses que fait le serveur sur de futurs rappels sont les mêmes que celles décrites dans la structure open_delegation4.

Le résultat réussi de WANT_DELEGATION est du type de données open_delegation4, qui est le même type de données que le champ "delegation" dans le résultat de l'opération OPEN (voir le paragraphe 18.16.3). Le serveur construit wdr_resok4 de la même façon qu'il construit la "délégation" de OPEN avec une différence : WANT_DELEGATION NE DOIT PAS retourner un type de délégation de OPEN_DELEGATE_NONE.

Si ((wda_want & OPEN4_SHARE_ACCESS_WANT_DELEG_MASK) & ~OPEN4_SHARE_ACCESS_WANT_NO_DELEG) est zéro, alors le client indique qu'il n'a aucun désir explicite ou aucun désir pour une délégation et le serveur DOIT retourner NFS4ERR_INVAL.

Le client utilise le fanion OPEN4_SHARE_ACCESS_WANT_CANCEL dans l'opération WANT_DELEGATION pour annuler un "want" demandé précédemment pour une délégation. Noter que si le serveur est en cours d'envoi de la délégation (via CB_PUSH_DELEG) au moment où le client envoie une annulation du "want" la délégation pourrait encore être poussée au client.

Si WANT_DELEGATION échoue à retourner une délégation, et si le serveur retourne NFS4_OK, le serveur DOIT régler le type de délégation à OPEN4_DELEGATE_NONE_EXT, et régler od_whynone, comme décrit au paragraphe 18.16. Les délégations d'écriture ne sont pas disponibles pour les types de fichier qui ne sont pas écrivables. Cela inclut les objets de fichier des types NF4BLK, NF4CHR, NF4LNK, NF4SOCK, et NF4FIFO. Si le client demande OPEN4_SHARE_ACCESS_WANT_WRITE_DELEG sans OPEN4_SHARE_ACCESS_WANT_READ_DELEG sur un objet avec un des types de fichier sus-mentionnés, le serveur doit régler wdr_resok4.od_whynone.ond_why à WND4_WRITE_DELEG_NOT_SUPP_FTYPE.

18.49.4 Mise en œuvre

Une demande pour une délégation en conflit n'est normalement pas destinée à déclencher le rappel de la délégation existante. Les serveurs peuvent choisir de traiter certains clients comme ayant une priorité supérieure telle que leur "want" déclenche le rappel d'une délégation existante, bien qu'on supose que cette situation soit inhabituelle.

Les serveurs vont généralement rappeler les délégations allouées par WANT_DELEGATION sur la même base que celles allouées par OPEN. CB_RECALL va généralement être fait seulement quand d'autres clients effectuent des opérations non cohérentes avec la délégation. La réponse normale au vieillissement des délégations est d'utiliser CB_RECALL_ANY, afin de donner au client l'opportunité de garder les délégations les plus utiles à son point de vue.

18.50 Opération 57 : DESTROY_CLIENTID - détruit un identifiant de client

18.50.1 Argument

```
struct DESTROY_CLIENTID4args {
        clientid4       dca_clientid;
};
```

18.50.2 Résultat

```
struct DESTROY_CLIENTID4res {
        nfsstat4        dcr_status;
};
```

18.50.3 Description

L'opération DESTROY_CLIENTID détruit l'identifiant de client. Si il y a des sessions (à la fois inactives et non inactives) des ouvertures, des verrous, des délégations, des dispositions, et/ou des "want" (paragraphe 18.49) associées au prêt non expiré de l'identifiant de client, le serveur DOIT retourner NFS4ERR_CLIENTID_BUSY. DESTROY_CLIENTID PEUT

être précédé d'une opération SEQUENCE pour autant que l'identifiant de client déduit de l'identifiant de session de SEQUENCE ne soit pas le même que l'identifiant de client à détruire. Si les identifiants de client sont les mêmes, alors le serveur DOIT retourner NFS4ERR_CLIENTID_BUSY.

Si DESTROY_CLIENTID n'est pas préfixé de SEQUENCE, il DOIT être la seule opération dans la demande COMPOUND (autrement, le serveur DOIT retourner NFS4ERR_NOT_ONLY_OP). Si l'opération est envoyée sans un SEQUENCE qui la précède, un client qui retransmet la demande peut recevoir une erreur en réponse, parce que la demande d'origine pourrait avoir été exécutée avec succès.

18.50.4 Mise en œuvre

DESTROY_CLIENTID permet à un serveur de réclamer immédiatement les ressources consommées par un identifiant de client inutilisé, et aussi d'oublier qu'il a généré l'identifiant de client. En oubliant qu'il a généré l'identifiant de client, le serveur peut réutiliser en toute sécurité l'identifiant de client sur une future opération EXCHANGE_ID.

18.51 Opération 58 : RECLAIM_COMPLETE - indique la fin de la réclamation

18.51.1 Argument

```
struct RECLAIM_COMPLETE4args {
/* * Si rca_one_fs est VRAI, CURRENT_FH : l'objet dans la réclamation de système de fichiers est terminé. */
      bool          rca_one_fs;
};
```

18.51.2 Résultats

```
struct RECLAIM_COMPLETE4res {
      nfsstat4       rcr_status;
};
```

18.51.3 Description

Une opération RECLAIM_COMPLETE est utilisée pour indiquer que le client a réclamé tout l'état de verrouillage qu'il va récupérer en utilisant la réclamation, quand il récupère l'état dû à un redémarrage de serveur ou à la migration d'un système de fichiers sur un autre serveur. Il y a deux types d'opérations RECLAIM_COMPLETE :

* Quand rca_one_fs est FAUX, un RECLAIM_COMPLETE global est fait. Cela indique que la récupération de tous les verrous que le client détenait sur les précédentes instances de serveur est achevée. La bride de fichier courante n'a pas besoin d'être établie dans ce cas.

* Quand rca_one_fs est VRAI, un RECLAIM_COMPLETE spécifique de système de fichiers est fait. Cela indique que la récupération de verrous pour un seul système de fichiers (celui désigné par la bride de fichier courante) due à la migration du système de fichiers a été achevée. La présence d'une bride de fichier courante est requise quand rca_one_fs est réglé à VRAI. Quand la bride de fichier courante désigne une bride de fichier dans un système de fichiers non en cours de migration, l'opération retourne NFS4_OK et est autrement ignorée.

Une fois qu'un RECLAIM_COMPLETE est fait, il ne peut plus y avoir d'autres opérations de réclamation pour des verrous dont la portée est définie comme ayant complètement récupéré. Une fois que le client envoie RECLAIM_COMPLETE, le serveur ne va pas permettre au client de faire des réclamations suivantes d'état de verrouillage pour cette portée et, si il en est de tentées, il va retourner NFS4ERR_NO_GRACE.

Chaque fois que un client établit un nouvel identifiant de client et avant qu'il fasse la première opération non de réclamation qui obtient un verrou, il DOIT envoyer un RECLAIM_COMPLETE avec rca_one_fs réglé à FAUX, même si il n'y a pas de verrous à réclamer. Si des opérations de verrouillage non de réclamation sont faites avant le RECLAIM_COMPLETE, une erreur NFS4ERR_GRACE va être retournée.

De même, quand le client accède à un système de fichiers qui a migré sur un nouveau serveur, avant qu'il envoie la première opération non de réclamation qui obtient un verrou sur ce nouveau serveur, il DOIT envoyer un

RECLAIM_COMPLETE avec rca_one_fs réglé à VRAI et la bride de fichier courante dans ce système de fichiers, même si il n'y a pas de verrous à réclamer. Si des opérations de verrouillage non de réclamation sont faites sur ce système de fichiers avant le RECLAIM_COMPLETE, une erreur NFS4ERR_GRACE va être retournée.

On devrait noter qu'il y a des situations dans lesquelles un client a besoin de produire les deux formes de RECLAIM_COMPLETE. Un exemple est une instance de migration de système de fichiers dans laquelle le système de fichiers est migré à un serveur pour lequel le client n'a pas d'identifiant de client. Par suite, le client doit obtenir un identifiant de client du serveur (impliquant la responsabilité de faire RECLAIM_COMPLETE avec rca_one_fs réglé à FAUX) ainsi que RECLAIM_COMPLETE avec rca_one_fs réglé à VRAI pour achever la période de grâce par système de fichiers associée à la migration de ce système de fichiers. Les deux peuvent être faits dans n'importe quel ordre tant que toutes les réclamations de verrous nécessaires ont été faites avant de produire l'une d'elles.

Tout verrou non réclamé au moment où RECLAIM_COMPLETE est fait devient non réclamable. Le client NE DOIT PAS tenter de les réclamer, ni durant l'instance courante de serveur, ni dans une instance suivante de serveur, ni sur un autre serveur auquel la responsabilité de ce système de fichiers est transférée. Si le client devait le faire, il violerait le protocole en se représentant lui-même comme possédant des verrous qu'il ne possède pas, et n'a donc aucun droit de réclamer. Voir au paragraphe 8.4.3 de la [RFC5661] la discussion des conditions limites relatives à la réclamation de verrou.

Par l'envoi d'un RECLAIM_COMPLETE, le client indique qu'il est prêt à procéder à des opérations normales de verrouillage non de réclamation. Le client devrait savoir que de telles opérations peuvent résulter temporairement en erreurs NFS4ERR_GRACE jusqu'à ce que le serveur soit prêt à terminer sa période de grâce.

18.51.4 Mise en œuvre

Les serveurs vont normalement utiliser les informations sur quand l'activité de réclamation est achevée pour réduire la longueur de la période de grâce. Quand le serveur maintient dans une mémorisation persistante une liste des clients qui pourraient avoir eu des verrous, il est capable d'utiliser le fait que tous ces clients ont fait un RECLAIM_COMPLETE pour terminer la période de grâce et commencer les opérations normales (c'est-à-dire, accorder des demandes de nouveaux verrous) plus tôt qu'ils ne pourraient autrement.

La latence peut être minimisée en faisant un RECLAIM_COMPLETE au titre de la demande COMPOUND dans laquelle la dernière opération de réclamation de verrou est faite. Quand il n'y a pas de réclamations à faire, RECLAIM_COMPLETE devrait être fait immédiatement afin de permettre que la période de grâce se termine aussitôt que possible.

RECLAIM_COMPLETE devrait seulement être fait une fois pour chaque instance de serveur ou occasion de transition d'un système de fichiers. Si il est fait une seconde fois, l'erreur NFS4ERR_COMPLETE_ALREADY va résulter. Noter que à cause de la protection contre la répétition des caractéristiques de session, le ré-essai de demandes COMPOUND contenant l'opération RECLAIM_COMPLETE n'aura pas cette erreur pour résultat.

Quand un RECLAIM_COMPLETE est envoyé, le client accuse effectivement réception de tous les verrous non encore réclamés comme perdus. Cela permet au serveur de permettre à nouveau au client de récupérer les verrous si des conditions limites apparaissent, comme décrit au paragraphe 8.4.3, qui avaient causé la désactivation par le serveur de la capacité du client à récupérer les verrous.

Parce que les précédentes descriptions de RECLAIM_COMPLETE n'étaient pas suffisamment explicites sur les circonstances dans lesquelles utiliser RECLAIM_COMPLETE avec rca_one_fs réglé à VRAI était approprié, il y a eu des cas où il a été mal utilisé par des clients qui ont produit des RECLAIM_COMPLETE avec rca_one_fs réglé à VRAI quand il n'aurait pas dû l'être. Il y a eu aussi des cas dans lesquels des serveurs n'ont, de diverses façons, pas répondu à une telle mauvaise utilisation comme décrit ci-dessus, soit en ignorant le réglage de rca_one_fs (traitant l'opération comme un RECLAIM_COMPLETE global) ou ignorant l'opération entière.

Alors que les clients NE DEVRAIENT PAS faire un mauvais usage de cette caractéristique, et que les serveurs DEVRAIENT répondre à un tel mauvais usage comme décrit ci-dessus, les mises en œuvre doivent avoir connaissance des considérations suivantes pour faire les compromis nécessaires entre interopérabilité avec les mises en œuvre existantes et la prise en charge appropriée des facilités pour permettre la récupération de verrous en cas de migration de système de fichiers.

* Quand les serveurs ne prennent pas en charge de devenir serveur de destination d'un système de fichiers sujet à migration, il n'y a pas de possibilité qu'un RECLAIM_COMPLETE par système de fichiers soit fait légitimement, et ses

occurrences DEVRAIENT être ignorées. Cependant, les conséquences négatives d'accepter un tel usage fautif sont assez limitées tant que le client ne le fait pas avant que toutes les réclamations nécessaires soient faites.

* Quand un serveur pourrait devenir la destination d'un système de fichiers qui migre, l'utilisation inappropriée de RECLAIM_COMPLETE par système de fichiers est plus problématique. Dans le cas où le système de fichiers désigné n'est pas dans une période de grâce par système de fichiers, le RECLAIM_COMPLETE par système de fichiers DEVRAIT être ignoré, les conséquences négatives de son acceptation étant limitées, comme dans le cas dans lequel la migration n'est pas prise en charge. Cependant, si le serveur rencontre un système de fichiers en train de faire une migration, l'opération ne peut pas être acceptée comme si c'était un RECLAIM_COMPLETE global sans invalider son utilisation prévue.

18.52 Opération 10044 : ILLEGAL - opération illégale

18.52.1 Arguments

vide.

18.52.2 Résultat

```
struct ILLEGAL4res {
        nfsstat4        status;
};
```

18.52.3 Description

Cette opération est un bouche-trou pour coder un résultat pour traiter le cas d'un client qui envoie un code d'opération dans COMPOUND qui n'est pas pris en charge. Voir la description de la procédure COMPOUND pour les détails.

Le champ d'état de ILLEGAL4res DOIT être réglé à NFS4ERR_OP_ILLEGAL.

18.52.4 Mise en œuvre

Un client ne va probablement pas envoyer d'opération avec le code OP_ILLEGAL mais si il le fait, la réponse va être ILLEGAL4res tout comme elle le serait avec tout autre code d'opération invalide. Noter que si le serveur obtient un code d'opération illégal qui n'est pas OP_ILLEGAL, et si le serveur vérifie les codes d'opération légaux durant la phase de décodage XDR, alors ILLEGAL4res ne sera pas retourné.

19. Procédures de rappel NFSv4.1

Les procédures utilisées pour les rappels sont définies dans les paragraphes qui suivent. Dans l'intérêt de la clarté, les termes "client" et "serveur" se réfèrent aux clients et serveurs NFS, en dépit du fait que pour un rappel RPC individuel, le sens de ces termes va être précisément le contraire.

Les deux procédures, CB_NULL et CB_COMPOUND, DOIVENT être mises en œuvre.

19.1 Procédure 0 : CB_NULL - pas d'opération

19.1.1 Arguments

vide.

19.1.2 Résultats

vide

19.1.3 Description

CB_NULL est la procédure ONC RPC NULL standard, avec l'argument vide standard et la réponse vide. Même si il n'y a pas de fonctionnalité directe associée à cette procédure, le serveur va utiliser CB_NULL pour confirmer l'existence d'un chemin pour les RPC du serveur au client.

19.1.4 Erreurs

Aucune.

19.2 Procédure 1 : CB_COMPOUND - opérations composées

19.2.1 Arguments

```
enum nfs_cb_opnum4 {
        OP_CB_GETATTR                   = 3,
        OP_CB_RECALL                    = 4,
/* Opérations de rappel nouvelles pour NFSv4.1 */
        OP_CB_LAYOUTRECALL              = 5,
        OP_CB_NOTIFY                    = 6,
        OP_CB_PUSH_DELEG                = 7,
        OP_CB_RECALL_ANY                = 8,
        OP_CB_RECALLABLE_OBJ_AVAIL = 9,
        OP_CB_RECALL_SLOT               = 10,
        OP_CB_SEQUENCE                  = 11,
        OP_CB_WANTS_CANCELLED           = 12,
        OP_CB_NOTIFY_LOCK               = 13,
        OP_CB_NOTIFY_DEVICEID           = 14,
        OP_CB_ILLEGAL                   = 10044
};

union nfs_cb_argop4 switch (unsigned argop) {
   cas OP_CB_GETATTR : CB_GETATTR4args          opcbgetattr;
   cas OP_CB_RECALL : CB_RECALL4args            opcbrecall;
   cas OP_CB_LAYOUTRECALL : CB_LAYOUTRECALL4args      opcblayoutrecall;
   cas OP_CB_NOTIFY : CB_NOTIFY4args                  opcbnotify;
   cas OP_CB_PUSH_DELEG : CB_PUSH_DELEG4args        opcbpush_deleg;
   cas OP_CB_RECALL_ANY : CB_RECALL_ANY4args       opcbrecall_any;
   cas OP_CB_RECALLABLE_OBJ_AVAIL : CB_RECALLABLE_OBJ_AVAIL4args opcbrecallable_obj_avail;
   cas OP_CB_RECALL_SLOT : CB_RECALL_SLOT4args     opcbrecall_slot;
   cas OP_CB_SEQUENCE : CB_SEQUENCE4args           opcbsequence;
   cas OP_CB_WANTS_CANCELLED : CB_WANTS_CANCELLED4args   opcbwants_cancelled;
   cas OP_CB_NOTIFY_LOCK : CB_NOTIFY_LOCK4args      opcbnotify_lock;
   cas OP_CB_NOTIFY_DEVICEID : CB_NOTIFY_DEVICEID4args   opcbnotify_deviceid;
   cas OP_CB_ILLEGAL :  vide
};

struct CB_COMPOUND4args {
        utf8str_cs          tag;
        uint32_t            minorversion;
        uint32_t            recall_ident;
        nfs_cb_argop4   argarray<>;
};
```

19.2.2 Résultats

```
union nfs_cb_resop4 switch (unsigned resop) {
   cas OP_CB_GETATTR :    CB_GETATTR4res  opcbgetattr;
   cas OP_CB_RECALL :      CB_RECALL4res     opcbrecall;
```

```
/* nouvelles opérations NFSv4.1 */
   cas OP_CB_LAYOUTRECALL :       CB_LAYOUTRECALL4res  opcblayoutrecall;
   cas OP_CB_NOTIFY :             CB_NOTIFY4res        opcbnotify;
   cas OP_CB_PUSH_DELEG :         CB_PUSH_DELEG4res    opcbpush_deleg;
   cas OP_CB_RECALL_ANY :         CB_RECALL_ANY4res    opcbrecall_any;
   cas OP_CB_RECALLABLE_OBJ_AVAIL : CB_RECALLABLE_OBJ_AVAIL4res  opcbrecallable_obj_avail;
   cas OP_CB_RECALL_SLOT :        CB_RECALL_SLOT4res   opcbrecall_slot;
   cas OP_CB_SEQUENCE :           CB_SEQUENCE4res      opcbsequence;
   cas OP_CB_WANTS_CANCELLED : CB_WANTS_CANCELLED4res  opcbwants_cancelled;
   cas OP_CB_NOTIFY_LOCK :        CB_NOTIFY_LOCK4res   opcbnotify_lock;
   cas OP_CB_NOTIFY_DEVICEID :    CB_NOTIFY_DEVICEID4res  opcbnotify_deviceid;

/* opération non nouvelle */
   cas OP_CB_ILLEGAL :    CB_ILLEGAL4res  opcbillegal;
};

struct CB_COMPOUND4res {
    nfsstat4          status;
    utf8str_cs        tag;
    nfs_cb_resop4  resarray<>;
};
```

19.2.3 Description

La procédure CB_COMPOUND est utilisée pour combiner une ou plusieurs procédures de rappel en une seule demande RPC. Le principal programme de rappel RPC a deux procédures principales : CB_NULL et CB_COMPOUND. Toutes les autres opérations utilisent la procédure CB_COMPOUND comme enveloppe.

Durant le traitement de la procédure CB_COMPOUND, le client peut trouver qu'il n'a pas les ressources disponibles pour exécuter certaines ou toutes les opérations de la séquence CB_COMPOUND. Voir les détails au paragraphe 2.10.6.4.

Le champ minorversion des arguments DOIT être le même que le minorversion de la procédure COMPOUND utilisée pour créer l'identifiant de client et la session. Pour NFSv4.1, minorversion DOIT être réglé à 1.

Un champ "status" est contenu dans le résultat de CB_COMPOUND. Cet état DOIT être égal à l'état de la dernière opération exécutée dans la procédure CB_COMPOUND. Donc, si une opération comportait une erreur, alors la valeur de "status" va être la même valeur d'erreur que retournée pour l'opération qui a échoué.

Le champ "tag" est traité de la même façon que celui de la procédure COMPOUND (voir le paragraphe 16.2.3).

Les codes d'opération illégale sont traités de la même façon qu'ils sont traités pour la procédure COMPOUND.

19.2.4 Mise en œuvre

La procédure CB_COMPOUND est utilisée pour combiner des opérations individuelles dans une seule demande RPC. Le client interprète chaque opération à son tour. Si une opération est exécutée par le client et si l'état de cette opération est NFS4_OK, alors la prochaine opération dans la procédure CB_COMPOUND est exécutée. Le client continue ce processus jusqu'à ce qu'il n'y ait plus d'opérations à exécuter ou qu'une des opérations ait une valeur d'état autre que NFS4_OK.

19.2.5 Erreurs

CB_COMPOUND va bien sûr retourner chaque erreur que chaque opération sur le canal de retour peut retourner (voir le Tableau 13). Cependant, si CB_COMPOUND retourne zéro opération, évidemment l'erreur retournée par COMPOUND n'a rien à voir avec une erreur retournée par une opération. La liste des erreurs que CB_COMPOUND va retourner si il traite zéro opération inclut :

Erreur	Notes
NFS4ERR_BADCHAR	L'argument tag a un caractère que le répondeur ne prend pas en charge.

NFS4ERR_BADXDR	
NFS4ERR_DELAY	
NFS4ERR_INVAL	L'argument tag n'est pas codé en UTF-8
NFS4ERR_MINOR_VERS_MISMATCH	
NFS4ERR_SERVERFAULT	
NFS4ERR_TOO_MANY_OPS	
NFS4ERR_REP_TOO_BIG	
NFS4ERR_REP_TOO_BIG_TO_CACHE	
NFS4ERR_REQ_TOO_BIG	

Tableau 24 : Retours d'erreur CB_COMPOUND

20. Opérations de rappel NFSv4.1

20.1 Opération 3 : CB_GETATTR - obtient les attributs

20.1.1 Argument

```
struct CB_GETATTR4args {
        nfs_fh4 fh;
        bitmap4 attr_request;
};
```

20.1.2 Résultat

```
struct CB_GETATTR4resok {
        fattr4  obj_attributes;
};

union CB_GETATTR4res switch (nfsstat4 status) {
   cas NFS4_OK : CB_GETATTR4resok       resok4;
   défaut : vide
};
```

20.1.3 Description

L'opération CB_GETATTR est utilisée par le serveur pour obtenir l'état courant modifié d'un fichier qui a été délégué OPEN_DELEGATE_WRITE. Les attributs de taille et de changement sont les seuls dont il soit garanti qu'il vont être servis par le client. Voir au paragraphe 10.4.3 une pleine description de comment le client et le serveur vont interagir avec l'utilisation de CB_GETATTR.

Si la bride de fichier spécifiée n'est pas une pour laquelle le client détient une délégation OPEN_DELEGATE_WRITE, une erreur NFS4ERR_BADHANDLE est retournée.

20.1.4 Mise en œuvre

Le client retourne les bits de attrmask et les valeurs d'attribut associées seulement pour l'attribut de changement, et les attributs qu'il peut changer (time_modify, et size).

20.2 Opération 4 : CB_RECALL - rappelle une délégation

20.2.1 Argument

```
struct CB_RECALL4args {
      stateid4      stateid;
      bool          truncate;
      nfs_fh4       fh;
```

```
};
```

20.2.2 Résultat

```
struct CB_RECALL4res {
    nfsstat4    status;
};
```

20.2.3 Description

L'opération CB_RECALL est utilisée pour commencer le processus de rappel d'une délégation et la retourner au serveur.

Le fanion truncate est utilisé pour optimiser le rappel pour un objet de fichier qui est un fichier régulier et est sur le point d'être tronqué à zéro. Quand il est VRAI, le client est libéré de l'obligation de propager au serveur les données modifiées pour le fichier, parce que ces données ne sont pas pertinentes.

Si la bride spécifié n'est pas une pour qui le client détient une délégation, une erreur NFS4ERR_BADHANDLE est retournée.

Si l'identifiant d'état spécifié n'est pas un correspondant à une délégation OPEN pour le fichier spécifié par la bride de fichier, une erreur NFS4ERR_BAD_STATEID est retournée.

20.2.4 Mise en œuvre

Le client DEVRAIT répondre au rappel immédiatement. Répondre ne termine pas le rappel sauf quand la valeur du champ État de la réponse n'est ni NFS4ERR_DELAY ni NFS4_OK. Le rappel n'est pas achevé tant que la délégation n'est pas retournée en utilisant une opération DELEGRETURN.

20.3 Opération 5 : CB_LAYOUTRECALL - rappel de disposition du client

20.3.1 Argument

```
/* * Arguments et résultats de rappel NFSv4.1 */

enum layoutrecall_type4 {
    LAYOUTRECALL4_FILE = LAYOUT4_RET_REC_FILE,
    LAYOUTRECALL4_FSID = LAYOUT4_RET_REC_FSID,
    LAYOUTRECALL4_ALL  = LAYOUT4_RET_REC_ALL
};

struct layoutrecall_file4 {
    nfs_fh4     lor_fh;
    offset4     lor_offset;
    length4     lor_length;
    stateid4    lor_stateid;
};

union layoutrecall4 switch(layoutrecall_type4 lor_recalltype) {
  cas LAYOUTRECALL4_FILE : layoutrecall_file4  lor_layout;
  cas LAYOUTRECALL4_FSID : fsid4               lor_fsid;
  cas LAYOUTRECALL4_ALL : vide
};

struct CB_LAYOUTRECALL4args {
        layouttype4         clora_type;
        layoutiomode4       clora_iomode;
        bool                clora_changed;
        layoutrecall4       clora_recall;
```

```
};
```

20.3.2 Résultat

```
struct CB_LAYOUTRECALL4res {
        nfsstat4        clorr_status;
};
```

20.3.3 Description

L'opération CB_LAYOUTRECALL est utilisée par le serveur pour rappeler les dispositions du client ; par suite, le client va commencer le processus de retour des dispositions via LAYOUTRETURN. L'opération CB_LAYOUTRECALL spécifie une des trois formes de traitement de rappel avec la valeur de layoutrecall_type4. Le rappel est pour une des suivantes : une disposition spécifique d'un fichier spécifique (LAYOUTRECALL4_FILE) un identifiant de système de fichiers entier (LAYOUTRECALL4_FSID) ou tous les systèmes de fichiers (LAYOUTRECALL4_ALL).

Le comportement de l'opération varie sur la base de la valeur de layoutrecall_type4. La valeur et les comportements sont :

LAYOUTRECALL4_FILE : pour qu'une disposition corresponde à la demande de rappel, les valeurs des champs suivants doivent correspondre à celles de la disposition : clora_type, clora_iomode, lor_fh, et la gamme d'octets spécifiée par lor_offset et lor_length. Le champ clora_iomode peut avoir la valeur spéciale de LAYOUTIOMODE4_ANY. La valeur spéciale LAYOUTIOMODE4_ANY va correspondre à tout iomode originellement retourné dans une disposition ; donc, elle agit comme un caractère générique. L'autre valeur spéciale utilisée est pour lor_length. Si lor_length a une valeur de NFS4_UINT64_MAX, le champ lor_length signifie la taille maximum possible de fichier. Si une disposition correspondante est trouvée, elle DOIT être retournée en utilisant l'opération LAYOUTRETURN (voir le paragraphe 18.44). Un exemple d'utilisation de la valeur spéciale du champ est si clora_iomode est LAYOUTIOMODE4_ANY, lor_offset est zéro, et lor_length est NFS4_UINT64_MAX, alors la disposition entière est à retourner.

L'erreur NFS4ERR_NOMATCHING_LAYOUT est seulement retournée quand le client ne détient pas de dispositions pour le fichier ou si le client n'a aucune disposition en chevauchement pour la spécification dans le rappel de la disposition.

LAYOUTRECALL4_FSID et LAYOUTRECALL4_ALL : si LAYOUTRECALL4_FSID est spécifié, l'identifiant de système de fichiers *(fsid)* spécifie le système de fichiers pour lequel toutes les dispositions en instance DOIVENT être retournées. Si LAYOUTRECALL4_ALL est spécifié, toutes les dispositions en instance DOIVENT être retournées. De plus, LAYOUTRECALL4_FSID et LAYOUTRECALL4_ALL spécifient que toutes les transpositions d'identifiant d'appareil de mémorisation en adresse d'appareil de mémorisation dans le ou les systèmes de fichiers affectés sont aussi rappelées. Le LAYOUTRETURN respectif avec LAYOUTRETURN4_FSID ou LAYOUTRETURN4_ALL accuse réception au serveur que le client a invalidé lesdites transpositions d'appareils. Voir au paragraphe 12.5.5.2.1.5 sur les considérations sur le rappel de dispositions "en vrac".

L'erreur NFS4ERR_NOMATCHING_LAYOUT est seulement retournée quand le client ne détient pas de disposition et n'a pas de transposition valide d'identifiant d'appareil.

Dans le traitement de la demande de rappel de disposition, le client change aussi son comportement sur la base de la valeur du champ clora_changed. Ce champ est utilisé par le serveur pour fournir du contexte supplémentaire sur la raison pour laquelle la disposition est rappelée. Une valeur de FAUX pour clora_changed indique qu'aucun changement dans la disposition n'est attendu et que le client peut écrire des données modifiées sur les appareils de mémorisation impliqués ; cela doit être fait avant de retourner la disposition via LAYOUTRETURN. Une valeur de VRAI pour clora_changed indique que le serveur change la disposition. Des exemples de changements de disposition et des raisons pour une indication de VRAI sont les suivants : le serveur de métadonnées redessine les bandes du fichier ou une erreur permanente s'est produite sur un appareil de mémorisation et le serveur de métadonnées aimerait fournir une nouvelle disposition pour le fichier. Donc, une valeur de clora_changed de VRAI indique un certain niveau de changement pour la disposition et le client NE DEVRAIT PAS écrire et engager des données modifiées sur les appareils de mémorisation. Dans ce cas, le client écrit et engage les données à travers le serveur de métadonnées.

Voir au paragraphe 12.5.3 la description de comment le champ lor_stateid dans les arguments est construit. Noter que le champ "seqid" de lor_stateid NE DOIT PAS être zéro. Voir aux paragraphes 8.2, 12.5.3, et 12.5.5.2 la discussion et les exigences.

20.3.4 Mise en œuvre

Le traitement du client pour CB_LAYOUTRECALL est similaire à celui de CB_RECALL (rappel de délégations de fichier) en ce que le client répond à la demande avant de réellement retourner les dispositions via l'opération LAYOUTRETURN. Alors que le client répond immédiatement au CB_LAYOUTRECALL, l'opération n'est pas considérée achevée (c'est-à-dire, elle est considérée comme en cours) tant que toutes les dispositions affectées ne sont pas retournées au serveur via l'opération LAYOUTRETURN.

Avant de retourner la disposition au serveur via LAYOUTRETURN, le client devrait attendre la réponse aux opérations READ, WRITE, ou COMMIT en cours ou en instance qui utilisent la disposition rappelée.

Si le client détient des données modifiées qui sont affectées par une disposition rappelée, le client a diverses options pour écrire les données sur le serveur. Comme toujours, le client peut écrire les données à travers le serveur de métadonnées. En fait, le client peut ne pas avoir de choix autre que d'écrire sur le serveur de métadonnées quand l'argument clora_changed est VRAI et qu'une nouvelle disposition est indisponible au serveur. Cependant, le client peut être capable d'écrire les données modifiées sur l'appareil de mémorisation si l'argument clora_changed est FAUX ; cela doit être fait avant de retourner la disposition via LAYOUTRETURN. Si le client devait obtenir une nouvelle disposition couvrant la gamme d'octets des données modifiées, alors écrire sur les appareils de mémorisation est une solution de remplacement disponible. Noter qu'avant d'obtenir une nouvelle disposition, le client doit d'abord retourner la disposition originale.

Dans le cas de données modifiées écrites alors que la disposition est détenue, le client doit utiliser les opérations LAYOUTCOMMIT au moment approprié ; une LAYOUTCOMMIT exigée doit être faite avant LAYOUTRETURN. Si une grande quantité de données modifiées est en instance, le client peut envoyer des LAYOUTRETURN pour des portions de la disposition rappelée ; cela permet au serveur de surveiller les progrès du client et le respect de la demande de rappel originale. Cependant, le dernier LAYOUTRETURN dans une séquence de retours DOIT spécifier la gamme complète rappelée (voir les détails au paragraphe 12.5.5.1).

Si un serveur a besoin de supprimer un identifiant d'appareil et si il y a des dispositions qui se réfèrent à l'identifiant d'appareil, CB_LAYOUTRECALL DOIT être invoqué pour causer le retour par le client de toutes les dispositions se référant à l'identifiant d'appareil avant que le serveur puisse supprimer l'identifiant d'appareil. Si le client ne retourne pas les dispositions affectées, le serveur PEUT révoquer les dispositions.

20.4 Opération 6 : CB_NOTIFY - notifie au client les changements de répertoire

20.4.1 Argument

```
/* * Types de notification de répertoire. */
enum notify_type4 {
        NOTIFY4_CHANGE_CHILD_ATTRS      = 0,
        NOTIFY4_CHANGE_DIR_ATTRS        = 1,
        NOTIFY4_REMOVE_ENTRY            = 2,
        NOTIFY4_ADD_ENTRY               = 3,
        NOTIFY4_RENAME_ENTRY            = 4,
        NOTIFY4_CHANGE_COOKIE_VERIFIER = 5
};

/* Informations d'entrée changées.  */
struct notify_entry4 {
        component4      ne_file;
        fattr4          ne_attrs;
};

/* Informations d'entrée précédentes. */
struct prev_entry4 {
        notify_entry4   pe_prev_entry;                  /* ce que READDIR a retourné pour cette entrée */
        nfs_cookie4     pe_prev_entry_cookie;
};
```

```
struct notify_remove4 {
        notify_entry4   nrm_old_entry;
        nfs_cookie4     nrm_old_entry_cookie;
};

struct notify_add4 {
/* * Informations sur un objet éventuellement renommé. */
        notify_remove4    nad_old_entry<1>;
        notify_entry4     nad_new_entry;                    /* ce que READDIR a retourné pour cette entrée */
        nfs_cookie4       nad_new_entry_cookie<1>;
        prev_entry4       nad_prev_entry<1>;
        bool              nad_last_entry;
};

struct notify_attr4 {
        notify_entry4     na_changed_entry;
};

struct notify_rename4 {
        notify_remove4    nrn_old_entry;
        notify_add4       nrn_new_entry;
};

struct notify_verifier4 {
          verifier4       nv_old_cookieverf;
          verifier4       nv_new_cookieverf;
};

/* * Les objets de type notify_<>4 et notify_device_<>4 sont codés dans cela. */
  typedef opaque notifylist4<>;

struct notify4 {                                    /* composé à partir de notify_type4 ou notify_deviceid_type4 */
          bitmap4         notify_mask;
          notifylist4     notify_vals;
};

struct CB_NOTIFY4args {
          stateid4     cna_stateid;
          nfs_fh4      cna_fh;
          notify4      cna_changes<>;
};
```

20.4.2 Résultat

```
struct CB_NOTIFY4res {
        nfsstat4        cnr_status;
};
```

20.4.3 Description

L'opération CB_NOTIFY est utilisée par le serveur pour envoyer des notifications aux clients sur les changements aux répertoires délégués. L'enregistrement de notifications pour les répertoires se produit quand la délégation est établie en utilisant GET_DIR_DELEGATION. Ces notifications sont envoyées sur le canal de retour. La notification est envoyée une fois la demande d'origine traitée sur le serveur. Le serveur va envoyer un dispositif de notifications pour les changements qui pourraient s'être produits dans le répertoire. Les notifications sont envoyées comme des listes de paires de gabarits binaires et de valeurs. Voir au paragraphe 3.3.7 la description de comment fonctionnent les gabarits binaires NFSv4.1.

Si le serveur a plus de notifications qu'il n'en peut tenir dans la demande CB_COMPOUND, il DEVRAIT envoyer une séquence de demandes CB_COMPOUND afin que la vue du client du répertoire ne soit pas confuse. Par exemple, si le

serveur indique qu'un fichier nommé "foo" est ajouté et que le fichier "foo" est supprimé, l'ordre dans lequel le client reçoit ces notifications doit être le même que l'ordre dans lequel les opérations correspondantes se sont produites sur le serveur.

Si le client détenant la délégation fait dans le répertoire des changements qui causent l'ajout ou la suppression de fichiers ou sous répertoires, le serveur va notifier le ou les changements résultants au client. Si le client qui détient la délégation fait seulement des changements d'attribut ou de vérificateur de mouchard, le serveur n'envoie pas de notification à ce client. Le serveur va envoyer les informations suivantes pour chaque opération :

NOTIFY4_ADD_ENTRY : le serveur va envoyer des informations sur la nouvelle entrée de répertoire créée avec le mouchard pour cette entrée. Les informations d'entrée (type de données notify_add4) incluent le nom de composant de l'entrée et des attributs. Le serveur va envoyer ce type d'entrée quand un fichier est créé, quand une entrée est ajoutée à un répertoire par suite d'un changement de nom à travers les répertoires (voir ci-dessous) et quand une liaison physique est créée avec un fichier existant. Si cette entrée est ajoutée à la fin du répertoire, le serveur va établir le fanion nad_last_entry à VRAI. Si le fichier est ajouté de telle façon qu'il y ait au moins une entrée avant lui, le serveur va aussi retourner les précédentes informations d'entrée (nad_prev_entry, un dispositif de longueur variable de jusqu'à un élément. Si le dipositif est de longueur zéro, il n'y a pas d'entrée précédente) avec son mouchard. C'est pour aider le client à trouver la bonne localisation dans son antémémoire de noms de fichiers et antémémoires de répertoires où cette entrée devrait être mise en antémémoire. Si le mouchard de la nouvelle entrée est disponible, il va être dans le champ nad_new_entry_cookie (un autre dispositif de longueur variable de jusqu'à un élément). Si l'ajout de l'entrée cause la suppression d'une autre entrée (ce qui ne peut arriver que dans le cas de changement de nom) atomique avec l'ajout, alors les informations sur cette entrée sont rapportées dans nad_old_entry.

NOTIFY4_REMOVE_ENTRY : le serveur va envoyer des informations sur l'entrée de répertoire supprimée. Le serveur va aussi envoyer la valeur du mouchard pour l'entrée supprimée afin que les clients puissent aller aux informations en antémémoire pour cette entrée.

NOTIFY4_RENAME_ENTRY : le serveur va envoyer les informations sur les deux entrées ancienne et nouvelle. Cela inclut le nom et les attributs pour chaque entrée. De plus, si le changement de nom cause la suppression d'une entrée (c'est-à-dire, dans le cas du changement de nom d'un fichier) alors ceci est rapporté dans nrn_new_new_entry.nad_old_entry. Cette notification n'est envoyée que si les deux entrées sont dans le même répertoire. Si le changement de nom est à travers des répertoires, le serveur va envoyer une notification de suppression à un répertoire et une notification d'ajout à l'autre répertoire, en supposant que les deux ont une délégation de répertoire.

NOTIFY4_CHANGE_CHILD_ATTRS/NOTIFY4_CHANGE_DIR_ATTRS : le client va utiliser le gabarit d'attribut pour informer le serveur des attributs pour lesquels il veut recevoir des notifications. Cette notification de changement peut être demandée pour des changements aux attributs de répertoire aussi bien que des changements à tout attribut du fichier dans le répertoire en utilisant deux gabarits d'attribut séparés. Le client ne peut pas demander de notification de changement d'attribut pour un fichier spécifique. Un gabarit d'attribut couvre tous les fichiers dans le répertoire. Lors d'un changement d'attribut, le serveur va renvoyer les valeurs des attributs changés. Les notifications pourrait ne pas avoir de sens pour certains attributs au niveau du système de fichiers, et il appartient au serveur de décider quel sous ensemble il veut prendre en charge. Le client peut négocier la fréquence de notifications d'attribut en faisant savoir au serveur à quelle fréquence il veut avoir notification d'un changement d'attribut. Le serveur va retourner les fréquences de notification prises en charge ou une indication qu'aucune notification n'est permise pour les attributs de répertoire ou d'enfants en établissant respectivement les attributs dir_notif_delay et dir_entry_notif_delay.

NOTIFY4_CHANGE_COOKIE_VERIFIER : si le vérificateur de mouchard change alors qu'un client détient une délégation, le serveur va le notifier au client afin qu'il puisse invalider ses mouchards et envoyer à nouveau un READDIR pour obtenir le nouvel ensemble de mouchards.

20.5 Opération 7: CB_PUSH_DELEG - offre au client une délégation précédemment demandée

20.5.1 Argument

```
struct CB_PUSH_DELEG4args {
      nfs_fh4             cpda_fh;
      open_delegation     cpda_delegation;
};
```

20.5.2 Résultat

```
struct CB_PUSH_DELEG4res {
        nfsstat4 cpdr_status;
};
```

20.5.3 Description

CB_PUSH_DELEG est utilisé par le serveur à la fois pour signaler au client que la délégation qu'il veut (précédemment indiquée via un "want" établi à partir d'une opération OPEN ou WANT_DELEGATION) est disponible et offrir simultanément la délégation au client. Le client a le choix d'accepter la délégation en retournant NFS4_OK au serveur, de retarder la décision d'accepter la délégation offerte en retournant NFS4ERR_DELAY, ou d'un rejet permanent de l'offre de délégation en retournant NFS4ERR_REJECT_DELEG. Quand une délégation est rejetée de cette façon, le "want" précédemment établi est supprimé de façon permanente et la délégation est sujette à acquisition par un autre client.

20.5.4 Mise en œuvre

Si le client retourne NFS4ERR_DELAY et si il y a une demande de délégation en conflit, le serveur PEUT la traiter aux dépens du client qui a retourné NFS4ERR_DELAY. Le "want" du client ne va pas être annulé, mais PEUT être traité après d'autres demandes de délégation ou "want" enregistrés.

Quand un client retourne un état autre que NFS4_OK, NFS4ERR_DELAY, ou NFS4ERR_REJECT_DELAY, le "want" reste en instance, bien que les serveurs puissent décider de l'annuler en envoyant un CB_WANTS_CANCELLED.

20.6 Opération 8 : CB_RECALL_ANY - garde tous les objets rappelables

20.6.1 Argument

```
const RCA4_TYPE_MASK_RDATA_DLG          = 0;
const RCA4_TYPE_MASK_WDATA_DLG          = 1;
const RCA4_TYPE_MASK_DIR_DLG            = 2;
const RCA4_TYPE_MASK_FILE_LAYOUT        = 3;
const RCA4_TYPE_MASK_BLK_LAYOUT         = 4;
const RCA4_TYPE_MASK_OBJ_LAYOUT_MIN     = 8;
const RCA4_TYPE_MASK_OBJ_LAYOUT_MAX     = 9;
const RCA4_TYPE_MASK_OTHER_LAYOUT_MIN   = 12;
const RCA4_TYPE_MASK_OTHER_LAYOUT_MAX   = 15;

struct  CB_RECALL_ANY4args      {
      uint32_t      craa_objects_to_keep;
      bitmap4       craa_type_mask;
};
```

20.6.2 Résultat

```
struct CB_RECALL_ANY4res {
      nfsstat4      crar_status;
};
```

20.6.3 Description

Le serveur peut décider qu'il ne peut pas détenir tout l'état pour les objets rappelables, comme les délégations et dispositions, sans épuiser ses ressources. Dans ce cas, bien que non optimal, le serveur est libre de rappeler les objets individuels pour réduire la charge.

Parce que l'objet général d'objets rappelables comme les délégations est d'éliminer l'interaction du client avec le serveur, le serveur ne peut pas interpréter le manque d'utilisation récente comme l'indication que l'objet n'est plus utile. L'absence d'utilisation visible est cohérente avec une délégation empêchant des opérations potentielles d'être envoyées au serveur.

Dans le cas de dispositions, bien qu'il soit vrai que l'utilité d'une disposition est indiquée par l'utilisation de la disposition quand les appareils de mémorisation reçoivent les demandes d'entrée/sortie, parce que il n'est pas obligatoire qu'un appareil de mémorisation indique au serveur de métadonnées une utilisation passée ou présente d'une disposition, le serveur de métadonnées ne va probablement pas savoir quelles dispositions sont de bonnes candidates au rappel en réponse à de faibles ressources.

Afin de mettre en œuvre un schéma efficace de réclamation pour de tels objets, la connaissance par le serveur des ressources disponibles doit être utilisée pour déterminer quand des objets doivent être rappelés, les clients choisissant les objets réels à retourner.

Les mises en œuvre de serveur peuvent différer dans leurs exigences d'allocation de ressources. Par exemple, un serveur peut partager les ressources entre toutes les classes d'objets rappelables, tandis qu'un autre peut utiliser des réservoirs de ressources séparés pour les dispositions et les délégations, ou encore séparer les ressources par types de délégations.

Quand un réservoir de ressource donné est sur-utilisé, le serveur peut envoyer un CB_RECALL_ANY aux clients détenant des objets rappelables des types impliqués, leur permettant de conserver un certain nombre de ces objets et de retourner ceux en excédent. Un gabarit spécifie quels types d'objets sont à limiter. Le client choisit, sur la base de sa propre connaissance de l'utilité actuelle, les objets de cette classe qui devraient être retournés.

Un certain nombre de bits sont définis. Pour certains d'entre eux, des gammes sont définies et il revient à la définition du protocole de mémorisation de spécifier comment elles sont utilisées. Ce sont des gammes réservées pour les protocoles de mémorisation fondés sur l'objet et pour d'autres protocoles de mémorisation expérimentaux. Une RFC définissant un tel protocole de mémorisation devra spécifier comment des bits particuliers dans ces gammes sont utilisés. Par exemple, elle peut spécifier une transposition entre attributs de la disposition (lecture ou écriture, taille de zone) et le bit à utiliser, ou elle peut définir un champ dans la disposition où la position de bit associée est rendue disponible par le serveur au client.

RCA4_TYPE_MASK_RDATA_DLG : le client va retourner des délégations OPEN_DELEGATE_READ sur des objets de fichier non répertoire.

RCA4_TYPE_MASK_WDATA_DLG : le client va retourner des délégations OPEN_DELEGATE_WRITE sur des objets de fichier régulier.

RCA4_TYPE_MASK_DIR_DLG : le client va retourner des délégations de répertoire.

RCA4_TYPE_MASK_FILE_LAYOUT : le client va retourner des dispositions de type LAYOUT4_NFSV4_1_FILES.

RCA4_TYPE_MASK_BLK_LAYOUT : voir la description dans la [RFC5663].

RCA4_TYPE_MASK_OBJ_LAYOUT_MIN à RCA4_TYPE_MASK_OBJ_LAYOUT_MAX : voir la description dans la [RFC5664].

RCA4_TYPE_MASK_OTHER_LAYOUT_MIN à RCA4_TYPE_MASK_OTHER_LAYOUT_MAX : cette gamme est réservée pour dire au client de rappeler des dispositions de type expérimental ou spécifique de site (voir le paragraphe 3.3.13).

Quand un bit est établi dans le gabarit de type qui correspond à un type indéfini d'objet rappelable, NFS4ERR_INVAL DOIT être retourné. Quand un bit est établi qui correspond à un type défini d'objet mais que le client ne prend pas en charge, NFS4ERR_INVAL NE DOIT PAS être retourné. De futures versions mineures de NFSv4 pourront étendre l'ensemble des bits de gabarit de type valides.

CB_RECALL_ANY spécifie un compte d'objets que le client peut conserver par opposition à un compte que le client doit retourner. C'est pour éviter une concurrence potentielle entre un CB_RECALL_ANY qui avait un compte d'objets à libérer avec un ensemble d'opérations générées par le client pour retourner des dispositions ou délégations. Par suite de cette concurrence, le client et le serveur auraient des idées différentes sur le nombre d'objets à retourner. Donc, le client pourrait à tort en libérer trop.

Si les demandes de ressource l'y invitent, le serveur peut envoyer un autre CB_RECALL_ANY avec un compte plus faible, même si il n'a pas encore reçu un accusé de réception du client pour un CB_RECALL_ANY précédent avec le même gabarit de type. Bien que la possibilité existe qu'ils aient été reçus par le client dans un ordre différent de celui de leur envoi, une telle permutation du flux de rappel est sans dommage. C'est la tâche du client de ramener la taille de l'ensemble d'objets rappelables en ligne avec chaque CB_RECALL_ANY reçu, et jusqu'à ce que cette obligation soit satisfaite, elle ne

peut pas être annulée ou modifiée par un CB_RECALL_ANY suivant pour le même gabarit de type. Donc, si le serveur envoie deux CB_RECALL_ANY, l'effet sera le même que si le compte inférieur avait été envoyé, quel que soit l'ordre de réception du rappel. Noter que cela signifie que un serveur ne peut pas annuler l'effet d'un CB_RECALL_ANY par l'envoi d'un autre rappel avec un compte plus élevé. Quand un CB_RECALL_ANY est reçu et que le compte est déjà dans l'ensemble limite ou est au dessus d'une limite que le client essaye de réduire, ce rappel n'a pas d'effet.

Les serveurs sont généralement libres de refuser des objets rappelables quand des ressources insuffisantes sont disponibles. Noter que l'effet d'une telle politique est implicitement de donner la préséance aux objets existants par rapport à ceux demandés, avec le résultat que les ressources pourraient n'être pas utilisées de façon optimale. Pour empêcher cela, les serveurs sont bien avisés de régler le point auquel ils commencent d'envoyer des rappels CB_RECALL_ANY un peu en dessous de celui auquel ils cessent d'abandonner les nouvelles délégations et dispositions. Cela permet au client de purger ses objets moins utilisés chaque fois qu'approprié et donc de continuer de voir ses demandes suivantes recevoir de nouvelles ressources libérées par l'objet retourné.

20.6.4 Mise en œuvre

Le client peut choisir de retourner tout type d'objet spécifié par le gabarit. Si un serveur souhaite limiter l'utilisation des objets d'un type spécifique, il devrait seulement spécifier ce type dans le gabarit qu'il envoie. Si le client devait échouer à retourner les objets demandés, il appartient au serveur de traiter cette situation, normalement par l'envoi de rappels spécifiques (c'est-à-dire, l'envoi d'opérations CB_RECALL) pour limiter de façon appropriée l'usage des ressources. Le serveur devrait donner au client assez de temps pour retourner les objets avant de procéder à des rappels spécifiques. Ce temps ne devrait pas être inférieur à la période de prêt.

20.7 Opération 9 : CB_RECALLABLE_OBJ_AVAIL - signale des ressources pour des objets rappelables

20.7.1 Argument

```
typedef CB_RECALL_ANY4args CB_RECALLABLE_OBJ_AVAIL4args;
```

20.7.2 Résultat

```
struct CB_RECALLABLE_OBJ_AVAIL4res {
        nfsstat4        croa_status;
};
```

20.7.3 Description

CB_RECALLABLE_OBJ_AVAIL est utilisé par le serveur pour signaler au client que le serveur a des ressources pour accorder des objets rappelables qui pourraient avoir été précédemment refusés par OPEN, WANT_DELEGATION, GET_DIR_DELEG, ou LAYOUTGET.

L'argument craa_objects_to_keep signifie le nombre total d'objets rappelables des types indiqués dans l'argument type_mask que le serveur estime qu'il peut permettre au client d'avoir, y compris le nombre de tels objets que le client a déjà. Un client qui essaye d'acquérir plus d'objets rappelables que ce que le serveur l'informe qu'il peut avoir court le risque d'avoir des objets rappelés.

Le serveur n'est pas obligé de réserver la différence entre le nombre d'objets qu'a actuellement le client et la valeur de craa_objects_to_keep, ni retarder la réponse à CB_RECALLABLE_OBJ_AVAIL n'empêche le serveur d'utiliser les ressources des objets rappelables pour un autre objet. Bien sûr, si un client répond lentement à CB_RECALLABLE_OBJ_AVAIL, le serveur pourrait interpréter que le client a réduit la capacité de gérer les objets rappelables, et donc annule ou réduit toute réservation qu'il maintenait au nom du client. Donc, si le client désire acquérir plus d'objets rappelables, il doit répondre rapidement à CB_RECALLABLE_OBJ_AVAIL, et alors envoyer les opérations appropriées pour acquérir des objets rappelables.

20.8 Opération 10 : CB_RECALL_SLOT - change les limites de contrôle de flux

20.8.1 Argument

```
struct CB_RECALL_SLOT4args {
        slotid4      rsa_target_highest_slotid;
};
```

20.8.2 Résultat

```
struct CB_RECALL_SLOT4res {
        nfsstat4     rsr_status;
};
```

20.8.3 Description

L'opération CB_RECALL_SLOT demande au client de retourner des créneaux de session, et si applicable, des crédits de transport (par exemple, des crédits RDMA pour les connexions associées au canal des opérations) du canal avant de la session. CB_RECALL_SLOT spécifie rsa_target_highest_slotid, la valeur du plus fort identifiant de créneau de la cible que le serveur veut pour la session. Le client DOIT alors progresser vers la réduction du plus fort identifiant de créneau de la session à la valeur cible.

Si la session a seulement des connexions non RDMA associées à son canal d'opérations, alors le client doit seulement attendre que s'achèvent toutes les demandes en instance avec un identifiant de créneau supérieur à rsa_target_highest_slotid, en voyant alors un seul COMPOUND consistant en une seule opération SEQUENCE, avec le champ sa_highestslot réglé à rsa_target_highest_slotid. Si il y a des connexions fondées sur RDMA associées au canal d'opération, alors le client doit aussi envoyer assez de messages "RDMA Send" de longueur zéro pour faire du compte total de crédits RDMA rsa_target_highest_slotid + 1 ou moins.

20.8.4 Mise en œuvre

Si le client échoue à réduire le plus fort créneau qu'il avait sur le canal avant à ce que le serveur demande, le serveur peut forcer la question en affirmant le contrôle de flux sur le côté récepteur de toutes les connexions liées au canal avant, et alors finir de servir toutes les demandes en instance qui sont dans des créneaux supérieurs à rsa_target_highest_slotid. Une fois que cela est fait, le serveur peut alors ouvrir le contrôle de flux, et chaque fois que le client envoie une nouvelle demande sur un créneau supérieur à rsa_target_highest_slotid, le serveur peut retourner NFS4ERR_BADSLOT.

20.9 Opération 11 : CB_SEQUENCE - fournit le séquençage et le contrôle du canal de retour

20.9.1 Argument

```
struct referring_call4 {
      sequenceid4     rc_sequenceid;
      slotid4         rc_slotid;
};

struct referring_call_list4 {
      sessionid4        rcl_sessionid;
      referring_call4   rcl_referring_calls<>;
};

struct CB_SEQUENCE4args {
      sessionid4            csa_sessionid;
      sequenceid4           csa_sequenceid;
      slotid4               csa_slotid;
      slotid4               csa_highest_slotid;
      bool                  csa_cachethis;
      referring_call_list4  csa_referring_call_lists<>;
};
```

20.9.2 Résultat

```
struct CB_SEQUENCE4resok {
      sessionid4      csr_sessionid;
      sequenceid4     csr_sequenceid;
      slotid4         csr_slotid;
      slotid4         csr_highest_slotid;
      slotid4         csr_target_highest_slotid;
};

union CB_SEQUENCE4res switch (nfsstat4 csr_status) {
  cas NFS4_OK : CB_SEQUENCE4resok   csr_resok4;
  défaut : vide
};
```

20.9.3 Description

L'opération CB_SEQUENCE est utilisée pour gérer la comptabilité opérationnelle pour le canal de retour de la session sur laquelle une demande est envoyée. Le contenu inclut l'identifiant de session auquel cette demande appartient, l'identifiant de créneau et l'identifiant de séquence utilisés par le serveur pour mettre en œuvre le contrôle de demande de session et la sémantique de exactement une fois, et les maxima d'identifiant de créneau échangés qui sont utilisés pour ajuster la taille de la mémoire tampon de réponse. Dans chaque demande CB_COMPOUND, CB_SEQUENCE DOIT apparaître une fois et DOIT être la première opération. L'erreur NFS4ERR_SEQUENCE_POS DOIT être retournée quand CB_SEQUENCE est trouvé dans toute position au delà de la première dans un CB_COMPOUND. Si une autre opération est dans la première position de CB_COMPOUND, NFS4ERR_OP_NOT_IN_SESSION DOIT être retourné.

Voir au paragraphe 18.46.3 la description du traitement des créneaux.

Si csa_cachethis est VRAI, alors le serveur demande que le client mette en antémémoire la réponse dans la mémoire tampon de réponse de rappel. Le client DOIT mettre la réponse en antémémoire (voir le paragraphe 2.10.6.1.3).

Le dispositif csa_referring_call_lists est la liste des demandes COMPOUND, identifiées par identifiant de session, identifiant de créneau, et identifiant de séquence. Ce sont les demandes que le client a précédemment envoyées au serveur. Ces demandes précédentes ont créé l'état que certaines opérations dans le même CB_COMPOUND que la csa_referring_call_lists identifie. Un identifiant de session est inclus parce que l'état de prêt est lié à un identifiant de client, et un identifiant de client peut avoir plusieurs sessions. Voir au paragraphe 2.10.6.3.

La valeur de l'argument csa_sequenceid relatif à l'identifiant de séquence mis en antémémoire sur le créneau entre dans un des trois cas suivants :

* Si la différence entre csa_sequenceid et l'identifiant de séquence mis en antémémoire du client à l'identifiant de créneau est deux (2) ou plus, ou si csa_sequenceid est moins que l'identifiant de séquence mis en antémémoire (en tenant compte du retour à zéro de la valeur non signée d'identifiant de séquence) alors le client DOIT retourner NFS4ERR_SEQ_MISORDERED.

* Si csa_sequenceid et l'identifiant de séquence mis en antémémoire sont le même, c'est un ré-essai, et le client retourne la réponse en antémémoire de la demande CB_COMPOUND.

* Si csa_sequenceid est supérieur de un (en tenant compte du retour à zéro) à l'identifiant de séquence en antémémoire, alors c'est une nouvelle demande, et l'identifiant de séquence du créneau est incrémenté. Les opérations suivantes de CB_SEQUENCE, si il en est, sont traitées. Si il n'y a pas d'autre opération, les seuls autres effets sont de mettre en antémémoire la réponse CB_SEQUENCE dans le créneau, de maintenir l'activité de la session, et quand le serveur reçoit la réponse CB_SEQUENCE, renouveler le prêt de l'état relatif à l'identifiant de client.

Si le serveur réutilise un identifiant de créneau et un identifiant de séquence pour une demande complètement différente, le client PEUT traiter la demande comme si elle était un ré-essai de ce qui a déjà été exécuté. Le client PEUT cependant détecter la réutilisation illégale du serveur et retourner NFS4ERR_SEQ_FAUX_RETRY.

Si CB_SEQUENCE retourne une erreur, alors l'état du créneau (identifiant de séquence, réponse en antémémoire) NE DOIT PAS changer. Voir au paragraphe 2.10.6.1.3 les conditions quand l'erreur NFS4ERR_RETRY_UNCACHED_REP pourrait être retournée.

Le client retourne deux valeurs de "highest_slotid" : csr_highest_slotid et csr_target_highest_slotid. La première est le plus fort identifiant de créneau que le client va accepter dans une future opération CB_SEQUENCE, et NE DEVRAIT PAS être moins que la valeur de csa_highest_slotid (mais voir une exception au paragraphe 2.10.6.1). La dernière est le plus fort identifiant de créneau que le client préférerait qu'utilise le serveur sur une future opération CB_SEQUENCE.

20.10 Opération 12 : CB_WANTS_CANCELLED - annule la délégation Wants en instance

20.10.1 Argument

```
struct CB_WANTS_CANCELLED4args {
        bool cwca_contended_wants_cancelled;
        bool cwca_resourced_wants_cancelled;
};
```

20.10.2 Résultat

```
struct CB_WANTS_CANCELLED4res {
        nfsstat4        cwcr_status;
};
```

20.10.3 Description

L'opération CB_WANTS_CANCELLED est utilisée pour notifier au client que certains ou tous les "wants" qu'il a enregistré pour des délégations et dispositions rappelables ont été annulés.

Si cwca_contended_wants_cancelled est VRAI, cela indique que le serveur ne va pas pousser au client de délégations qui deviendront disponibles après la fin de la contention.

Si cwca_resourced_wants_cancelled est VRAI, cela indique que le serveur ne va pas notifier au client quand il y a des ressources sur le serveur pour accorder des délégations ou dispositions.

Après avoir reçu une opération CB_WANTS_CANCELLED, le client est libre de tenter d'acquérir les délégations ou dispositions qu'il attendait, et éventuellement réenregistrer les "wants".

20.10.4 Mise en œuvre

Quand un client a une demande OPEN, WANT_DELEGATION, ou GET_DIR_DELEGATION en instance, quand un CB_WANTS_CANCELLED est envoyé, le serveur peut devoir rendre clair au client si une promesse de signaler une disponibilité de délégation est arrivée avant le CB_WANTS_CANCELLED et est donc couverte par elle, ou après le CB_WANTS_CANCELLED auquel cas elle n'était pas couverte par elle. Le serveur peut faire cette distinction en mettant les demandes appropriées dans la liste des appels de référence dans le CB_SEQUENCE associé.

20.11 Opération 13 : CB_NOTIFY_LOCK - notifie au client une possible disponibilité de verrou

20.11.1 Argument

```
struct CB_NOTIFY_LOCK4args {
    nfs_fh4         cnla_fh;
    lock_owner4   cnla_lock_owner;
};
```

20.11.2 Résultat

```
struct CB_NOTIFY_LOCK4res {
        nfsstat4        cnlr_status;
};
```

20.11.3 Description

Le serveur peut utiliser cette opération pour indiquer qu'un verrou de gamme d'octets pour le fichier et le propriétaire de verrou donnés, précédemment demandé par le client via une opération LOCK infructueuse, pourrait être disponible.

Ce rappel est destiné à être utilisé par les serveurs pour aider à réduire la latence des verrous bloquants dans le cas où ils reconnaissent qu'un client qui a interrogé pour un verrou bloquant de gamme d'octets peut maintenant être capable d'acquérir le verrou. Si le serveur prend en charge ce rappel pour un fichier donné, il DOIT établir le fanion OPEN4_RESULT_MAY_NOTIFY_LOCK quand il répond à des ouvertures réussies pour ce fichier. Cela n'engage pas le serveur à l'utilisation de CB_NOTIFY_LOCK, mais le client peut utiliser cela comme indication pour décider à quelle fréquence interroger pour les verrous déduits de cette ouverture.

Si une opération OPEN résulte en une mise à niveau, dans laquelle l'identifiant d'état retourné a une valeur "autre" correspondant à celle d'un identifiant d'état déjà alloué, avec un nouveau "seqid" indiquant un changement dans le verrou représenté, alors la valeur du fanion OPEN4_RESULT_MAY_NOTIFY_LOCK quand elle répond à ce nouvel OPEN contrôle le traitement à partir de ce point. Quand des OPEN parallèles sont faits sur le même fichier et propriétaire d'ouverture, l'ordre des champs "seqid" des identifiants d'état retournés (sous réserve du retour à zéro) est à utiliser pour choisir la valeur contrôlant le fanion OPEN4_RESULT_MAY_NOTIFY_LOCK.

20.11.4 Mise en œuvre

Le serveur NE DOIT PAS accorder le verrou de gamme d'octets au client tant qu'il n'a pas reçu une opération LOCK du client. De même, le client recevant ce rappel ne peut pas supposer qu'il a maintenant le verrou ou qu'une opération LOCK suivante pour le verrou va réussir.

Le serveur n'est pas obligé de mettre en œuvre ce rappel, et même si il le fait, il n'est pas obligé de l'utiliser dans un cas particulier. Donc, le client doit encore s'appuyer sur l'interrogation pour les verrous bloquants, comme décrit au paragraphe 9.6.

De même, le client n'est pas obligé de mettre en œuvre ce rappel, et même si il le fait, est encore libre de l'ignorer. Donc, le serveur NE DOIT PAS supposer que le client va agir sur la base du rappel.

20.12 Opération 14 : CB_NOTIFY_DEVICEID - notifie au client des changements d'identifiant d'appareil

20.12.1 Argument

```
/* * Types de notification d'appareil. */
enum notify_deviceid_type4 {
        NOTIFY_DEVICEID4_CHANGE = 1,
        NOTIFY_DEVICEID4_DELETE   = 2
};

/* Pour NOTIFY4_DEVICEID4_DELETE */
struct notify_deviceid_delete4 {
        layouttype4     ndd_layouttype;
        deviceid4        ndd_deviceid;
};

/* Pour NOTIFY4_DEVICEID4_CHANGE */
struct notify_deviceid_change4 {
      layouttype4     ndc_layouttype;
      deviceid4        ndc_deviceid;
      bool             ndc_immediate;
```

```
};

struct CB_NOTIFY_DEVICEID4args {
        notify4 cnda_changes<>;
};
```

20.12.2 Résultat

```
struct CB_NOTIFY_DEVICEID4res {
        nfsstat4        cndr_status;
};
```

20.12.3 Description

L'opération CB_NOTIFY_DEVICEID est utilisée par le serveur pour envoyer des notifications aux clients sur les changements aux identifiants d'appareil pNFS. L'enregistrement des notifications d'identifiant d'appareil est facultatif et est fait via GETDEVICEINFO. Ces notifications sont envoyées sur le canal de retour une fois que la demande d'origine a été traitée sur le serveur. Le serveur va envoyer un dispositif de notifications, cnda_changes, comme une liste de paires de gabarits binaires et de valeurs. Voir au paragraphe 3.3.7 une description du fonctionnement des gabarits binaires NFSv4.1.

Comme avec CB_NOTIFY (paragraphe 20.4.3) il est possible que le serveur ait plus de notifications qu'il n'en peut tenir dans un CB_COMPOUND, exigeant donc plusieurs CB_COMPOUND. À la différence de CB_NOTIFY, la mise en série n'est pas un problème parce que à la différence des entrées de répertoire, les identifiants d'appareil ne peuvent pas être réutilisés après avoir été supprimés (paragraphe 12.2.10).

Toutes les notifications d'identifiant d'appareil contiennent un identifiant d'appareil et un type de disposition. Le type de disposition est nécessaire parce que deux types de disposition différents peuvent partager le même identifiant d'appareil, et l'identifiant d'appareil commun peut avoir des transpositions complètement différentes pour chaque type de disposition.

Le serveur va envoyer les notifications suivantes :

NOTIFY_DEVICEID4_CHANGE : une transposition d'identifiant d'appareil en adresse d'appareil fournie précédemment a changé et le client utilise GETDEVICEINFO pour obtenir la transposition mise à jour. La notification est codée en une valeur de type de données notify_deviceid_change4. Ce type de données contient aussi un champ booléen, ndc_immediate, qui à VRAI indique que le changement va être appliqué immédiatement, et donc que le client pourrait ne pas être capable de terminer une entrée/sortie en cours sur l'identifiant d'appareil. Si ndc_immediate est FAUX, alors pour une durée indéfinie, le client peut achever l'entrée/sortie en cours. Après l'achèvement de l'entrée/sortie en cours, le client DEVRAIT obtenir les nouvelles transpositions d'identifiant d'appareil en adresse d'appareil avant d'envoyer de nouvelles demandes d'entrée/sortie aux appareils de mémorisation visés par l'identifiant d'appareil.

NOTIFY4_DEVICEID_DELETE : supprime un identifiant d'appareil des transpositions. Cette notification NE DOIT PAS être envoyée si le client a une disposition qui se réfère à l'identifiant d'appareil. En d'autres termes, si le serveur envoie une notification de suppression d'identifiant d'appareil, un des suivants est vrai pour les dispositions associées au type de disposition :

* Le client n'a jamais eu une disposition se référant à cet identifiant d'appareil.
* Le client a retourné toutes les dispositions se référant à cet identifiant d'appareil.
* Le serveur a révoqué toutes les dispositions se référant à cet identifiant d'appareil.

La notification est codée dans une valeur de type de données notify_deviceid_delete4. Après qu'un serveur a supprimé un identifiant d'appareil, il NE DOIT PAS réutiliser cet identifiant d'appareil pour le même type de disposition jusqu'à ce que l'identifiant de client soit supprimé.

20.13 Opération 10044 : CB_ILLEGAL - opération de rappel illégale

20.13.1 Argument

vide

20.13.2 Résultat

```
/* * CB_ILLEGAL : réponse pour les numéros d'opération illégaux. */
struct CB_ILLEGAL4res {
        nfsstat4        status;
};
```

20.13.3 Description

Cette opération est un bouche-trou pour coder un résultat pour traiter le cas du serveur qui envoie un code d'opération dans CB_COMPOUND qui n'est pas défini dans la spécification NFSv4.1. Voir au paragraphe 19.2.3 les détails.

Le champ État de CB_ILLEGAL4res DOIT être réglé à NFS4ERR_OP_ILLEGAL.

20.13.4. Mise en œuvre

Un serveur ne va probablement pas envoyer une opération avec le code OP_CB_ILLEGAL, mais si il le fait, la réponse va être CB_ILLEGAL4res tout comme elle le serait avec tout autre code d'opération invalide. Noter que si le client obtient un code d'opération illégal qui n'est pas OP_ILLEGAL, et si le client vérifie les codes d'opération légaux durant la phase de décodage XDR, alors une instance de type de données CB_ILLEGAL4res ne sera pas retournée.

21. Considérations sur la sécurité

Historiquement, le modèle d'authentification de NFS était fondé sur la machine entière étant le client NFS, avec le serveur NFS qui fait confiance au client NFS pour authentifier l'utilisateur final. Le serveur NFS à son tour partageait ses fichiers seulement avec des clients spécifiques, identifiés par l'adresse réseau de source du client. Dans ce modèle, la nuance de sécurité RPC AUTH_SYS identifiait simplement l'utilisateur final en utilisant le client au serveur NFS. Quand il traitait les réponses NFS, le client s'assurait que les réponses venaient de la même adresse réseau et numéro d'accès que ceux auxquels la demande avait été envoyée. Bien qu'un tel modèle soit facile à mettre en œuvre et simple à déployer et utiliser, ii n'est pas sûr. Donc, il est EXIGÉ des mises en œuvre de NFSv4.1 qu'elles prennent en charge un modèle de sécurité qui utilise l'authentification de bout en bout, où un utilisateur final sur un client s'authentifie mutuellement (via des schémas cryptographiques qui n'exposent pas de mots de passe ou clés en clair sur le réseau) avec un principal sur un serveur NFS. L'intégrité et la confidentialité des demandes et réponses NFS est également considérée. Les questions d'authentification mutuelle de bout en bout, d'intégrité, et de confidentialité sont discutées au paragraphe 2.2.1.1.1. Il y a des considérations spécifiques de l'utilisation de Kerberos V5 comme décrit au paragraphe 2.2.1.1.1.2.1.1.

Noter que d'être de mise en œuvre EXIGÉE ne signifie pas qu'il soit EXIGÉ d'utiliser ; AUTH_SYS peut être utilisé par les clients et serveurs NFSv4.1. Cependant, AUTH_SYS est simplement une nuance de sécurité FACULTATIVE dans NFSv4.1, et donc l'interopérabilité via AUTH_SYS n'est pas assurée.

Pour des raisons de réduction de frais généraux d'administration, de meilleures performances, et/ou de réduction d'utilisation de CPU, les utilisateurs de mises en œuvre de NFSv4.1 pourraient refuser d'utiliser les mécanismes de sécurité qui permettent la protection de l'intégrité sur chaque appel et réponse de procédure distante. L'utilisation de mécanismes sans protection de l'intégrité laisse l'utilisateur vulnérable à une attaque par interposition du client et serveur NFS qui modifie la demande et/ou la réponse RPC. Bien que les mises en œuvre soient libres de fournir l'option d'utiliser de plus faibles mécanismes de sécurité, il y a trois opérations en particulier qui garantissent que la mise en œuvre outrepasse les choix de l'utilisateur.

* Les deux premières de ces opérations sont SECINFO et SECINFO_NO_NAME. Il est RECOMMANDÉ que le client envoie les deux opérations de telle façon qu'elles soient protégées avec une nuance de sécurité qui a la protection de l'intégrité, comme RPCSEC_GSS avec le service rpc_gss_svc_integrity ou rpc_gss_svc_privacy. Sans protection de l'intégrité encapsulant SECINFO et SECINFO_NO_NAME et leurs résultats, un interposé pourrait modifier les résultats de telle façon que le client pourrait choisir un algorithme plus faible dans l'ensemble permis par le serveur, rendant le client et/ou serveur vulnérable à d'autres attaques.

* La troisième opération qui DEVRAIT utiliser la protection de l'intégrité est un GETATTR pour les attributs fs_locations et fs_locations_info, afin d'atténuer la sévérité d'une attaque par interposition. L'attaque a deux étapes. D'abord l'attaquant modifie le résultat non protégé d'une opération pour retourner NFS4ERR_MOVED. Ensuite, quand

le client poursuit avec un GETATTR pour les attributs fs_locations ou fs_locations_info, l'attaquant modifie le résultat pour causer la migration du trafic du client sur un serveur contrôlé par l'attaquant. Avec la protection de l'intégrité, cette attaque est déjouée.

Par rapport aux précédentes versions de NFS, NFSv4.1 a des considérations de sécurité supplémentaires pour pNFS (voir les paragraphes 12.9 et 13.12) l'état de verrouillage et de session (voir le paragraphe 2.10.8.3) et la récupération d'état durant la période de grâce (voir le paragraphe 8.4.2.1.1). Par rapport à l'état de verrouillage et de session, si la protection d'état SP4_SSV est utilisée, le paragraphe 2.10.10 a des considérations de sécurité spécifiques pour le client et serveur NFSv4.1.

Les considérations de sécurité pour la réclamation de verrou diffèrent entre les deux situations différentes dans lesquelles la réclamation d'état est à faire. La situation de défaillance du serveur est discutée au paragraphe 8.4.2.1.1, tandis que la réclamation d'état par système de fichiers faite à l'appui de migration/duplication est discutée au paragraphe 11.11.9.1.

L'utilisation de caractéristiques d'espace de noms multi serveurs décrite à la Section 11 soulève la possibilité que des demandes de détermination de l'ensemble des adresses réseau correspondant à un serveur donné pourraient interférer avec ou avoir leurs réponses modifiées en vol. À la lumière de cette possibilité, les considérations suivantes sont à noter :

* Quand le DNS est utilisé pour convertir les noms de serveur en adresses et que DNSSEC [RFC4033] n'est pas disponible, la validité des adresses réseau retournées ne peut généralement pas être fiable. Cependant, quand il est combiné avec un résolveur de confiance, DNS sur TLS [RFC7858] et DNS sur HTTPS [RFC8484] peuvent être de confiance pour fournir des résolutions d'adresses valides.

  Dans les situations où la validité des adresses fournies ne peut pas être établie et où le client utilise RPCSEC_GSS pour accéder au serveur désigné, il est possible que l'authentification mutuelle découvre des adresses de serveur invalides tant que la mise en œuvre de RPCSEC_GSS utilisée n'utilise pas des interrogations non sûres du DNS pour canoniser les composants du nom d'hôte des noms de principal de service, comme expliqué dans la [RFC4120].

* Aller chercher les attributs qui contiennent les informations de localisation de système de fichiers DEVRAIT être effectué en utilisant la protection de l'intégrité. Il est important de noter ici qu'un client qui fait une demande de cette sorte sans utiliser la protection de l'intégrité doit connaître les conséquences négatives de ce comportement, qui peuvent conduire à ce que des noms d'hôtes ou adresses réseau invalides soient retournés. Cela inclut des cas dans lesquels le client est dirigé sur un serveur sous le contrôle d'un attaquant, qui pourrait obtenir l'accès à des données écrites ou fournir des valeurs incorrectes pour des données lues. À la lumière de cela, le client doit reconnaître qu'utiliser de telles informations de localisation retournées pour accéder à un serveur NFSv4 sans utiliser RPCSEC_GSS (c'est-à-dire, en utilisant AUTH_SYS) fait peser des dangers car il peut en résulter une interaction du client avec un tel serveur contrôlé par un attaquant sans aucune facilité d'authentification pour vérifier l'identité du serveur.

* En dépit du fait que c'est une exigence que les mises en œuvre fournissent la "prise en charge" de l'utilisation de RPCSEC_GSS, on ne peut pas supposer que l'utilisation de RPCSEC_GSS est toujours disponible entre une paire particulière de client-serveur.

* Quand un client a les adresses réseau d'un serveur mais pas les noms d'hôte associés, cela va interférer avec sa capacité d'utiliser RPCSEC_GSS.

À la lumière de ceci, un serveur DEVRAIT présenter des entrées de localisation de système de fichiers qui correspondent aux systèmes de fichiers sur d'autres serveurs qui utilisent un nom d'hôte. Cela permettrait au client d'interroger les fs_locations sur le serveur de destination pour obtenir des informations de multiplexage (ainsi que des informations de duplication) en utilisant la protection de l'intégrité, en validant le nom fourni tout en assurant que la réponse n'a pas été modifiée en vol.

Quand RPCSEC_GSS n'est pas disponible sur un serveur, le client doit avoir connaissance du fait que les entrées de localisation sont sujettes à modification en vol et qu'on ne peut donc pas s'y fier. Dans le cas d'un client dirigé sur un autre serveur après NFS4ERR_MOVED, cela pourrait fausser l'authentification fournie par l'utilisation de RPCSEC_GSS sur le serveur de destination désigné. Même quand l'authentification RPCSEC_GSS est disponible à la destination, le serveur pourrait encore s'authentifier correctement comme le serveur auquel le client a été dirigé à tort. Sans moyen de décider si le serveur est valide, le client peut seulement déterminer, en utilisant RPCSEC_GSS, que le serveur correspond au nom fourni, sans aucune base pour faire confiance à ce serveur. Par suite, le client NE DEVRAIT PAS utiliser de telles entrées de localisation non vérifiées comme base d'une migration, même si RPCSEC_GSS pourrait être disponible à la destination.

Quand un attribut de localisation de système de fichiers est cherché lors de la connexion avec un serveur NFS, ce

DEVRAIT, comme déclaré ci-dessus, être fait avec la protection de l'intégrité. Quand ce n'est pas possible, il est généralement mieux pour le client d'ignorer les informations de multiplexage et de duplication ou simplement de ne pas aller chercher les informations de localisation pour cela.

Quand les informations de localisation ne peuvent pas être vérifiées, elles peuvent être soumises à un filtrage supplémentaire pour empêcher le client d'être redirigé de façon inappropriée. Par exemple, si une gamme d'adresses réseau peut être déterminée qui assure que les serveurs et clients utilisant AUTH_SYS sont soumis à l'ensemble de contraintes approprié (par exemple, isolation du réseau physique, contrôles administratifs sur les systèmes d'exploitation utilisés) alors les adresses réseau dans la gamme appropriée peuvent être utilisées avec d'autres éliminées ou interdites dans leur usage de AUTH_SYS. Pour résumer les considérations concernant l'utilisation de RPCSEC_GSS pour aller chercher les informations de localisation, on doit considérer les possibilités suivantes pour les demandes d'interroger les informations de localisation, avec des approches d'interrogation sur les serveur de référence et de destination arrivées séparément :

* L'utilisation de la protection de l'intégrité est RECOMMANDÉE dans tous les cas, car l'absence de protection de l'intégrité expose le client à la possibilité que les résultats soient modifiés dans le transit.

* L'utilisation de demandes produites sans RPCSEC_GSS (c'est-à-dire, en utilisant AUTH_SYS, qui n'a pas de disposition pour éviter de modification de données en vol) bien qu'indésirable et exposant potentiellement la sécurité, ne peut pas être évitée dans tous les cas. Lorsque l'utilisation des informations retournées ne peut pas être évitée, elle est rendue sujette à filtrage comme décrit ci-dessus pour éliminer la possibilité que le client traite une adresse invalide comme si c'était un serveur NFSv4. Les spécificités vont varier selon le degré d'isolation du réseau et si la demande est au serveur de référence ou de destination.

Même si de telles demandes ne subissent pas d'interférence en vol, il est possible à un serveur compromis de diriger le client sur des serveurs inappropriés, comme ceux sous le contrôle de l'attaquant. Il n'est pas clair que d'être dirigé sur de tels serveurs représente un menace plus grande pour le client que les dommages qui pourraient être faits par le serveur compromis lui-même. Cependant, il est possible que certaines sortes de compromissions de serveur transitoires pourraient être exploitées pour diriger un client sur un serveur capable de faire de plus grands dommages sur une plus longue période. Une étape utile pour se garder contre cette possibilité est de produire des demandes d'aller chercher les données de localisation en utilisant RPCSEC_GSS, même si aucune transposition en un principal RPCSEC_GSS n'est disponible. Dans ce cas, RPCSEC_GSS ne serait pas utilisé, comme il l'est normalement, pour identifier le client principal auprès du serveur, mais plutôt pour s'assurer (via l'authentification mutuelle RPCSEC_GSS) que le serveur contacté est celui prévu.

Des considérations similaires s'appliquent si la menace à éviter est la redirection du trafic du client sur des serveurs inappropriés (c'est-à-dire, aux pauvres performances). Dans les deux cas, il n'y a aucune raison pour que les informations retournées dépendent de l'identité du client principal les demandant, alors que la validité des informations du serveur, qui a la capacité d'affecter tous les principaux de clients, est de considérable importance.

22. Considérations relatives à l'IANA

Cette section utilise des termes qui sont définis dans la [RFC8126].

22.1 Actions de l'IANA

Cette mise à jour n'exige aucune modification, ni ajout, aux entrées de registre ou règles de registres associées à NFSv4.1. Cependant, comme le présent document rend obsolète la RFC 5661, l'IANA a mis à jour toutes les entrées de registre et les références de règles de registre qui pointent sur la RFC 5661 pour pointer à la place sur le présent document.

Les actions précédentes de l'IANA relatives à NFSv4.1 sont mentionnées dans les paragraphes suivants de la Section 22.

22.2 Définitions des attribut désignés

L'IANA a créé un registre appelé "Registre des définitions d'attributs nommés de NFSv4 ".

Le protocole NFSv4.1 prend en charge l'association d'un fichier avec zéro, un ou plusieurs attributs nommés. Les identifiants d'espace de noms pour ces attributs sont définis comme des noms de chaînes. Le protocole ne définit pas les allocations spécifiques de l'espace de noms pour ces attributs de fichier. Le registre de l'IANA promeut l'interopérabilité lorsque des intérêts communs existent. Bien qu'il soit permis aux développeurs d'application de définir et utiliser des

attributs comme nécessaire, ils sont encouragés à enregistrer les attributs auprès de l'IANA.

Ces attributs nommés enregistrés sont présumés s'appliquer à toutes les versions mineures de NFSv4, y compris celles définies après l'enregistrement. Si l'attribut nommé est destiné à être limité à des versions mineures spécifiques, cela devra être clairement déclaré dans les allocations du registre.

Toutes les allocations dans le registre sont faites sur la base du premier arrivé, premier servi, conformément au paragraphe 4.4 de la [RFC8126]. La politique pour chaque allocation est Spécification exigée, conformément au paragraphe 4.6 de la [RFC8126].

Dans la spécification NFSv4.1, le nom d'un attribut nommé peut en théorie faire jusqu'à 2^(32) - 1 octets, mais en pratique les clients et serveurs NFSv4.1 vont être incapables de traiter une chaîne de cette longueur. IANA devrait rejeter toute demande d'allocation d'un attribut nommé qui excède 128 caractères UTF-8. Pour donner à l'IESG la souplesse d'établir les bases d'allocation d'utilisation expérimentale et d'action de normalisation, les préfixes de "EXPE" et "STDS" sont réservés. L'attribut nommé avec un nom de longueur zéro est réservé.

Le préfixe "PRIV" est désigné pour l'utilisation privée. Un site qui veut faire usage d'attributs nommés non enregistrés sans risque de conflit avec une allocation du registre de l'IANA devrait utiliser le préfixe "PRIV" dans tous ses attributs nommés.

Parce que certains des clients et serveurs NFSv4.1 ont une sémantique insensible à la casse, les quinze permutations supplémentaires de minuscules et de casse mixte de "EXPE", "PRIV", et "STDS" sont réservées (par exemple, "expe", "expE", "exPe", etc. sont réservées). De même, l'IANA ne doit pas permettre deux allocations qui seraient en conflit si les deux attributs nommés étaient convertis en une casse commune.

Le registre des attributs nommés est une liste d'allocations, chacune contenant trois champs pour chaque allocation.

1. Un nom de chaîne US-ASCII qui est le nom réel de l'attribut. Ce nom doit être unique. Ce nom de chaîne peut être long de 1 à 128 caractères UTF-8.

2. Une référence à la spécification de l'attribut nommé. La référence peut consommer jusqu'à 256 octets (ou plus si l'IANA le permet).

3. Le point de contact de l'enregistrant. Le point de contact peut consommer jusqu'à 256 octets (ou plus si l'IANA le permet).

22.2.1 Registre initial

Il n'y a pas de registre initial.

22.2.2 Mise à jour des enregistrements

Il est toujours permis à l'enregistrant de mettre à jour le champ "point de contact". Tout autre changement exige une revue d'expert ou l'approbation de l'IESG.

22.3 Notifications des identifiants d'appareil

L'IANA a créé un registre appelé "Registre des notifications d'identifiant d'appareil NFSv4".

Il existe un potentiel d'ajout de nouveaux types de notification à l'opération CB_NOTIFY_DEVICEID (voir le paragraphe 20.12). Cela peut être fait via des changements aux opérations qui enregistrent les notifications, ou par l'ajout de nouvelles opérations à NFSv4. Cela exige une nouvelle version mineure de NFSv4, et exige un document sur la voie de la normalisation de la part de l'IETF. Une autre façon d'ajouter une notification est de spécifier un nouveau type de disposition (voir le paragraphe 22.5).

Donc, toutes les allocations au registre sont faites sur la base d'une action de normalisation conformément au paragraphe 4.6 de [RFC8126], avec revue d'expert exigée.

Le registre est une liste des allocations, chacune contenant cinq champs par allocation.

1. Le nom du type de notification. Ce nom doit avoir le préfixe "NOTIFY_DEVICEID4_". Ce nom doit être unique.

2. La valeur de la notification. L'IANA va allouer ce nombre, et la demande de l'enregistrant va utiliser TBD1 au lieu d'une valeur réelle. L'IANA DOIT utiliser un nombre entier qui ne peut pas être supérieur à 2^(32) - 1, et devrait être la prochaine valeur disponible. La valeur allouée doit être unique. Un expert désigné doit être utilisé pour s'assurer que quand le nom du type de notification et sa valeur sont ajoutés au type énuméré NFSv4.1 notify_deviceid_type4 de données dans la description XDR NFSv4.1 [RFC5662], le résultat continue d'être une description XDR valide.

3. La ou les RFC sur la voie de la normalisation qui décrivent la notification. Si la ou les RFC n'ont pas encore été publiées, l'enregistrant va utiliser RFCTBD2, RFCTBD3, etc. au lieu d'un numéro de RFC réel.

4. Comment la RFC introduit la notification. Ceci est indiqué par une seule valeur US-ASCII. Si la valeur est N, cela signifie une révision mineure au protocole NFSv4. Si la valeur est L, cela signifie un nouveau type de disposition pNFS. D'autres valeurs peuvent être utilisées avec l'approbation de l'IESG.

5. Les versions mineures de NFSv4 qui sont autorisées à utiliser la notification. Bien que ce soient des valeurs numériques, l'IANA ne va pas les allouer ; l'auteur des RFC pertinentes avec l'approbation de l'IESG alloue ces numéros. Chaque fois qu'il y a une nouvelle version mineure de NFSv4 approuvée, un expert désigné devrait revoir le registre pour faire les recommandations de mise à jour nécessaires.

22.3.1 Registre initial

Le registre initial est dans le Tableau 25. Noter que la prochaine valeur disponible est zéro.

Nom de notification	Valeur	RFC	Comment	Versions mineures
NOTIFY_DEVICEID4_CHANGE	1	RFC 8881	N	1
NOTIFY_DEVICEID4_DELETE	2	RFC 8881	N	1

Tableau 25 : Allocations initiales de notification d'identifiant d'appareil

22.3.2 Mise à jour des enregistrements

La mise à jour d'un enregistrement exige l'approbation de l'IESG sur avis d'un expert désigné.

22.4 Types de rappel d'objet

L'IANA a créé un registre appelé "Registre des types d'objets rappelables de NFSv4".

Il existe un potentiel d'ajout de nouveaux types d'objet à l'opération CB_RECALL_ANY (voir le paragraphe 20.6). Cela peut être fait via des changements aux opérations qui ajoutent des types rappelables, ou par l'ajout de nouvelles opérations à NFSv4. Cela exige une nouvelle version mineure de NFSv4, et exige un document sur la voie de la normalisation de la part de l'IETF. Une autre façon d'ajouter un nouvel objet rappelable est de spécifier un nouveau type de disposition (voir le paragraphe 22.5).

Toutes les allocations dans le registre sont faites par une action de normalisation selon le paragraphe 4.9 de la [RFC8126], avec revue d'expert exigée.

Les types d'objets rappelables sont des nombres de 32 bits non signés. Ce sont des valeurs non réservées. Les valeurs dans la gamme 12 à 15, inclus, sont conçues pour usage privé.

Le registre est une liste des allocations, chacune contenant cinq champs par allocation.

1. Le nom du type d'objet rappelable. Ce nom doit avoir le préfixe "RCA4_TYPE_MASK_". Le nom doit être unique.

2. La valeur du type d'objet rappelable. L'IANA va allouer ce numéro, et la demande de l'enregistrant va utiliser TBD1 au lieu d'une valeur réelle. L'IANA DOIT utiliser un nombre entier qui ne peut pas être supérieur à 2^(32) - 1, et devrait être la prochaine valeur disponible. La valeur doit être unique. Un expert désigné doit être utilisé pour s'assurer que quand le nom du type rappelable et sa valeur sont ajoutés à la description XDR de NFSv4 [RFC5662], le résultat

continue d'être une description XDR valide.

3. La ou les RFC sur la voie de la normalisation qui décrivent le type d'objet rappelable. Si la ou les RFC n'ont pas encore été publiées, l'enregistrant va utiliser RFCTBD2, RFCTBD3, etc. au lieu d'un numéro de RFC réel.

4. Comment la RFC introduit le type d'objet rappelable. Cela est indiqué par une seule valeur US-ASCII. Si la valeur est N, cela signifie une révision mineure du NFSv4. Si la valeur est L, cela signifie un nouveau type de disposition pNFS. D'autres valeurs peuvent être utilisées avec l'approbation de l'IESG.

5. Les versions mineures de NFSv4 à qui il est permis d'utiliser le type d'objet rappelable. Bien que ce soient des valeurs numériques, l'IANA ne va pas les allouer ; l'auteur des RFC pertinentes les alloue avec l'approbation de l'IESG. Chaque fois qu'une nouvelle version mineure de NFSv4 est approuvée, un expert désigné devrait revoir le registre pour faire les recommandations de mise à jour nécessaires.

22.4.1 Registre initial

Le registre initial est dans le Tableau 26. Noter que la prochaine valeur disponible est cinq.

Nom du type d'objet rappelable	Valeur	RFC	Comment	Versions mineures
RCA4_TYPE_MASK_RDATA_DLG	0	RFC 8881	N	1
RCA4_TYPE_MASK_WDATA_DLG	1	RFC 8881	N	1
RCA4_TYPE_MASK_DIR_DLG	2	RFC 8881	N	1
RCA4_TYPE_MASK_FILE_LAYOUT	3	RFC 8881	N	1
RCA4_TYPE_MASK_BLK_LAYOUT	4	RFC 8881	L	1
RCA4_TYPE_MASK_OBJ_LAYOUT_MIN	8	RFC 8881	L	1
RCA4_TYPE_MASK_OBJ_LAYOUT_MAX	9	RFC 8881	L	1

Tableau 26 : Allocations initiales de type d'objet rappelable

22.4.2 Mise à jour des enregistrements

La mise à jour d'un enregistrement exige l'approbation de l'IESG sur avis d'un expert désigné.

22.5 Types de disposition

L'IANA a créé un registre appelé "Registre des types de disposition pNFS".

Toutes les allocations sur ce registre sont faites sur la base d'une action de normalisation, avec avis d'expert exigé.

Les types de disposition sont des nombres de 32 bits. La valeur zéro est réservée. Les valeurs dans la gamme 0x80000000 à 0xFFFFFFFF inclus sont destinées à utilisation privée. L'IANA allouera les nombres de la gamme 0x00000001 à 0x7FFFFFFF inclus.

Le registre est une liste d'allocations, chacune contenant cinq champs.

1. Le nom du type de disposition. Ce nom doit avoir le préfixe "LAYOUT4_". Le nom doit être unique.

2. La valeur du type de disposition. L'IANA alloue ce numéro, et la demande de l'enregistrant utilisera TBD1 au lieu d'une valeur réelle. La valeur allouée doit être unique. Un expert désigné doit être utilisé pour assurer que quand le nom du type de disposition et sa valeur sont ajoutés au type de données NFSv4.1 dispositiontype4 énuméré dans la description XDR NFSv4.1 [RFC5662], le résultat continue d'être une description XDR valide.

3. La ou les RFC sur la voie de la normalisation qui décrivent la notification. Si la ou les RFC n'ont pas encore été publiées, l'enregistrant va utiliser RFCTBD2, RFCTBD3, etc. aui lieu d'un numéro réel de RFC. Collectivement, la ou les RFC doivent respecter les lignes directrices du paragraphe 22.5.3.

4. Comment la RFC introduit le type de disposition. Ceci est indiqué par une seule valeur US-ASCII. Si la valeur est N, cela signifie une révision mineure au protocole NFSv4. Si la valeur est L, cela signifie un nouveau type de disposition pNFS. D'autres valeurs peuvent être utilisées avec l'approbation de l'IESG.

5. Les versions mineures de NFSv4 à qui il est permis d'utiliser la notification. Bien que ce soient des valeurs numériques, l'IANA ne va pas les allouer ; l'auteur des RFC pertinentes avec l'approbation de l'IESG alloue ces numéros. Chaque fois qu'il y a une nouvelle version mineure de NFSv4 approuvée, un expert désigné devrait revoir le registre pour faire les recommandations de mise à jour nécessaires.

22.5.1 Registre initial

Le registre initial est dans le Tableau 27.

Nom de type de disposition	Valeur	RFC	Comment	Versions mineures
LAYOUT4_NFSV4_1_FILES	0x1	RFC 8881	N	1
LAYOUT4_OSD2_OBJECTS	0x2	RFC 5664	L	1
LAYOUT4_BLOCK_VOLUME	0x3	RFC 5663	L	1

Tableau 27 : Allocations initiales de type de disposition

22.5.2 Mise à jour des enregistrements

La mise à jour d'un enregistrement exige l'approbation de l'IESG sur l'avis d'un expert désigné.

22.5.3 Lignes directrices pour écrire les spécifications de type de disposition

L'auteur d'une nouvelle spécification de disposition pNFS doit suivre ces étapes pour obtenir l'acceptation du type de disposition comme RFC sur la voie de la normalisation :

1. L'auteur rédige la spécification de la nouvelle disposition.
2. La spécification du nouveau type de disposition DOIT, au minimum :
   * Définir le contenu des champs spécifiques du type de disposition des types de données suivants :
   - le champ da_addr_body du type de données device_addr4 ;
   - le champ loh_body du type de données layouthint4 ;
   - le champ loc_body du type de données layout_content4 (qui est le champ lo_content du type de données layout4) ;
   - le champ lou_body du type de données layout4update4 ;
   * Décrire ou définir le protocole d'accès à la mémorisation utilisée pour accéder aux appareils de mémorisation.
   * Décrire si la révocation de dispositions est prise en charge.
   * Au minimum, décrire les méthodes de récupération de :
     1. défaillance et redémarrage pour le client, le serveur, l'appareil de mémorisation ;
     2. l'expiration du prêt dans la perspective du client actif, du serveur, de l'appareil de mémorisation ;
     3. la perte de l'état de disposition résultant au barrage de l'accès du client aux appareils de mémorisation (pour un exemple, voir le paragraphe 12.7.3).
   * Inclure une section de considérations relatives à l'IANA, qui devront inclure :
     - une demande à l'IANA pour un nouveau type de disposition selon le paragraphe 22.5 ;
     - une liste de demandes à IANA pour tout nouveau type d'objet rappelable pour CB_RECALL_ANY ; chaque entrée doit être présentée sous la forme décrite au paragraphe 22.4 ;
     - une liste de demandes à l'IANA pour toute nouvelle valeur de notification pour CB_NOTIFY_DEVICEID ; chaque entrée doit être présentée sous la forme décrite au paragraphe 22.3.
   * Inclure une section de considérations sur la sécurité. Cette section DOIT expliquer comment les modèles NFSv4.1 d'authentification, d'autorisation, et de contrôle d'accès sont préservés. C'est-à-dire, si un serveur de métadonnées va restreindre une opération READ ou WRITE, comment pNFS va via la disposition restreindre de façon similaire une opération correspondante d'entrée ou de sortie ?
3. L'auteur documente la spécification de la nouvelle disposition comme un projet Internet.
4. L'auteur soumet le projet Internet pour revue au processus de normalisation de l'IETF comme défini dans le BCP 9 "Processus de normalisation de l'Internet -- révision 3" [RFC2026]). La spécification de la nouvelle disposition sera soumise pour éventuelle publication comme RFC sur la voie de la normalisation.
5. La spécification de disposition progresse à travers le processus de normalisation de l'IETF.

22.6 Définitions des variables de chemin

Ce paragraphe traite des considérations de l'IANA associées à la caractéristique de substitution de variable pour les noms

de localisation comme décrit au paragraphe 11.17.3. Les variables sujettes à substitution consistent en un nom de domaine et un nom spécifique au sein de ce domaine, avec les deux séparés par deux-points. Il y a là deux ensembles de considérations de l'IANA :
1. La liste de noms de variables.
2. Pour chaque nom de variable, la liste des valeurs possibles.

Donc, il va y avoir un registre pour la liste des noms de variables, et éventuellement un registre pour la liste des valeurs de chaque nom de variable.

22.6.1 Registre des variables de chemin

L'IANA a créé un registre appelé "Registre des variables de chemin NFSv4".

22.6.1.1 Valeurs des variables de chemin

Les noms de variables sont de la forme "${", suivi par un nom de domaine, suivi par deux-point (":") suivi par une portion spécifique du domaine du nom de variable, suivi par "}". Quand le nom de domaine est "ietf.org", tous les noms de variables doivent être enregistrés auprès de l'IANA sur la base d'une action de normalisation, avec revue d'expert exigée. Les variables de chemin avec des noms de domaine enregistrés ni partie ni égaux à ietf.org sont alloués sur une base hiérarchique (déléguée au propriétaire du domaine) et donc ne concernant pas l'IANA, sauf si le propriétaire du domaine choisit d'enregistrer un nom de variable de son domaine. Si le propriétaire du domaine fait ce choix, IANA l'enregistrera sur la base du premier arrivé, premier servi. Pour traiter les enregistrants qui n'ont pas leur propre domaine, l'IANA acceptera les demandes d'enregistrement de variables avec le prefix "${FCFS.ietf.org:" sur la base du premier arrivé, premier servi. Les allocations sur la base du premier arrivé, premier servi n'exigent pas de revue d'expert, sauf si l'enregistrant veut aussi que l'IANA établisse un registre pour les valeurs de la variable enregistrée.

Le registre est une liste d'allocations, chacune contenant trois champs.

1. Le nom de la variable. Il doit commencer par "${" suivi par un nom de domaine enregistré, suivi par ":", ou il doit commencer par "${FCFS.ietf.org". Le nom ne doit pas faire plus de 64 caractères UTF-8. Le nom doit être unique.

2 Pour les allocations faites sur la base d'une action de normalisation, la ou les RFC sur la voie de la normalisation qui décrivent la variable. Si la ou les RFC n'ont pas encore été publiées, l'enregistrant utilisera RFCTBD1, RFCTBD2, etc. au lieu d'un numéro de RFC réel. Noter que les RFC n'ont pas à faire partie d'une version mineure de NFS. Pour les allocations faites sur la base du premier arrivé, premier servi, une explication (ne consommant pas plus de 1024 octets, ou plus si l'IANA le permet) de l'objet de la variable. Une référence à l'explication peut être substituée.

3. Le point de contact, incluant une adresse de messagerie électronique. Le point de contact peut consommer jusqu'à 256 octets (ou plus si l'IANA le permet). Pour les allocations faites sur la base d'une action de normalisation, le point de contact est toujours l'IESG.

22.6.1.1.1 Registre initial

Le registre initial est au Tableau 28.

Nom de variable	RFC	Point de contact
${ietf.org:CPU_ARCH}	RFC 8881	IESG
${ietf.org:OS_TYPE}	RFC 8881	IESG
${ietf.org:OS_VERSION}	RFC 8881	IESG

Tableau 28 : Liste initiale des variables de chemin

L'IANA a créé des registres pour les valeurs de noms de variables ${ietf.org:CPU_ARCH} et ${ietf.org:OS_TYPE}. Voir les paragraphes 22.6.2 et 22.6.3.

Pour les valeurs de variable ${ietf.org:OS_VERSION}, aucun registre n'est nécessaire car les spécificités des valeurs de la variable varient avec la valeur de ${ietf.org:OS_TYPE}. Donc, les valeurs pour ${ietf.org:OS_VERSION} sont sur une base d'allocation hiérarchique et ne sont pas du ressort de l'IANA.

22.6.1.1.2 Mise à jour des enregistrements

La mise à jour d'une allocation faite sur la base d'une action de normalisation exige l'approbation de l'IESG sur avis d'un expert désigné.

L'enregistrant peut toujours mettre à jour le point de contact d'une allocation faite sur la base du premier arrivé, premier servi. Toute autre mise à jour exige une revue d'expert.

22.6.2 Valeurs pour la variable ${ietf.org:CPU_ARCH}

L'IANA a créé un registre appelé "Registre NFSv4 des valeurs de ${ietf.org:CPU_ARCH}".

Les allocations au registre sont faites sur la base du premier arrivé, premier servi. La valeur de longueur zéro de ${ietf.org:CPU_ARCH} est réservée. Les valeurs avec un préfixe de "PRIV" sont destinées à une utilisation privée.

Le registre est une liste d'allocations, chacune contenant trois champs.

1. Une valeur de la variable ${ietf.org:CPU_ARCH}. La valeur doit être longue de 1 à 32 caractères UTF-8. La valeur doit être unique.

2. Une explication (ne consommant pas plus de 1024 octets, ou plus si l'IANA le permet) de ce que note la valeur de l'architecture de CPU. Une référence à l'explication peut être substituée.

3. Le point de contact, incluant une adresse de messagerie électronique. Le point de contact peut consommer jusqu'à 256 octets (ou plus si l'IANA le permet).

22.6.2.1 Registre initial

Il n'y a pas de registre initial.

22.6.2.2 Mise à jour des enregistrements

Celui qui enregistre est libre de mettre à jour l'allocation, c'est-à-dire, changer les champs d'explication et/ou de point de contact.

22.6.3 Valeurs pour la variable ${ietf.org:OS_TYPE} Variable

L'IANA a créé un registre appelé "Registre NFSv4 des valeurs de ${ietf.org:OS_TYPE}".

Les allocations au registre sont faites sur la base du premier arrivé, premier servi. La valeur de longueur zéro de ${ietf.org:OS_TYPE} est réservée. Les valeurs avec un préfixe de "PRIV" sont destinées à une utilisation privée.

Le registre est une liste d'allocations, chacune contenant trois champs.

1. Une valeur de la variable ${ietf.org:OS_TYPE}. La valeur doit être longue de 1 à 32 caractères UTF-8. La valeur doit être unique.

2. Une explication (ne consommant pas plus de 1024 octets, ou plus si l'IANA le permet) de ce que note la valeur de l'architecture de CPU. Une référence à l'explication peut être substituée.

3. Le point de contact, incluant une adresse de messagerie électronique. Le point de contact peut consommer jusqu'à 256 octets (ou plus si l'IANA le permet).

22.6.3.1 Registre initial

Il n'y a pas de registre initial.

22.6.3.2 Mise à jour des enregistrements

Celui qui enregistre est libre de mettre à jour l'allocation, c'est-à-dire, changer les champs d'explication et/ou de point de contact.

23. Références

23.1 Références normatives

[IEEE-1003.1] Norme IEEE 1003.1, "The Open Group Base Specifications Issue 6", édition 2004, ISBN 1931624232, <https://www.opengroup.org>.

[ISO-10646-1] Norme ISO/CEI 10646, "Technologie de l'information – Jeu de caractères universel codé sur plusieurs octets (UCS) - Partie 1 : Architecture et plan multilingue de base", mai 1993.

[NIST] National Institute of Standards and Technology, "Computer Security Objects Register", mai 2016, <https://csrc.nist.gov/projects/computer-security-objects-register/algorithm-registration>.

[RFC2026] S. Bradner, "Le processus de normalisation de l'Internet -- Révision 3", (BCP0009) octobre 1996. *(Remplace* RFC1602, RFC1871) (*MàJ par* RFC3667, 3668, 3932, 3979, 3978, 5378, 6410, 8179, 8789, 9282)

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. DOI 10.17487/RFC2119, *(MàJ par* RFC8174)

[RFC2203] M. Eisler, A. Chiu, L. Ling, "Spécification du protocole RPCSEC_GSS", DOI 10.17487/RFC2203, septembre 1997. *(P.S.)*

[RFC2277] H. Alvestrand, "Politique de l'IETF en matière de jeux de caractères et de langages", BCP 18, DOI 10.17487/RFC2277, janvier 1998.

[RFC2743] J. Linn, "Interface générique de programme d'application de service de sécurité, version 2, mise à jour 1", DOI 10.17487/RFC2743, janvier 2000. (*MàJ par* RFC5554)

[RFC3454] P. Hoffman et M. Blanchet, "Préparation de chaînes internationalisées ("stringprep")", décembre 2002, DOI 10.17487/RFC3454. *(P.S.)*

[RFC3491] P. Hoffman et M. Blanchet, "Nameprep : Profil Stringprep pour les noms de domaine internationalisés (IDN)", DOI 10.17487/RFC3491, mars 2003. (*Remplacée par la* RFC5891, *P.S.*)

[RFC4033] R. Arends, et autres, "Introduction et exigences pour la sécurité du DNS", mars 2005, DOI 10.17487/RFC4033

[RFC4055] J. Schaad et autres, "Algorithmes et identifiants supplémentaires pour la cryptographie RSA à utiliser dans le profil de certificat d'infrastructure de clé publique X.509 et de liste de révocation de certificat (CRL) pour l'Internet", DOI 10.17487/RFC4055, juin 2005.

[RFC4120] C. Neuman et autres, "Service Kerberos d'authentification de réseau (V5)", DOI 10.17487/RFC4120, juillet 2005. (*MàJ par* RFC4537, 5021, 6649, 7751, 8062, 8129, 8429)

[RFC4121] L. Zhu et autres, "Version 2 du mécanisme d'interface de programme d'application de service de sécurité générique (GSS-API) de Kerberos version 5", DOI 10.17487/RFC4121, juillet 2005. (*MàJ* RFC1964) (*MàJ par les* RFC6542, 6649, 8062)) *(P.S.)*

[RFC4506] M. Eisler, éd., "XDR : norme de représentation des données externes", DOI 10.17487/RFC4506, mai 2006. (STD0067)

[RFC5040] R. Recio et autres, "Spécification d'un protocole d'accès direct à une mémoire distante", DOI 10.17487/RFC5040, octobre 2007. *(P.S. ; MàJ par* RFC7146 *)*

[RFC5403] M. Eisler, "RPCSEC_GSS version 2", DOI 10.17487/RFC5403, février 2009. (*MàJ* RFC2203 ; *P.S. ; MàJ par* RFC7861)

[RFC5531] R. Thurlow, "RPC : version 2 de la spécification du protocole d'appel de procédure à distance", DOI

10.17487/RFC5531, mai 2009. *(P.S. ; remplace* RFC1831 ; *MàJ par* RFC9289*)*

[RFC5657] L. Dusseault, R. Sparks, "Guide pour les rapports sur l'interfonctionnement et la mise en œuvre de l'avancement des projets de normes", BCP0009, septembre 2009. *(MàJ* RFC2026)

[RFC5662] S. Shepler, M. Eisler, D. Noveck, "Système de fichiers réseau (NFS) version 4.1 : Description de la norme de représentation des données externes (XDR)", DOI 10.17487/RFC5662, janvier 2010. *(P.S.)*

[RFC5665] M. Eisler, "Considérations de l'IANA pour les formats d'identifiant de réseau et d'adresse universelle dans les appels de procédure distante (RPC)", DOI 10.17487/RFC5665, janvier 2010. (*MàJ* RFC1833) *(P.S.)*

[RFC6410] R. Housley, D. Crocker, E. Burger, "Réduction de la voie de la normalisation à deux niveaux de maturité", octobre 2011. (*MàJ la* RFC2026) (BCP009)

[RFC7100] P. Resnick, "Retrait du document de résumé des "Normes officielles de protocole de l'Internet"", BCP0009, décembre 2013. (*Remplace* RFC5000) (*MàJ* RFC2026)

[RFC7127] O. Kolkman, S. Bradner, S. Turner, "Caractérisation des normes proposées", BCP0009, janvier 2014. (*MàJ* RFC2026)

[RFC7475] S. Dawkins, "Augmentation du nombre de directeurs dans les zones de l'IETF", mars 2015. BCP 9. *MàJ* 2026, 2418

[RFC7858] Z. Hu, et autres, "Spécification pour le DNS sur TLS", DOI 10.17487/RFC7858, mai 2016. *(P.S. ; MàJ par* RFC8310, 8331)

[RFC7861] W. Adamson, N. Williams, "Sécurité de l'appel de procédure distante (RPC) version 3", DOI 10.17487/RFC7861, novembre 2016. *(P.S. ; MàJ* RFC5403*)*

[RFC8000] W. Adamson, N. Williams, "Exigences pour le déploiement de l'espace de noms multi domaines NFSv4", DOI 10.17487/RFC8000,novembre 2016. *(Information)*

[RFC8166] C. Lever, W. Simpson, T. Talpey, "Transport d'accès direct à la mémoire distante pour l'appel de procédure distante version 1", DOI 10.17487/RFC8166, juin 2017. *(P.S ; MàJ* RFC5666 ; *MàJ par* RFC8797*)*

[RFC8267] C. Lever, "Lien de couche supérieure de système de fichier réseau (NFS) à RPC sur RDMA version 1", DOI 10.17487/RFC8267, octobre 2017. *(P.S. ; remplace* RFC5667*)*

[RFC8484] P. Hoffman, P. McManus, "Interrogations au DNS sur HTTPS", DOI 10.17487/RFC8484, octobre 2018. *(P.S.)*

23.2 Références pour information

[AFS] Spasojevic, M. and M. Satayanarayanan, "An Empirical Study of a Wide-Area Distributed File System", ACM Transactions on Computer Systems, Vol. 14, No. 2, pp. 200-222, DOI 10.1145/227695.227698, mai 1996, <https://doi.org/10.1145/227695.227698>.

[ERR-2006] RFC Errata, Erratum 2006 à la RFC 5661, <https://www.rfc-editor.org/errata/eid2006>.

[Errata] IESG, "IESG processing of errata for the IETF Stream", juillet 2008, <https://www.ietf.org/about/groups/iesg/statements/traitement-rfc-errata/>.

[FCP-2] Snively, R., "Fibre Channel Protocol for SCSI, 2nd Version (FCP-2)", ANSI/INCITS, 350-2003, octobre 2003.

[HANS] Bhide, A., Elnozahy, E. N., and S. P. Morgan, "A Highly Available Network Server", USENIX Conference Proceedings, janvier 1991.

[NFS-Serv] Juszczak, C., "Improving the Performance and Correctness of an NFS Server", USENIX Conference Proceedings, juin 1990.

[OSD] Weber, R.O., "Object-Based Storage Device Commands (OSD)", ANSI/INCITS, 400-2004, juillet 2004, <https://www.t10.org/drafts.htm>.

[PVFS] Carns, P. H., Ligon III, W. B., Ross, R. B., and R. Thakur, "PVFS: A Parallel File System for Linux Clusters.", Proceedings of the 4th Annual Linux Showcase and Conference, 2000.

[REV] Roach, A., "Process for Handling Non-Major Revisions to Existing RFCs", Travail en cours, Internet-Draft, draft-roach-bis-documents-00, 7 mai 2019, <https://tools.ietf.org/html/draft-roach-bis-documents-00>.

[RFC1813] B. Callaghan, B. Pawloowski et P. Staubach, "Spécification du protocole NFS version 3", DOI 10.17487/RFC1813, juin 1995. *(Information)*

[RFC1833] R. Srinivasan, "Protocoles de liaison pour RPC ONC version 2", DOI 10.17487/RFC1833, août 1995. *(P.S.) (MàJ par* RFC5665)

[RFC2054] B. Callaghan, "Spécification de client WebNFS", DOI 10.17487/RFC2054, octobre 1996. *(Information)*

[RFC2055] B. Callaghan, "Spécification de serveur WebNFS", DOI 10.17487/RFC2055, octobre 1996. *(Information)*

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997, DOI 10.17487/RFC2104.

[RFC2224] B. Callaghan, "Schéma d'URL NFS", DOI 10.17487/RFC2224, octobre 1997. *(Information)*

[RFC2623] M. Eisler, "Questions de sécurité de NFS versions 2 et 3 et utilisation de RPCSEC_GSS et Kerberos v5 par le protocole NFS", DOI 10.17487/RFC2623, juin 1999. *(P.S.)*

[RFC2624] S. Shepler, "Considérations sur la conception de NFS version 4", DOI 10.17487/RFC2624, juin 1999. *(Info.)*

[RFC2755] A. Chiu, M. Eisler, B. Callaghan, "Négociation de sécurité pour WebNFS", DOI 10.17487/RFC2755, janvier 2000. *(Information)*

[RFC2847] M. Eisler, "LIPKEY – mécanisme de clé publique à infrastructure basse utilisant SPKM", DOI 10.17487/RFC2847, juin 2000. *(P.S.)*

[RFC3232] J. Reynolds, "Numéros alloués : la RFC 1700 est remplacée par une base de données en ligne", janvier 2002.

[RFC3530] S. Shepler et autres, "Protocole de système de fichiers réseau (NFS) v. 4", DOI 10.17487/RFC3530, avril 2003. (*P.S. ; remplacée par* RFC7530)

[RFC3552] E. Rescorla, B. Korver, "Lignes directrices pour la rédaction d'une section de considérations sur la sécurité dans les RFC", DOI 10.17487/RFC3552, juillet 2003. (BCP0072)

[RFC5661] S. Shepler, M. Eisler, D. Noveck, "Système de fichiers réseau (NFS) version 4.1 : Protocole", DOI 10.17487/RFC5661, janvier 2010. *(P.S. ; MàJ par* RFC8178, RFC8434 *; remplacée par* RFC8881)

[RFC5663] D. Black, S. Fridella, J. Glasgow, "Disposition parallèle de bloc/volume du système de fichier réseau (pNFS)", DOI 10.17487/RFC5663, janvier 2010. *(P.S.)* (*MàJ par la* RFC6688)

[RFC5664] B. Halevy, B. Welch, J. Zelenka, "Opérations fondées sur l'objet en NFS parallèle (pNFS)", DOI 10.17487/RFC5664, janvier 2010. *(P.S.)*

[RFC7143] M. Chadalapaka et autres, "Protocole (consolidé) d'interface Internet de petit système d'ordinateur (iSCSI)", DOI 10.17487/RFC7143, avril 2014. (*Remplace* RFC3720, 3980, 4850, 5048) (*MàJ* RFC3721) *(P.S.)*

[RFC7258] S. Farrell, H. Tschofenig, "La surveillance envahissante est une attaque", BCP0188, DOI 10.17487/RFC7258, mai 2014. *(MàJ par* RFC8044)

[RFC7530] T. Haynes, D. Noveck, "Protocole du système de fichier réseau (NFS) version 4", DOI 10.17487/RFC7530, mars 2015. *(P.S. ; Remplace* 3580, *MàJ par* RFC7931, RFC8587)

[RFC7931] D. Noveck, et autres, "Mise à jour de la spécification de la migration de NFS4.0", DOI 10.17487/RFC7931, juillet 2016. *(P.S. ; MàJ* RFC7530*)*

[RFC8126] M. Cotton, B. Leiba, T. Narten, "Lignes directrices pour la rédaction d'une section de considérations relatives à l'IANA dans les RFC", juin 2017. BCP 26, DOI 10.17487/RFC8126, *(Remplace* RFC5226)

[RFC8178] D. Noveck, "Règles pour les extensions NFSv4 et les versions mineures", DOI 10.17487/RFC8178, juillet 2017. *(P.S. ; MàJ* 5661*,* 7862*)*

[RFC8434] T. Haynes, "Exigences pour les types d'agencement de système de fichiers réseau parallèles (pNFS)", DOI 10.17487/RFC8434, août 2018. *(P.S. ; MàJ* RFC5661)

[RPC-XID] Werme, R., "RPC XID Issues", USENIX Conference Proceedings, février 1996.

[SYNCHRO] Floyd, S. and V. Jacobson, "The Synchronization of Periodic Routing Messages", IEEE/ACM Transactions on Networking, 2(2), pp. 122-136, avril 1994.

[XNFS] The Open Group, "Protocols for Interworking: XNFS, Version 3W", ISBN 1-85912-184-5, février 1998.

Appendice A. Motifs de cette mise à jour

Le présent document inclut une explication de comment clients et serveurs déterminent les chemins d'accès réseau particuliers à utiliser pour accéder à un système de fichiers. Cela inclut des descriptions de comment traiter les changements aux répliques spécifiques à utiliser ou à l'ensemble d'adresses à utiliser pour y accéder, et comment traiter de façon transparente les transferts de responsabilité qui doivent être faits. Cela inclut les cas dans lesquels il y a un glissement d'une réplique à une autre et ceux dans lesquels différents chemins d'accès réseau sont utilisés pour accéder à la même réplique.

Par suite de problèmes dans la [RFC5661], il a été nécessaire de fournir les mises à jour spécifiques faites par le présent document. Ces mises à jour sont décrites dans l'Appendice B.

* La [RFC5661], bien qu'elle traite de situations dans lesquelles diverses formes de mise en grappes permettaient la coordination de l'état alloué par les serveurs coopérants à utiliser, ne donnait pas de disposition pour la migration d'état transparente. Dans NFSv4.0, la migration d'état transparente a d'abord été expliquée clairement dans la [RFC7530] et corrigée et précisée dans la [RFC7931]. Aucune explication correspondante pour NFSv4.1 n'avait été fournie.

* Bien que NFSv4.1 fournisse une claire définition de comment la détection de multiplexage doit être faite, il n'y avait pas de claire spécification de comment la découverte de multiplexage devait être faite, en dépit du fait que la spécification indiquait clairement que cette information pourrait être rendue disponible via les attributs de localisation de système de fichiers.

* Parce que l'existence de plusieurs chemins d'accès réseau au même système de fichiers était traitée comme si il y avait plusieurs répliques, les questions relatives aux transitions entre les répliques ne pouvaient jamais être clairement distinguées des transitions relatives au multiplexage entre les adresses utilisées pour accéder à une instance de système de fichiers particulière. Par suite, dans des situations où les deux changements de configuration de migration et de multiplexage étaient impliqués, aucun d'eux ne pouvait être traité clairement, et les relations entre ces deux caractéristiques n'étaient pas sérieusement établies.

* Parce que l'utilisation de deux chemins d'accès réseau pour la même instance de système de fichiers (c'est-à-dire, multiplexage) était souvent traitée comme si deux répliques étaient impliquées, il était considéré que les deux répliques étaient utilisées simultanément. Par suite, le traitement des répliques utilisées simultanément dans la [RFC5661] n'était pas clair, car il couvrait les deux cas distincts d'une seule instance de système de fichiers accédée par deux différents chemins d'accès réseau et deux répliques accédées simultanément, avec les limitations de ce dernier cas pas clairement établies.

La majorité des conséquences de ces questions est traitée en présentant à la Section 11 un remplacement de la Section 11 de la [RFC5661]. Ce remplacement modifie les paragraphes existants dans cette section et en ajoute de nouveaux, comme décrit à l'Appendice B.1. Aussi, certains paragraphes existants ont été supprimés. Ces changements ont été faits pour :

* Réorganiser la description pour que le cas de deux chemins d'accès réseau à la même instance de système de fichiers soit distinguée clairement du cas de deux différentes répliques car, dans le premier cas, l'état de verrouillage est partagé et qu'il peut aussi y avoir partage de l'état de session.

* Fournir une déclaration claire concernant le désir de transfert d'état transparent entre les répliques ainsi qu'une recommandation que un transfert transparent ou une période de grâce d'un seul système de fichiers soit fourni.

* Retracer spécifiquement comment un client va traiter de tels transferts, en tenant compte des différences avec le traitement de la [RFC7931] rendues nécessaires par les changements majeur du protocole dans NFSv4.1.

* Discuter les relations entre le transfert d'état transparent et NFS parallèle (pNFS).

* Préciser l'attribut fs_locations_info afin de spécifier quelles portions des informations fournies s'appliquent à un chemin d'accès réseau spécifique et à la réplique que le chemin utilise pour y accéder.

De plus, d'autres sections de la [RFC5661] ont été mises à jour pour corriger les conséquences des hypothèses sous-jacentes incorrectes sur le traitement des questions d'espace de noms de multi serveurs. Elles sont décrites dans les appendices B.2 à B.4.

* Une section introductive révisée concernant les facilités d'espace de noms multi-serveurs est fournie.

* Un traitement plus réaliste de la portée de serveur est fourni. Ce traitement reflète la coordination plus limitée de l'état de verrouillage adoptée par les serveurs qui partagent en fait une portée de serveur commune.

* Du texte confus concernant les changements de propriétaire de serveur a été précisé.

* La description de certaines erreurs existantes a été modifiée pour expliquer plus clairement certaines situations d'erreur pour refléter l'existence de multiplexage et la possible utilisation de périodes de grâce spécifiques de système de fichiers. Pour les détails, voir l'Appendice B.3.

* De nouvelles descriptions de certaines opérations existantes sont fournies, soit parce que le traitement existant ne tenait pas compte de situations qui apparaîtraient en traitant la migration d'état transparente, soit parce que certains types de problèmes de réclamation n'était pas traités de façon adéquate dans le contexte de périodes de grâce spécifiques de système de fichiers. Pour les détails, voir l'Appendice B.2.

Appendice B. Changements apportés par cette mise à jour

B.1 Révisions à la Section 11 de la RFC 5661

Un certain nombre de domaines ont été révisés ou étendus, dans de nombreux cas en remplaçant un paragraphe au sein de la Section 11 de la [RFC5661] :

* Le nouveau matériel introductif, incluant une section de terminologie, remplace le matériel de la [RFC5661], allant du début de la Section 11 d'origine jusqu'à et inclus le paragraphe 11.1. Le nouveau matériel commence au début de la Section 11 et continues jusqu'à 11.2.

* Une réorganisation significative du matériel des paragraphes 11.4 et 11.5 de la [RFC5661] était nécessaire. Les raisons de la réorganisation de ces paragraphes en un seul paragraphe avec plusieurs sous paragraphes sont discutées dans l'Appendice B.1.1. Ce remplacement apparaît dans le paragraphe 11.5.

Du nouveau matériel relatif au traitement des attributs de localisation de système de fichiers est contenu dans les paragraphes 11.5.1 et 11.5.7.

* Un nouveau paragraphe décrivant les exigences pour le traitement d'utilisateur et de groupe au sein d'un espace de noms multi serveurs a été ajouté au paragraphe 11.7.

* Un remplacement majeur pour le paragraphe 11.7 de la [RFC5661], intitulé "Effectuer les transitions de système de fichiers", apparaît aux paragraphes 11.9 à 11.14. Les raisons de la réorganisation de cette section en plusieurs paragraphes sont discutées à l'Appendice B.1.2.

* Un remplacement du paragraphe 11.10 de la [RFC5661], intitulé "Attribut fs_locations_info", apparaît au paragraphe 11.17, l'Appendice B.1.3 décrivant les différences entre le nouveau paragraphe et le traitement dans la [RFC5661]. Un traitement révisé était nécessaire parce que le traitement original ne montrait pas clairement comment les informations d'attribut ajoutées se rapportent au cas de chemins tronqués vers la même réplique. Ces questions n'étaient pas traitées dans la [RFC5661] où les concepts de réplique et de chemin de réseau utilisé pour accéder à une réplique n'étaient pas clairement distingués.

B.1.1 Réorganisation des paragraphes 11.4 et 11.5 de la RFC 5661

Précédemment, les questions relatives au fait que plusieurs entrées de localisation dirigeaient le client sur la même instance de système de fichiers étaient traitées au paragraphe 11.4 de la [RFC5661]. À cause du nouveau traitement de multiplexage, ces questions sont maintenant au paragraphe 11.5.

Dans cette nouvelle section, le multiplexage est couvert au paragraphe 11.5.2 avec l'autre utilisation des informations de localisation de système de fichiers décrites aux paragraphes 11.5.3 à 11.5.6.

Par suite, le paragraphe 11.5, qui remplace le paragraphe 11.4 de la [RFC5661], est substantiellement différent du paragraphe qu'il remplace en ce que des paragraphes originaux ont été remplacés par les paragraphes correspondants comme décrit ci-dessous, tandis que de nouveaux paragraphes ont été ajoutés :

* le matériel du paragraphe 11.5, sans sous paragraphes, remplace celui du paragraphe 11.4 de la [RFC5661] sans sous paragraphes ;
* le paragraphe 11.5.1 est le nouveau premier sous paragraphe de la section globale ;
* le paragraphe 11.5.2 est le nouveau second sous paragraphe de la section globale ;
* chacun des paragraphes 11.5.4, 11.5.5, et 11.5.6 remplace (dans l'ordre) un des paragraphes 11.4.1, 11.4.2, et 11.4.3 de la [RFC5661] ;
* le paragraphe 11.5.7 est le nouveau sous paragraphe final de la section globale.

B.1.2 Réorganisation du matériel traitant des transitions de système de fichiers

Le matériel relatif à la transition de système de fichiers, précédemment contenu au paragraphe 11.7 de la [RFC5661] a été réorganisé et augmenté comme décrit ci-dessous :

* Parce que il peut y avoir un glissement des chemins d'accès réseau utilisés pour accéder à une instance de système de fichiers sans aucun glissement entre les répliques, un nouveau paragraphe 11.9 distingue les cas dans lesquels il y a un glissement entre des répliques distinctes et ceux impliquant un glissement des chemins d'accès réseau sans glissement entre les répliques.

  Par suite, le nouveau paragraphe 11.10 traite des transitions d'adresse réseau, tandis que le gros du paragraphe 11.7 original de la [RFC5661] a été largement modifié comme reflété au paragraphe 11.11, qui est maintenant limité aux cas dans lesquels il y a un glissement entre deux ensembles différents de répliques.

* Le paragraphe additionnel 11.12 discute du cas où un glissement sur une réplique différente est fait et où l'état est transféré pour permettre au client d'avoir un accès continuel à son état de verrouillage accumulé sur le nouveau serveur.

* Le paragraphe additionnel 11.13 discute de la réponse du client pour accéder aux transitions, comment il détermine si une migration s'est produite, et comment il obtient l'accès à l'état de verrouillage et de session transféré.

* Le paragraphe additionnel 11.14 discute des responsabilités des serveurs de source et destination quand ils transfèrent l'état de verrouillage et de session.

Cette réorganisation a causé un renumérotage des paragraphes au sein de la Section 11 de la [RFC5661] comme décrit ci-dessous :

* Les nouveaux paragraphes 11.9 et 11.10 ont résulté en le renumérotage des paragraphes existants avec ces numéros.
* Le paragraphe 11.7 de la [RFC5661] a été substantiellement modifié et apparaît comme paragraphe 11.11. Les modifications nécessaires reflètent le fait que ce paragraphe traite seulement des transitions entre les répliques, tandis que les transitions entre adresses réseau sont traitées dans d'autres paragraphes. Les détails de cette réorganisation sont décrits plus loin.
* Les paragraphes 11.12, 11.13, et 11.14 ont été ajoutés.

* Par conséquent, les paragraphes 11.8, 11.9, 11.10, et 11.11 de la [RFC5661] apparaissent maintenant respectivement comme paragraphes 11.15, 11.16, 11.17, et 11.18.

Au titre de cette réorganisation générale, le paragraphe 11.7 de la [RFC5661] a été modifié comme décrit ci-dessous :
* Les paragraphes 11.7 et 11.7.1 de la [RFC5661] ont été remplacés respectivement par les paragraphes 11.11 et 11.11.1.
* Le paragraphe 11.7.2 de la RFC 5661 (et les sous paragraphes inclus) a été supprimé.
* Les paragraphes 11.7.3, 11.7.4, 11.7.5, 11.7.5.1, et 11.7.6 de la [RFC5661] ont été remplacés respectivement par les paragraphes 11.11.2, 11.11.3, 11.11.4, 11.11.4.1, et 11.11.5 de ce document.
* Le paragraphe 11.7.7 de la [RFC5661] a été remplacé par le paragraphe 11.11.9. Ce sous paragraphe a été déplacé à la fin de la section qui traite des transitions de système de fichiers.
* Les paragraphes 11.7.8, 11.7.9, et 11.7.10 de la [RFC5661] ont été remplacés respectivement par les paragraphes 11.11.6, 11.11.7, et 11.11.8 de ce document.

B.1.3 Mise à jour du traitement de fs_locations_info

Divers éléments de l'attribut fs_locations_info contiennent des information qui s'appliquent à une réplique de système de fichiers spécifique ou à un chemin de réseau ou ensemble de chemins de réseau utilisé pour accéder à une telle réplique. Le traitement original de fs_locations_info (paragraphe 11.10 de la [RFC5661]) ne distinguait pas clairement ces cas, en partie parce que le document ne distinguait pas clairement les répliques des chemins utilisés pour y accéder.

De plus, des éclaircissement particuliers ont été fournis à l'égard des champs suivants :
* traitement de FSLI4GF_GOING : il a été précisé que cela s'applique seulement à l'indisponibilité d'une réplique plutôt qu'au chemin pour accéder à une réplique ;
* pour décrire la valeur appropriée qu'un serveur utilise pour fli_valid_for, il a été précisé qu'il n'est pas besoin que le client aille fréquemment chercher la valeur fs_locations_info pour être prêt aux glissements dans les schémas de multiplexage ;
* des précisions aux règles d'extensions de fls_info ont été fournies. Le traitement original reflétait le modèle d'extension qui avait cours au mement de la rédaction de la [RFC5661], mais a été mis à jour en accord avec le modèle d'extension décrit dans la [RFC8178].

B.2 Révisions des opérations de la RFC 5661

Des descriptions ont été révisées pour traiter des problèmes qui sont apparus en effectuant les changements nécessaires des caractéristiques d'espace de noms multi-serveurs.

* Le traitement de EXCHANGE_ID (paragraphe 18.35 de la [RFC5661]) supposait que les identifiants de client ne peuvent pas être créés/confirmés autrement que par les opérations EXCHANGE_ID et CREATE_SESSION. Aussi, l'utilisation nécessaire de EXCHANGE_ID dans la récupération de la migration et les situations qui s'y rapportent n'était pas traité de façon claire. Un traitement révisé de EXCHANGE_ID était nécessaire, et il apparaît au paragraphe 18.35, tandis que les différences spécifiques entre lui et le traitement dans la [RFC5661] sont expliquées à l'Appendice B.2.1.

* Le traitement de RECLAIM_COMPLETE au paragraphe 18.51 de la [RFC5661] n'était pas suffisamment clair sur l'objet et l'utilisation de rca_one_fs et sur comment le serveur allait traiter des valeurs inappropriées de cet argument. Parce que la confusion résultante soulevait des problèmes d'interopérabilité, un nouveau traitement de RECLAIM_COMPLETE était nécessaire, et il apparaît au paragraphe 18.51, tandis que les différences spécifiques entre lui et le traitement de la [RFC5661] sont discutées à l'Appendice B.2.2. De plus, les définitions des erreurs relatives à la réclamation ont reçu un traitement mis à jour au paragraphe 15.1.9 pour refléter le fait qu'il y a plusieurs contextes pour les opérations de réclamation de verrous.

B.2.1 Révision du traitement de EXCHANGE_ID

Il y avait un certain nombre de problèmes dans le traitement original de EXCHANGE_ID dans la [RFC5661] qui causaient des problèmes pour la migration d'état transparente et pour le transfert d'accès entre différents chemins d'accès réseau pour la même instance de système de fichiers.
Ces problèmes survenaient du fait que ce traitement était écrit :
* en supposant qu'un identifiant de client peut seulement devenir connu d'un serveur en ayant été créé par l'exécution d'un EXCHANGE_ID, avec confirmation de l'identifiant seulement possible par l'exécution de CREATE_SESSION ;
* en considérant les interactions entre un client et un serveur se produisant seulement sur une seule adresse réseau.

Comme ces hypothèses sont devenues invalides dans le contexte de migration d'état transparente et de l'utilisation active du multiplexage, le traitement a été modifié à plusieurs égards :

* Il avait été supposé qu'un EXCHANGE_ID exécuté quand le serveur savait déjà qu'une certaine instance de client mettait à jour ses paramètres associés (par exemple, par rapport aux rappels) ou traitait une réponse précédemment perdue en la retransmettant. Par suite, toute séquence de créneaux retournée par cette opération serait inutile. Le traitement original allait jusqu'à dire que il "NE DOIT PAS" être utilisé, bien que cet usage ne soit pas en accord avec la [RFC2119]. Ceci créait une difficulté quand un EXCHANGE_ID est fait après une migration d'état transparente car cette séquence de créneaux devrait être utilisée dans une opération CREATE_SESSION suivante.

  Dans le traitement mis à jour, CREATE_SESSION est une façon de confirmer les identifiants de client, mais il est entendu que d'autres façons sont possibles. La séquence de créneaux peut être utilisée comme nécessaire, et les cas dans lesquels elle ne sera d'aucune utilisé sont notés de façon appropriée.

* Il avait été supposé que les seules fonctions de EXCHANGE_ID étaient d'informer le serveur du client, de créer l'identifiant de client, et de le communiquer au client. Quand plusieurs connexions simultanées sont impliquées, comme il arrive souvent quand il y a multiplexage, ce traitement était inadéquat en ce qu'il ignorait le rôle de EXCHANGE_ID pour associer l'identifiant de client à la connexion sur laquelle il est fait, afin qu'il puisse être utilisé par une CREATE_SESSION suivante dont les paramètres n'incluraient pas un identifiant de client explicite.

Le nouveau traitement discute explicitement le rôle de EXCHANGE_ID en associant l'identifiant de client à la connexion afin qu'il puisse être utilisé par CREATE_SESSION et en associant une connexion à une session existante.

Le nouveau traitement se trouve au paragraphe 18.35. Il remplace le traitement du paragraphe 18.35 de la [RFC5661].

B.2.2 Révision du traitement de RECLAIM_COMPLETE

Les changements suivants ont été faits au traitement de RECLAIM_COMPLETE de la [RFC5661] pour arriver au traitement du paragraphe 18.51 :

* En plusieurs endroits, le texte a été rendu plus explicite sur l'objet de rca_one_fs et sa connexion à la migration de système de fichiers.
* Il est discuté de situations dans lesquelles des formes particulières de RECLAIM_COMPLETE devraient être faites.
* Il y a une discussion des questions d'interopérabilité entre les mises en œuvre qui peut survenir à cause du manque de clarté du précédent traitement de RECLAIM_COMPLETE.

B.3 Révisions aux définitions d'erreur dans la RFC 5661

Le nouveau traitement de diverses situations exigeait la révision de certaines définitions d'erreur :

* À cause du besoin de traiter de façon appropriée les questions relatives au multiplexage, certaines utilisations du terme "réplique" dans la [RFC5661] sont devenues problématiques parce que un glissement dans les chemins d'accès réseau était considéré être un glissement à une réplique différente. Par suite, la définition originale de NFS4ERR_MOVED (au paragraphe 15.1.2.4 de la [RFC5661]) a été mise à jour pour refléter le traitement différent de l'indisponibilité d'un système de fichiers particulier via une adresse réseau spécifique.

  Comme une telle situation n'est plus considérée constituer une indisponibilité d'une instance de système de fichiers, la description a été changée, même si l'ensemble de circonstances dans lesquelles elle va être retournée reste le même. Le nouveau paragraphe reconnaît explicitement qu'une adresse réseau différente pourrait être utilisée, tandis que la description précédente traitait à tort cela comme un glissement entre deux répliques alors que seulement une instance de système de fichiers pouvait être impliquée. La description mise à jour apparaît au paragraphe 15.1.2.4.

* Pour s'accommoder de l'utilisation de périodes de grâce spécifiques de système de fichiers, il a été nécessaire de préciser certaines des définitions d'erreurs relatives à la réclamation à la Section 15 de la [RFC5661] afin que le texte s'applique correctement aux réclamations pour tous les types de périodes de grâce. La mise à jour des descriptions apparaît au paragraphe 15.1.9.

* Pour fournir les éclaircissements sur le rapport d'erreur 2006 [ERR-2006] et pour les adapter à une explication appropriée de l'interaction de NFS4ERR_DELAY avec la mémoire tampon de réponse, une description révisée de NFS4ERR_DELAY apparaît au paragraphe 15.1.1.3. Ce rapport d'erreur, à la différence de beaucoup d'autres rapports

d'erreur de la RFC 5661, est traité dans le pésent document à cause de l'utilisation extensive de NFS4ERR_DELAY en connexion avec la migration d'état et la migration de session.

B.4 Autres révisions à la RFC 5661

À côté de la reformulation majeure de la Section 11 de la [RFC5661] et des révisions associées aux opérations et erreurs existantes, un certain nombre de changements en rapport étaient nécessaires :

* Le résumé du paragraphe 1.7.3.3 de la [RFC5661] a été révisé pour refléter les changements faits à la Section 11. Le sommaire mis à jour apparaît au paragraphe 1.8.3.3.

La discussion de portée de serveur au paragraphe 2.10.4 de la [RFC5661] a été remplacé car elle paraissait exiger un niveau de coordination inter-serveurs incompatible avec sa fonction de base d'éviter un moyen uniforme global d'allouer des valeurs de propriétaire de serveur. Un traitement révisé apparaît au paragraphe 2.10.4.

* La discussion du multiplexage au paragraphe 2.10.5 de la [RFC5661] a été révisé pour expliquer plus clairement la prise en charge de plusieurs types de multiplexage et comment le client peut être informé de la configuration existante de multiplexage. De plus, alors que le dernier paragraphe (sans les sous paragraphes) de cette section traitant des changements de server_owner était littéralement vrai, il a été une source de confusion. Comme le paragraphe original pouvait être lu comme suggérant que de tels changements soient traités de façon non interruptive, la question a été clarifiée dans le paragraphe 2.10.5 révisé.

Appendice C. Questions de sécurité qui doivent être traitées

Les questions suivantes du traitement de sécurité dans la spécification NFSv4.1 doivent être examinées :

* La section des considérations sur la sécurité de la [RFC5661] n'était pas conforme au BCP 72 [RFC3552]. Le souci principal est le fait que cette section ne contenait pas d'analyse des menaces.

* L'analyse initiale des questions de sécurité existantes avec NFSv4.1 a rendu probable qu'une section révisée des considérations sur la sécurité pour le protocole existant (contenant une analyse des menaces) conclurait que NFSv4.1 ne respecte pas le but d'une utilisation sûre sur l'Internet.

La section des considérations sur la sécurité du présent document (Section 21) n'a pas fait l'objet d'une révision stricte pour corriger les difficultés mentionnées ci-dessus. À la place, elle a été modifiée pour tenir un compte approprié des questions relatives aux caractéristiques d'espace de noms multi-serveurs discutées à la Section 11, laissant la discussion incomplète et les faiblesses de sécurité en l'état.

Les questions de sécurité majeures suivantes doivent être traitées de façon satisfaisante avant qu'une section de considérations de sécurité mise à jour puisse être publiée au titre d'un document bis pour NFSv4.1 :

* La poursuite de l'utilisation de AUTH_SYS et les expositions pour la sécurité qu'elle crée doivent être considérées. Le traitement de cette question ne doit pas être limité à celle de si la désignation de cela comme FACULTATIF était justifié ou si cela devrait être changé. En tous cas, il peut n'être pas possible à ce point de corriger les problèmes de sécurité créés par la poursuite de l'utilisation de AUTH_SYS simplement en révisant cette désignation.

* L'absence d'attention dans le protocole à la possibilité d'attaque de surveillance invasive comme celles décrites dans la [RFC7258] (aussi BCP 188). Dans ce cas, l'utilisation de CREATE_SESSION sans protection de la confidentialité doit être examinée car elle expose l'identifiant de session à la vue d'un attaquant. C'est ennuyeux car c'est précisément le type de caractéristique de protocole auquel il est fait allusion dans la RFC 7258, qui peut permettre plus de dommages de la part de l'attaquant car cela permet des attaques de déni de service qui peuvent être exécutées effectivement avec seulement un seul accréditif, normalement de faible valeur, même quand l'authentification RPCSEC_GSS est utilisée.

* L'absence d'utilisation effective de la confidentialité et de l'intégrité, même lorsque l'infrastructure pour prendre en charge l'utilisation de RPCSEC_GSS est présente, doit être examinée. À la lumière des expositions de la sécurité que crée cette situation, il ne suffit pas de définir un protocole qui pourrait régler ce problème avec la fourniture de ressources suffisantes. À la place, ce qui est nécessaire est un moyen de fournir la sécurité nécessaire avec des coûts de performances très limités et sans exiger d'infrastructure de sécurité, ce que l'expérience a montré qu'il est difficile à

fournir pour de nombreux clients et serveurs.

En essayant de fournir une mise à niveau majeure de la sécurité pour un protocole déployé comme NFSv4.1, le groupe de travail et la communauté de l'Internet vont probablement se trouver confrontés à un certain nombre de considérations telles que les suivantes :

* Le besoin de s'accommoder des déploiements existants de protocoles spécifiés précédemment dans les normes proposées existantes.
* La difficulté d'effectuer des changements aux mises en œuvre interopérantes existantes.
* La difficulté de faire des changements aux protocoles NFSv4 autres que ceux sous la forme d'extensions FACULTATIVES.
* La tendance des responsables des déploiements NFSv4 existants à ignorer les fautes de sécurité dans le contexte des réseaux de zone locale sous l'impression erronée que l'isolation de réseau fournit, en et par elle-même, l'isolation de tous les attaquants potentiels.

Étant donné que les difficultés sus-mentionnées s'appliquent aussi à la version mineure zéro, il peut y avoir un sens à traiter ces questions de sécurité dans un document commun qui s'applique à toutes les versions mineures NFSv4. Si cette approche est suivie, la section des considérations sur la sécurité d'un éventuel document NFv4.1 bis ferait référence à ce document commun, et les RFC de définition d'autres versions mineures pourraient le faitre aussi.

Remerciements

Remerciements pour la présente mise à jour

Les auteurs souhaitent reconnaître le rôle important de Andy Adamson de Netapp pour préciser le besoin de la fonction de découverte de multiplexage, et explorer le rôle des attributs de localisation de système de fichiers en fournissant le soutien nécessaire.

Les auteurs souhaitent remercier Tom Haynes de Hammerspace d'avoir attiré notre attention sur le fait que internationalisation et sécurité pourraient mieux être traitées dans des documents traitant de ces questions de protocole car elles s'appliquent à toutes les versions mineures de NFSv4.

Les auteurs souhaitent aussi reconnaître le travail de Xuan Qi de Oracle sur les prototypes de fonction de migration d'état transparente de client et serveur NFSv4.1.

Les auteurs souhaitent remercier les autres personnes qui ont attiré notre attention sur des questions importantes. Les commentaires de Trond Myklebust de Primary Data relatifs au multiplexage ont aidé à préciser le rôle du DNS dans la découverte de multiplexage. Les commentaires de Rick Macklem ont éttiré notre attention sur les problèmes du traitement de la version par système de fichiers de RECLAIM_COMPLETE.

Les auteurs souhaitent remercier Olga Kornievskaia de Netapp de ses utiles commentaires de relecture.

Remerciements pour la RFC 5661

Le texte initial pour les extensions de SECINFO était édité par Mike Eisler avec des contributions de Peng Dai, Sergey Klyushin, et Carl Burnett.

Le texte initial pour les extensions de SESSIONS était édité par Tom Talpey, Spencer Shepler, Jon Bauman avec des contributions de Charles Antonelli, Brent Callaghan, Mike Eisler, John Howard, Chet Juszczak, Trond Myklebust, Dave Noveck, John Scott, Mike Stolarchuk, et Mark Wittle.

Le texte initial relatif aux caractéristiques d'espace de noms multi-serveurs, incluant le concept de référents, a été apporté par Dave Noveck, Carl Burnett, et Charles Fan avec des contributions de Ted Anderson, Neil Brown, et Jon Haswell.

Le texte initial pour la prise en charge des délégations de répertoire a été apporté par Saadia Khan avec des apports de Dave Noveck, Mike Eisler, Carl Burnett, Ted Anderson, et Tom Talpey.

Le texte initial pour les explications d'ACL a été apporté par Sam Falkner et Lisa Week.

Le travail sur pNFS était inspiré par celui sur NASD et OSD fait par Garth Gibson. Gary Grider a aussi été un champion

des entrées/sorties parallèles à hautes performances. Garth Gibson et Peter Corbett ont commencé l'effort pNFS avec un document de déclaration de problème pour l'IETF qui a été la base du travail sur pNFS dans NFSv4.1.

Le texte initial pour la prise en charge de NFS parallèle a été édité par Brent Welch et Garth Goodson. Les auteurs supplémentaires pour ces documents étaient Benny Halevy, David Black, et Andy Adamson. Des apports supplémentaires sont venus du groupe informel qui a contribué à la construction des projets initiaux de pNFS ; des remerciements spécifiques sont dus à Gary Grider, Peter Corbett, Dave Noveck, Peter Honeyman, et Stephen Fridella.

Fredric Isaman a trouvé plusieurs erreurs dans les projets de versions de description ONC RPC XDR du protocole NFSv4.1.

Audrey Van Belleghem a fourni, de nombreuses façons, la coordination et la gestion essentielle du processus d'édition des documents de spécification.

Richard Jernigan a fourni des retours sur la conception du schéma de mise en bandes de la disposition de fichier.

Plusieurs équipes formelles d'inspection ont été formées pour revoir les diverses zones du protocole. Toutes les inspections ont trouvé des erreurs significatives et des améliorations possibles. Les équipes d'inspection de NFSv4.1 étaient :

* Les ACL, avec les inspecteurs suivants : Sam Falkner, Bruce Fields, Rahul Iyer, Saadia Khan, Dave Noveck, Lisa Week, Mario Wurzl, et Alan Yoder.

* Les sessions, avec les inspecteurs suivants : William Brown, Tom Doeppner, Robert Gordon, Benny Halevy, Fredric Isaman, Rick Macklem, Trond Myklebust, Dave Noveck, Karen Rochford, John Scott, et Peter Shah.

* Inspection pNFS initiale, avec les inspecteurs suivants : Andy Adamson, David Black, Mike Eisler, Marc Eshel, Sam Falkner, Garth Goodson, Benny Halevy, Rahul Iyer, Trond Myklebust, Spencer Shepler, et Lisa Week.

* Espace de noms global, avec les inspecteurs suivants : Mike Eisler, Dan Ellard, Craig Everhart, Fredric Isaman, Trond Myklebust, Dave Noveck, Theresa Raj, Spencer Shepler, Renu Tewari, et Robert Thurlow.

* Type de disposition de fichier NFSv4.1, avec les inspecteurs suivants : Andy Adamson, Marc Eshel, Sam Falkner, Garth Goodson, Rahul Iyer, Trond Myklebust, et Lisa Week.

* Verrouillage et délégations de répertoire NFSv4.1, avec les inspecteurs suivants : Mike Eisler, Pranoop Erasani, Robert Gordon, Saadia Khan, Eric Kustarz, Dave Noveck, Spencer Shepler, et Amy Weaver.

* EXCHANGE_ID et DESTROY_CLIENTID, avec les inspecteurs suivants : Mike Eisler, Pranoop Erasani, Robert Gordon, Benny Halevy, Fredric Isaman, Saadia Khan, Ricardo Labiaga, Rick Macklem, Trond Myklebust, Spencer Shepler, et Brent Welch.

* Inspection finale de pNFS, avec les inspecteurs suivants : Andy Adamson, Mike Eisler, Mark Eshel, Sam Falkner, Jason Glasgow, Garth Goodson, Robert Gordon, Benny Halevy, Dean Hildebrand, Rahul Iyer, Suchit Kaura, Trond Myklebust, Anatoly Pinchuk, Spencer Shepler, Renu Tewari, Lisa Week, et Brent Welch.

Une équipe de révision a travaillé pour générer les tableaux d'allocations des ensembles d'erreurs des opérations et s'assurer que chaque allocation avait au moins deux personnes pour les valider. Ont participé au processus Andy Adamson, Mike Eisler, Sam Falkner, Garth Goodson, Robert Gordon, Trond Myklebust, Dave Noveck, Spencer Shepler, Tom Talpey, Amy Weaver, et Lisa Week.

Jari Arkko, David Black, Scott Bradner, Lisa Dusseault, Lars Eggert, Chris Newman, et Tim Polk ont fourni de précieuses relectures et conseils.

Olga Kornievskaia a trouvé plusieurs erreurs dans la spécification de SSV.

Ricardo Labiaga a trouvé plusieurs endrois où l'utilisation de RPCSEC_GSS était sous spécifiée.

Ceux qui ont fourni divers commentaires sont : Andy Adamson, Sunil Bhargo, Alex Burlyga, Pranoop Erasani, Bruce Fields, Vadim Finkelstein, Jason Goldschmidt, Vijay K. Gurbani, Sergey Klyushin, Ricardo Labiaga, James Lentini, Anshul Madan, Daniel Muntz, Daniel Picken, Archana Ramani, Jim Rees, Mahesh Siddheshwar, Tom Talpey, et Peter Varga.

Adresse des auteurs

David Noveck (éditeur)
NetApp
1601 Trapelo Road, Suite 16
Waltham, MA 02451
USA
téléphone : +1-781-768-5347
mél : dnoveck@netapp.com

Charles Lever
Oracle Corporation
1015 Granger Avenue
Ann Arbor, MI 48104
USA
téléphone : +1-248-614-5091
mél : chuck.lever@oracle.com