Network Working Group
Internet-Draft
Document:
draft-ietf-nfsv4-directory-delegation-01.txt
March 2005

S. Khan
Author
Network Appliance, Inc.

NFSv4.1: Directory Delegations and Notifications

Status of this Memo

ABSTRACT

This document proposes adding directory delegations and notifications to NFS Version 4 [RFC3530]. It is hoped that these changes will be part of a new minor version of NFS, such as NFSv4.1.

TABLE OF CONTENTS

1. Introduction

This document assumes understanding of the NFSv4.0 specification. It also assumes that the changes proposed by [talpey] will be present in the same minor version or certain incremental additions to this proposal will be required, as discussed later in the document.

The major addition to NFS version 4 in the area of caching is the ability of the server to delegate certain responsibilities to the client. When the server grants a delegation for a file to a client, the client receives certain semantics with respect to the sharing of that file with other clients. At OPEN, the server may provide the client either a read or write delegation for the file. If the client is granted a read delegation, it is assured that no other client has the ability to write to the file for the duration of the delegation. If the client is granted a write delegation, the client is assured that no other client has read or write access to the file. This reduces network traffic and server load by allowing the client to perform certain operations on local file data and can also provide stronger consistency for the local data.

Directory caching for the NFS version 4 protocol is similar to previous versions. Clients typically cache directory information for a duration determined by the client. At the end of a predefined timeout, the client will query the server to see if the directory has been updated. By caching attributes, clients reduce the number of GETATTR calls made to the server to validate attributes. Furthermore, frequently accessed files and directories, such as the current working directory, have their attributes cached on the client so that some NFS operations can be performed without having to make an RPC call. By caching name and inode information about most recently looked up entries in DNLC (Directory Name Lookup Cache), clients do not need to send LOOKUP calls to the server every time these files are accessed.

This caching approach works reasonably well at reducing network traffic in many environments. However, it does not address environments where there are numerous queries for files that do not exist. In these cases of "misses", the client must make RPC calls to the server in order to provide reasonable application semantics and promptly detect the creation of new directory entries. Examples of high miss activity are compilation in software development environments. The current behavior of NFS limits its potential scalability and wide-area sharing effectiveness in these types of environments. Other distributed stateful filesystem architectures such as AFS and DFS have proven that adding state around directory contents can greatly reduce network traffic in high miss environments.

Delegation of directory contents is proposed as an extension for NFSv4. Such an extension would provide similar traffic reduction benefits as with file delegations. By allowing clients to cache directory contents (in a read-only fashion) while being notified of changes, the client can avoid making frequent requests to interrogate the contents of slowly-changing directories, reducing network traffic and improving client performance.

These extensions allow improved namespace cache consistency to be achieved through delegations and synchronous recalls alone without asking for notifications. In addition, if time-based consistency is sufficient, asynchronous notifications can provide performance benefits for the client, and possibly the server, under some common operating conditions such as slowly-changing and/or very large directories.

2. Proposed protocol extensions.

This document includes the definition of protocol extensions to implement directory delegations. It is believed that these extension fit within the minor-versioning framework presented in RFC3530.

The NFSv4 Sessions Extensions [talpey] include a new operation (called SEQUENCE) in each COMPOUND procedure which carries the clientid associated with the session to which the procedure belongs. In NFSv4.0, only certain COMPOUND procedures may carry such a clientid. When present, this clientid provides all the necessary context for maintaining directory delegations, and dispatching appropriate callbacks.

If the directory delegation protocol described here is not able to leverage any pre-existing clientid present in each COMPOUND request, then the equivalent clientid must be provided where necessary. This could be accomplished by simply including the SEQUENCE operation in each compound of the new minor version, regardless of the status of any session.

Mainly in the interests of clarity of presentation, elements within these extensions are assigned numeric identifiers such as operation numbers and attribute identifiers. It should be understood that when these extensions are included in a minor version of NFSv4, the actual numeric identifiers assigned may be different from the ones chosen here.

3. **Design**

A new operation GET_DIR_DELEGATION is used by the client to ask for a directory delegation. The delegation covers directory attributes and all entries in the directory. If either of these change the delegation will be recalled synchronously. The operation causing the recall will have to wait before the recall is complete. Any changes to directory entry attributes will not cause the delegation to be recalled.

In addition to asking for delegations, a client can also ask for notifications for certain events. These events include changes to directory attributes and/or its contents. If a client asks for notification for a certain event, the server will notify the client when that event occurs. This will not result in the delegation being recalled for that client. The notifications are asynchronous and provide a way of avoiding recalls in situations where a directory is changing enough that the pure recall model may not be effective while trying to allow the client to get substantial benefit. In the absence of notifications, once the delegation is recalled the client has to refresh its directory cache which might not be very efficient for very large directories.

The delegation is read only and the client may not make changes to the directory other than by performing NFSv4 operations that modify the directory or the associated file attributes so that the server has knowledge of these changes. In order to keep the client namespace in sync with the server, the server will notify the client holding the delegation of the changes made as a result. This is to avoid any subsequent GETATTR or READDIR calls to the server. If a client holding the delegation makes any changes to the directory, the delegation will not be recalled.

Delegations can be recalled by the server at any time. Normally, the server will recall the delegation when the directory changes in a way that is not covered by the notification, or when the directory changes and notifications have not been requested.

Also if the server notices that handing out a delegation for a directory is causing too many notifications to be sent out, it may decide not to hand out a delegation for that directory or recall existing delegations. If another client removes the directory for which a delegation has been granted, the server will recall the delegation.

Both the notification and recall operations need a callback path to exist between the client and server. If the callback path does not exist, then delegation can not be granted. Note that with the session extensions [talpey] that should not be an issue. In the absense of sessions, the server will have to establish a callback path to the client to send callbacks.

4. New Operation 40: GET_DIR_DELEGATION - Get a directory delegation

```
   SYNOPSIS
        (cfh), requested notification -> (cfh), cookieverf, stateid,
        supported notification

   ARGUMENT
        struct GET_DIR_DELEGATION4args {
             dir_notification_type4      notification_type;
             attr_notice4                child_attr_delay;
             attr_notice4                dir_attr_delay;
        };

        /*
         * Notification types.
         */
        const DIR_NOTIFICATION_NONE                    = 0x00000000;
        const DIR_NOTIFICATION_CHANGE_CHILD_ATTRIBUTES  = 0x00000001;
        const DIR_NOTIFICATION_CHANGE_DIR_ATTRIBUTES   = 0x00000002;
        const DIR_NOTIFICATION_REMOVE_ENTRY            = 0x00000004;
        const DIR_NOTIFICATION_ADD_ENTRY               = 0x00000008;
        const DIR_NOTIFICATION_RENAME_ENTRY            = 0x00000010;
        const DIR_NOTIFICATION_CHANGE_COOKIE_VERIFIER  = 0x00000020;

        typedef uint32_t dir_notification_type4;

        typedef nfstime4 attr_notice4;


   RESULT
        struct GET_DIR_DELEGATION4resok {
                verifier4                       cookieverf;
               /* Stateid for get_dir_delegation */
                stateid4                        stateid;
               /* Which notifications can the server support */
               dir_notification_type4          supp_notification;
```

```
                bitmap4                         child_attributes;
                bitmap4                         dir_attributes;
        };

        union GET_DIR_DELEGATION4res switch (nfsstat4 status) {
        case NFS4_OK:
              /* CURRENT_FH: delegated dir */
               GET_DIR_DELEGATION4resok      resok4;
        default:
               void;
        };
```

DESCRIPTION:

The GET_DIR_DELEGATION operation is used by a client to request a directory delegation. The directory is represented by the current filehandle. The client also specifies whether it wants the server to notify it when the directory changes in certain ways by setting one or more bits in a bitmap. The server may also choose not to grant the delegation. In that case the server will return NFS4ERR_DIRDELEG_UNAVAIL. If the server decides to hand out the delegation, it will return a cookie verifier for that directory. If the cookie verifier changes when the client is holding the delegation, the delegation will be recalled unless the client has asked for notification for this event. In that case a notification will be sent to the client.

The server will also return a directory delegation stateid in addition to the cookie verifier as a result of the GET_DIR_DELEGATION operation. This stateid will appear in callback messages related to the delegation, such as notifications and delegation recalls. The client will use this stateid to return the delegation voluntarily or upon recall. A delegation is returned by calling the DELEGRETURN operation.

The server may not be able to support notifications of certain events. If the client asks for such notifications, the server must inform the client of its inability to do so as part of the GET_DIR_DELEGATION reply by not setting the appropriate bits in the supported notifications bitmask contained in the reply.

The GET_DIR_DELEGATION operation can be used for both normal and named attribute directories. It covers all the entries in the directory except the ".." entry. That means if a directory and its parent both hold directory delegations, any changes to the parent will not cause a notification to be sent for the child even though the child's ".." entry points to the parent.

IMPLEMENTATION:

Directory delegation provides the benefit of improving cache consistency of namespace information. This is done through synchronous callbacks. A server must support synchronous callbacks in order to support directory delegations. In addition to that, asynchronous notifications provide a way to reduce network traffic as well as improve client performance in certain conditions. Notifications would not be requested when the goal is just cache consitency.

Notifications are specified in terms of potential changes to the directory. A client can ask to be notified whenever an entry is added to a directory by setting notification_type to DIR_NOTIFICATION_ADD_ENTRY. It can also ask for notifications on entry removal, renames, directory attribute changes and cookie verifier changes by setting notification_type flag appropriately. In addition to that, the client can also ask for notifications upon attribute changes to children in the directory to keep its attribute cache up to date. However any changes made to child attributes do not cause the delegation to be recalled. If a client is interested in directory entry caching, or negative name caching, it can set the notification_type appropriately and the server will notify it of all changes that would otherwise invalidate its name cache. The kind of notification a client asks for may depend on the directory size, its rate of change and the applications being used to access that directory. However, the conditions under which a client might ask for a notification, is out of the scope of this specification.

The client will set one or more bits in a bitmap (notification_type) to let the server know what kind of notification(s) it is interested in. For attribute notifications it will set bits in another bitmap to indicate which attributes it wants to be notified of. If the server does not support notifications for changes to a certain attribute, it should not set that attribute in the supported attribute bitmap (supp_notification) specified in the reply.

In addition to that, the client will also let the server know if it wants to get the notification as soon as the attribute change occurs or after a certain delay by setting a delay factor, child_attr_delay for attribute changes to children and dir_attr_delay for attribute changes to the directory. If this delay factor is set to zero, that indicates to the server that the client wants to be notified of any attribute changes as soon as they occur. If the delay factor is set to N, the server will make a best effort guarantee that attribute updates are not out of sync by more than that. One value covers all attribute changes for the directory and another value covers all attribute changes for all children in the directory. If the client asks for a delay factor that the server does not support or that may cause significant resource consumption on the server by causing the server to send a lot of notifications, the server should not commit to sending out notifications for that attribute and therefore must not set the approprite bit in the child_attributes and dir_attributes bitmaps in the response.

The server will let the client know about which notifications it can support by setting appropriate bits in a bitmap. If it agrees to send attribute notifications, it will also set two attribute masks indicating which attributes it will send change notifications for. One of the masks covers changes in directory attributes and the other covers atttribute changes to any files in the directory.

The client should use a security flavor that the filesystem is exported with. If it uses a different flavor, the server should return NFS4ERR_WRONGSEC.

ERRORS

NFS4ERR_ACCESS
NFS4ERR_BADHANDLE
NFS4ERR_BADXDR
NFS4ERR_FHEXPIRED
NFS4ERR_INVAL
NFS4ERR_MOVED
NFS4ERR_NOFILEHANDLE
NFS4ERR_NOTDIR
NFS4ERR_RESOURCE
NFS4ERR_SERVERFAULT
NFS4ERR_STALE
NFS4ERR_DIRDELEG_UNAVAIL
NFS4ERR_WRONGSEC
NFS4ERR_EIO
NFS4ERR_NOTSUPP

5. New Recommended Attributes

#56 - supp_dir_attr_notice - notification delays on directory attributes
#57 - supp_child_attr_notice - notification delays on child attributes

DESCRIPTION:

These attributes allow the client and server to negotiate the frequency of notifications sent due to changes in attributes.

These attributes are returned as part of a GETATTR call on the directory. The supp_dir_attr_notice value covers all attribute changes to the directory and the supp_child_attr_notice covers all attribute changes to any child in the directory.

These attributes are per directory. The client needs to get these values by doing a GETATTR on the directory for which it wants notifications. However these attributes are only required when the client is interested in getting attribute notifications. For all other types of notifications and delegation requests without notifications, these attributes are not required.

When the client calls the GET_DIR_DELEGATION operation and asks for attribute change notifications, it will request a notification delay that is within the server's supported range. If the client violates what supp_attr_file_notice or supp_attr_dir_notice values are, the server should not commit to sending notifications for that change event.

A value of zero for these attributes means the server will send the notification as soon as the change occurs. It is not recommended to set this value to zero since that can put a lot of burden on the server. A value of N means that the server will make a best effort guarentee that attribute notification are not delayed by more than that. nfstime4 values that compute to negative values are illegal.

6. New Callback Operation: CB_NOTIFY - Notify directory changes

```
   SYNOPSIS
        stateid, notification -> {}

   ARGUMENT
        struct CB_NOTIFY4args {
                stateid4              stateid;
                dir_notification4     changes<>;
        };

        /*
         * Notification information sent to the client.
         */
        union dir_notification4
        switch (dir_notification_type4 notification_type) {
             case DIR_NOTIFICATION_CHANGE_CHILD_ATTRIBUTES:
                    dir_notification_attribute4 change_child_attributes;
             case DIR_NOTIFICATION_CHANGE_DIR_ATTRIBUTES:
                    fattr4                      change_dir_attributes;
             case DIR_NOTIFICATION_REMOVE_ENTRY:
```

```
                   dir_notification_remove4    remove_notification;
            case DIR_NOTIFICATION_ADD_ENTRY:
                   dir_notification_add4       add_notification;
            case DIR_NOTIFICATION_RENAME_ENTRY:
                   dir_notification_rename4    rename_notification;
            case DIR_NOTIFICATION_CHANGE_COOKIE_VERIFIER:
                   dir_notification_verifier4  verf_notification;
        };

        /*
         * Changed entry information.
         */
        struct dir_entry {
                component4      file;
                fattr4          attrs;
        };

        struct dir_notification_attribute4 {
                dir_entry    changed_entry;
        };

        struct dir_notification_remove4 {
               dir_entry      old_entry;
                nfs_cookie4    old_entry_cookie;
        };

        struct dir_notification_rename4 {
               dir_entry              old_entry;
               dir_notification_add4  new_entry;
        };

        struct dir_notification_verifier4 {
               verifier4       old_cookieverf;
               verifier4       new_cookieverf;
        };

        struct dir_notification_add4 {
               dir_entry       new_entry;
                /* what READDIR would have returned for this entry */
                nfs_cookie4     new_entry_cookie;
                bool            last_entry;
               prev_entry_info4     prev_info;
        };

        union prev_entry_info4 switch (bool isprev) {
        case TRUE:       /* A previous entry exists */
                prev_entry4 prev_entry_info;
        case FALSE:       /* we are adding to an empty
```

```
                    directory */
```

```
                void;
        };

        /*
         * Previous entry information
         */
        struct prev_entry4 {
                dir_entry       prev_entry;
                /* what READDIR returned for this entry */
                nfs_cookie4     prev_entry_cookie;
        };
```

RESULT

```
        struct CB_NOTIFY4res {
                nfsstat4        status;
        };
```

DESCRIPTION:

The CB_NOTIFY operation is used by the server to send notifications to clients about changes in a delegated directory. These notifications are sent over the callback path. The notification is sent once the original request has been processed on the server. The server will send an array of notifications for all changes that might have occurred in the directory. The dir_notification_type4 can only have one bit set for each notification in the array. If the client holding the delegation makes any changes in the directory that cause files or sub directories to be added or removed, the server will notify that client of the resulting change(s). If the client holding the delegation is making attribute or cookie verifier changes only, the server does not need to send notifications to that client. The server will send the following information for each operation:

- o ADDING A FILE: The server will send information about the new entry being created along with the cookie for that entry. The entry information contains the nfs name of the entry and attributes. If this entry is added to the end of the directory, the server will set a last_entry flag to true. If the file is added such that there is atleast one entry before it, the server will also return the previous entry information along with its cookie. This is to help clients find the right location in their DNLC or directory caches where this entry should be cached.

- o REMOVING A FILE: The server will send information about the directory entry being deleted. The server will also send the cookie value for the deleted entry so that clients can get to the cached information for this entry.

- RENAMING A FILE: The server will send information about both the old entry and the new entry. This includes name and attributes for each entry. This notification is only sent if both entries are in the same directory. If the rename is across directories, the server will send a remove notification to one directory and an add notification to the other directory, assuming both have a directory delegation.

- FILE/DIR ATTRIBUTE CHANGE: The client will use the attribute mask to inform the server of attributes for which it wants to receive notifications. This change notification can be requested for both changes to the attributes of the directory as well as changes to any file attributes in the directory by using two separate attribute masks. The client can not ask for change attribute notification per file. One attribute mask covers all the files in the directory. Upon any attribute change, the server will send back the values of changed attributes. Notifications might not make sense for some filesystem wide attributes and it is up to the server to decide which subset it wants to support. The client can negotiate the frequency of attribute notifications by letting the server know how often it wants to be notified of an attribute change. The server will return supported notification frequencies or an indication that no notification is permitted for directory or child attributes by setting the supp_dir_attr_notice and supp_child_attr_notice attributes respectively.

- COOKIE VERIFIER CHANGE: If the cookie verifier changes while a client is holding a delegation, the server will notify the client so that it can invalidate its cookies and reissue a READDIR to get the new set of cookies.

ERRORS

```
     NFS4ERR_BAD_STATEID
     NFS4ERR_INVAL
     NFS4ERR_BADXDR
     NFS4ERR_SERVERFAULT
```

7. Delegation Recall

The server will recall the directory delegation by sending a callback to the client. It will use the same callback procedure as used for recalling file delegations. The server will recall the delegation

when the directory changes in a way that is not covered by the notification. However the server will not recall the delegation if attributes of an entry within the directory change. Also if the server notices that handing out a delegation for a directory is causing too many notifications to be sent out, it may decide not to hand out a delegation for that directory. If another client tries to remove the directory for which a delegation has been granted, the server will recall the delegation.

The server will recall the delegation by sending a CB_RECALL callback to the client. If the recall is done because of a directory changing event, the request making that change will need to wait while the client returns the delegation.

8. New Callback Operation: CB_RECALL_ANY - Keep any N delegations

SYNOPSIS
```
     N -> {}
```

ARGUMENT
```
     struct CB_RECALLANYY4args {
             uint4          dlgs_to_keep;
     }
```

RESULT
```
     struct CB_RECALLANY4res {
            nfsstat4        status;
     };
```

DESCRIPTION:

The server may decide that it can not hold all the delegation state without running out of resources. Since the server has no knowledge of which delegations are being used more than others, it can not implement an effective reclaim scheme that avoids reclaiming frequently used delegations. In that case the server may issue a CB_RECALL_ANY callback to the client asking it to keep N delegations and return the rest. The reason why CB_RECALL_ANY specifies a count of delegations the client may keep as opposed to a count of delegations the client must yield is as follows. Were it otherwise, there is a potential for a race between a CB_RECALL_ANY that had a count of delegations to free with a set of client originated operations to return delegations. As a result of the race the client and server would have differing ideas as to how many delegations to return. Hence the client could mistakenly free too many delegations. This operation applies to delegations for a regular file (read or write) as well as for a directory.

The client can choose to return any type of delegation as a result of this callback i.e. read, write or directory delegation. The client can also choose to keep more delegations than what the server asked for and it is up to the server to handle this situation. The server must give the client enough time to return the delegations. This time should not be less than the lease period.

ERRORS
NFS4ERR_RESOURCE

9. Delegation Recovery

Crash recovery has two main goals, avoiding the necessity of breaking application guarantees with respect to locked files and delivery of updates cached at the client. Neither of these applies to directories protected by read delegations and notifications. Thus, the client is required to establish a new delegation on a server or client reboot.

10. Issues

10.1. Synchronous vs. Asynchronous notifications

Asynchronous notifications are defined as a way of updating namespace information for directories. For example, for directories that are very large or changing very slowly, the recall and subsequent reacquiring of state may be too expensive. In that case if used properly notifications can reduce the overhead of recalling delegations and re-fetching directory contents with a reduction in network traffic.

For achieving namespace cache consistency with lower network traffic, delegations along with synchronous callbacks are sufficient. Adding synchronous notifications on top of that does not provide much additional benefits.

11. RPC Definition File Changes

```
/*
 * Copyright (C) The Internet Society (2003)
 * All Rights Reserved.
 */

/*
 * nfs41_prot.x
 */
```

```
   %/* $Id: nfs41_prot.x,v 1.1 2004/02/01 05:10:53 saadia Exp $ */


   /* new operation, GET_DIR_DELEGATION */

   /*
    * Notification mask for letting the server know which notifications
    * the client is interested in.
    */
   typedef uint32_t dir_notification_type4;

   /*
    * The bitmask constants used for notification_type field
    */
   const DIR_NOTIFICATION_NONE                    = 0x00000000;
   const DIR_NOTIFICATION_CHANGE_CHILD_ATTRIBUTES  = 0x00000001;
   const DIR_NOTIFICATION_CHANGE_DIR_ATTRIBUTES   = 0x00000002;
   const DIR_NOTIFICATION_REMOVE_ENTRY            = 0x00000004;
   const DIR_NOTIFICATION_ADD_ENTRY               = 0x00000008;
   const DIR_NOTIFICATION_RENAME_ENTRY            = 0x00000010;
   const DIR_NOTIFICATION_CHANGE_COOKIE_VERIFIER  = 0x00000020;

   typedef nfstime4 attr_notice4;

   /*
    * Input arguments passed to the GET_DIR_DELEGATION operation.
    */
   struct GET_DIR_DELEGATION4args {
           /* CURRENT_FH: directory */
           dir_notification_type4      notification_type;
           attr_notice4                child_attr_delay;
           attr_notice4                dir_attr_delay;
   };

   /*
    * Result flags
    */

   struct GET_DIR_DELEGATION4resok {
           verifier4               cookieverf;
           /* Stateid for get_dir_delegation */
           stateid4                stateid;
           /* Which notifications can the server support */
           dir_notification_type4  supp_notification;
           /* Which attribute notifications can the server support */
           bitmap4                 child_attributes;
           bitmap4                 dir_attributes;
   };
```

```
   union GET_DIR_DELEGATION4res switch (nfsstat4 status) {
   case NFS4_OK:
         /* CURRENT_FH: delegated dir */
          GET_DIR_DELEGATION4resok      resok4;
   default:
          void;
   };


   /*
    * Operation arrays
    */

   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,
      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,
      OP_RESTOREFH            = 31,
      OP_SAVEFH               = 32,
      OP_SECINFO              = 33,
      OP_SETATTR              = 34,
      OP_SETCLIENTID          = 35,
      OP_SETCLIENTID_CONFIRM  = 36,
        OP_VERIFY               = 37,
         OP_WRITE                = 38,
         OP_RELEASE_LOCKOWNER    = 39,
         OP_OPENDIR              = 40,
         OP_ILLEGAL              = 10044
      };

      union nfs_argop4 switch (nfs_opnum4 argop) {
      case OP_ACCESS:        ACCESS4args opaccess;
      case OP_CLOSE:         CLOSE4args opclose;
      case OP_COMMIT:        COMMIT4args opcommit;
      case OP_CREATE:        CREATE4args opcreate;
      case OP_DELEGPURGE:    DELEGPURGE4args opdelegpurge;
      case OP_DELEGRETURN:   DELEGRETURN4args opdelegreturn;
      case OP_GETATTR:       GETATTR4args opgetattr;
      case OP_GETFH:         void;
      case OP_LINK:          LINK4args oplink;
      case OP_LOCK:          LOCK4args oplock;
      case OP_LOCKT:         LOCKT4args oplockt;
      case OP_LOCKU:         LOCKU4args oplocku;
      case OP_LOOKUP:        LOOKUP4args oplookup;
      case OP_LOOKUPP:       void;
      case OP_NVERIFY:       NVERIFY4args opnverify;
      case OP_OPEN:          OPEN4args opopen;
      case OP_OPENATTR:      OPENATTR4args opopenattr;
      case OP_OPEN_CONFIRM:  OPEN_CONFIRM4args opopen_confirm;
      case OP_OPEN_DOWNGRADE:        OPEN_DOWNGRADE4args opopen_downgrade;
      case OP_PUTFH:         PUTFH4args opputfh;
      case OP_PUTPUBFH:      void;
      case OP_PUTROOTFH:     void;
      case OP_READ:          READ4args opread;
      case OP_READDIR:       READDIR4args opreaddir;
      case OP_READLINK:      void;
      case OP_REMOVE:        REMOVE4args opremove;
      case OP_RENAME:        RENAME4args oprename;
      case OP_RENEW:         RENEW4args oprenew;
      case OP_RESTOREFH:     void;
      case OP_SAVEFH:        void;
      case OP_SECINFO:       SECINFO4args opsecinfo;
      case OP_SETATTR:       SETATTR4args opsetattr;
      case OP_SETCLIENTID:   SETCLIENTID4args opsetclientid;
      case OP_SETCLIENTID_CONFIRM:   SETCLIENTID_CONFIRM4args
                                         opsetclientid_confirm;
      case OP_VERIFY:        VERIFY4args opverify;
      case OP_WRITE:         WRITE4args opwrite;
      case OP_RELEASE_LOCKOWNER:     RELEASE_LOCKOWNER4args
                                     oprelease_lockowner;
      case OP_OPENDIR:       OPENDIR4args opopendir;
      case OP_ILLEGAL:       void;
```

```
   };
   union nfs_resop4 switch (nfs_opnum4 resop){
   case OP_ACCESS:        ACCESS4res opaccess;
   case OP_CLOSE:         CLOSE4res opclose;
   case OP_COMMIT:        COMMIT4res opcommit;
   case OP_CREATE:        CREATE4res opcreate;
   case OP_DELEGPURGE:    DELEGPURGE4res opdelegpurge;
   case OP_DELEGRETURN:   DELEGRETURN4res opdelegreturn;
   case OP_GETATTR:       GETATTR4res opgetattr;
   case OP_GETFH:         GETFH4res opgetfh;
   case OP_LINK:          LINK4res oplink;
   case OP_LOCK:          LOCK4res oplock;
   case OP_LOCKT:         LOCKT4res oplockt;
   case OP_LOCKU:         LOCKU4res oplocku;
   case OP_LOOKUP:        LOOKUP4res oplookup;
   case OP_LOOKUPP:       LOOKUPP4res oplookupp;
   case OP_NVERIFY:       NVERIFY4res opnverify;
   case OP_OPEN:          OPEN4res opopen;
   case OP_OPENATTR:      OPENATTR4res opopenattr;
   case OP_OPEN_CONFIRM:  OPEN_CONFIRM4res opopen_confirm;
   case OP_OPEN_DOWNGRADE:        OPEN_DOWNGRADE4res opopen_downgrade;
   case OP_PUTFH:         PUTFH4res opputfh;
   case OP_PUTPUBFH:      PUTPUBFH4res opputpubfh;
   case OP_PUTROOTFH:     PUTROOTFH4res opputrootfh;
   case OP_READ:          READ4res opread;
   case OP_READDIR:       READDIR4res opreaddir;
   case OP_READLINK:      READLINK4res opreadlink;
   case OP_REMOVE:        REMOVE4res opremove;
   case OP_RENAME:        RENAME4res oprename;
   case OP_RENEW:         RENEW4res oprenew;
   case OP_RESTOREFH:     RESTOREFH4res oprestorefh;
   case OP_SAVEFH:        SAVEFH4res opsavefh;
   case OP_SECINFO:       SECINFO4res opsecinfo;
   case OP_SETATTR:       SETATTR4res opsetattr;
   case OP_SETCLIENTID:   SETCLIENTID4res opsetclientid;
   case OP_SETCLIENTID_CONFIRM:   SETCLIENTID_CONFIRM4res
                                      opsetclientid_confirm;
   case OP_VERIFY:        VERIFY4res opverify;
   case OP_WRITE:         WRITE4res opwrite;
   case OP_RELEASE_LOCKOWNER:     RELEASE_LOCKOWNER4res
                                  oprelease_lockowner;
   case OP_OPENDIR:       OPENDIR4res opopendir;
   case OP_ILLEGAL:       ILLEGAL4res opillegal;
   };

   struct COMPOUND4args {
      utf8str_cs      tag;
      uint32_t        minorversion; /* == 1 !!! */
      nfs_argop4      argarray<>;
   };
   struct COMPOUND4res {
      nfsstat4 status;
      utf8str_cs      tag;
      nfs_resop4      resarray<>;
   };


   /*
    * New error codes
    */

   enum nfsstat4 {
           NFS4_OK                 = 0,    /* everything is okay      */
           NFS4ERR_PERM            = 1,    /* caller not privileged   */
           NFS4ERR_NOENT           = 2,    /* no such file/directory  */
           NFS4ERR_IO              = 5,    /* hard I/O error          */
           NFS4ERR_NXIO            = 6,    /* no such device          */
           NFS4ERR_ACCESS          = 13,   /* access denied           */
           NFS4ERR_EXIST           = 17,   /* file already exists     */
           NFS4ERR_XDEV            = 18,   /* different filesystems   */
           /* Unused/reserved        19 */
           NFS4ERR_NOTDIR          = 20,   /* should be a directory   */
           NFS4ERR_ISDIR           = 21,   /* should not be directory */
           NFS4ERR_INVAL           = 22,   /* invalid argument        */
           NFS4ERR_FBIG            = 27,   /* file exceeds server max */
           NFS4ERR_NOSPC           = 28,   /* no space on filesystem  */
           NFS4ERR_ROFS            = 30,   /* read-only filesystem    */
           NFS4ERR_MLINK           = 31,   /* too many hard links     */
           NFS4ERR_NAMETOOLONG     = 63,   /* name exceeds server max */
           NFS4ERR_NOTEMPTY        = 66,   /* directory not empty     */
           NFS4ERR_DQUOT           = 69,   /* hard quota limit reached*/
           NFS4ERR_STALE           = 70,   /* file no longer exists   */
           NFS4ERR_BADHANDLE       = 10001,/* Illegal filehandle      */
           NFS4ERR_BAD_COOKIE      = 10003,/* READDIR cookie is stale */
           NFS4ERR_NOTSUPP         = 10004,/* operation not supported */
           NFS4ERR_TOOSMALL        = 10005,/* response limit exceeded */
           NFS4ERR_SERVERFAULT     = 10006,/* undefined server error  */
           NFS4ERR_BADTYPE         = 10007,/* type invalid for CREATE */
           NFS4ERR_DELAY           = 10008,/* file "busy" - retry     */
           NFS4ERR_SAME            = 10009,/* nverify says attrs same */
           NFS4ERR_DENIED          = 10010,/* lock unavailable        */
           NFS4ERR_EXPIRED         = 10011,/* lock lease expired      */
           NFS4ERR_LOCKED          = 10012,/* I/O failed due to lock  */
           NFS4ERR_GRACE           = 10013,/* in grace period         */
           NFS4ERR_FHEXPIRED       = 10014,/* filehandle expired      */
           NFS4ERR_SHARE_DENIED    = 10015,/* share reserve denied    */
           NFS4ERR_WRONGSEC        = 10016,/* wrong security flavor   */
           NFS4ERR_CLID_INUSE      = 10017,/* clientid in use         */
           NFS4ERR_RESOURCE        = 10018,/* resource exhaustion     */
           NFS4ERR_MOVED           = 10019,/* filesystem relocated    */
           NFS4ERR_NOFILEHANDLE    = 10020,/* current FH is not set   */
           NFS4ERR_MINOR_VERS_MISMATCH = 10021,/* minor vers not supp */
           NFS4ERR_STALE_CLIENTID  = 10022,/* server has rebooted     */
           NFS4ERR_STALE_STATEID   = 10023,/* server has rebooted     */
           NFS4ERR_OLD_STATEID     = 10024,/* state is out of sync    */
           NFS4ERR_BAD_STATEID     = 10025,/* incorrect stateid       */
           NFS4ERR_BAD_SEQID       = 10026,/* request is out of seq.  */
           NFS4ERR_NOT_SAME        = 10027,/* verify - attrs not same */
           NFS4ERR_LOCK_RANGE      = 10028,/* lock range not supported*/
           NFS4ERR_SYMLINK         = 10029,/* should be file/directory*/
           NFS4ERR_RESTOREFH       = 10030,/* no saved filehandle     */
           NFS4ERR_LEASE_MOVED     = 10031,/* some filesystem moved   */
           NFS4ERR_ATTRNOTSUPP     = 10032,/* recommended attr not sup*/
           NFS4ERR_NO_GRACE        = 10033,/* reclaim outside of grace*/
           NFS4ERR_RECLAIM_BAD     = 10034,/* reclaim error at server */
           NFS4ERR_RECLAIM_CONFLICT = 10035,/* conflict on reclaim    */
           NFS4ERR_BADXDR          = 10036,/* XDR decode failed       */
           NFS4ERR_LOCKS_HELD      = 10037,/* file locks held at CLOSE*/
           NFS4ERR_OPENMODE        = 10038,/* conflict in OPEN and I/O*/
           NFS4ERR_BADOWNER        = 10039,/* owner translation bad   */
           NFS4ERR_BADCHAR         = 10040,/* utf-8 char not supported*/
           NFS4ERR_BADNAME         = 10041,/* name not supported      */
           NFS4ERR_BAD_RANGE       = 10042,/* lock range not supported*/
           NFS4ERR_LOCK_NOTSUPP    = 10043,/* no atomic up/downgrade  */
           NFS4ERR_OP_ILLEGAL      = 10044,/* undefined operation     */
           NFS4ERR_DEADLOCK        = 10045,/* file locking deadlock   */
           NFS4ERR_FILE_OPEN       = 10046,/* open file blocks op.    */
           NFS4ERR_ADMIN_REVOKED   = 10047,/* lockowner state revoked */
           NFS4ERR_CB_PATH_DOWN    = 10048,/* callback path down      */
           NFS4ERR_DIRDELEG_UNAVAIL= 10049,/* dir dlg. not returned   */
   };


   /*
    * New Callback operation CB_NOTIFY
    */

   struct CB_NOTIFY4args {
           stateid4                stateid;
           dir_notification4       changes<>;
   };

   /*
    * Changed entry information.
    */
   struct dir_entry {
           component4      file;
           fattr4          attrs;
```

```
   };

   struct dir_notification_attribute4 {
           dir_entry   changed_entry;
   };

   struct dir_notification_remove4 {
           dir_entry   old_entry;
           nfs_cookie4 old_entry_cookie;
   };

   struct dir_notification_rename4 {
           dir_entry              old_entry;
           dir_notification_add4  new_entry;
   };

   struct dir_notification_verifier4 {
           verifier4       old_cookieverf;
           verifier4       new_cookieverf;
   };

   struct dir_notification_add4 {
           dir_entry         new_entry;
           nfs_cookie4       new_entry_cookie; /* what READDIR would
                                                have returned
                                                for this entry */
           bool              last_entry;
           prev_entry_info4  prev_info;
   };

   union prev_entry_info4 switch (bool isprev) {
   case TRUE:
           prev_entry4 prev_entry_info;
   case FALSE:             /* we are adding to an empty directory */
           void;
   };

   /*
    * Previous entry information
    */
   struct prev_entry4 {
           dir_entry       prev_entry;
           /* what READDIR returned for this entry */
           nfs_cookie4     prev_entry_cookie;
   };

   /*
    * Notification information sent to the client.
    */
```

```
   union dir_notification4
   switch (dir_notification_type4 notification_type) {
           case DIR_NOTIFICATION_CHANGE_CHILD_ATTRIBUTES:
                   dir_notification_attribute4 change_child_attributes;
           case DIR_NOTIFICATION_CHANGE_DIR_ATTRIBUTES:
                   fattr4                      change_dir_attributes;
           case DIR_NOTIFICATION_REMOVE_ENTRY:
                   dir_notification_remove4    remove_notification;
           case DIR_NOTIFICATION_ADD_ENTRY:
                   dir_notification_add4       add_notification;
           case DIR_NOTIFICATION_RENAME_ENTRY:
                   dir_notification_rename4    rename_notification;
           case DIR_NOTIFICATION_CHANGE_COOKIE_VERIFIER:
                   dir_notification_verifier4  verf_notification;
   };

   struct CB_NOTIFY4res {
           nfsstat4        status;
   };

   /*
    * New Callback operation CB_RECALL_ANY
    *

   struct CB_RECALLANYY4args {
           uint4           dlgs_to_keep;
   }

   struct CB_RECALLANY4res {
           nfsstat4        status;
   };

   /*
    * Various definitions for CB_COMPOUND
    */
   enum nfs_cb_opnum4 {
           OP_CB_GETATTR           = 3,
           OP_CB_RECALL            = 4,
           OP_CB_NOTIFY            = 5,
           OP_CB_RECALL_ANY        = 6,
           OP_CB_ILLEGAL           = 10044
   };

   union nfs_cb_argop4 switch (unsigned argop) {
   case OP_CB_GETATTR:     CB_GETATTR4args opcbgetattr;
   case OP_CB_RECALL:      CB_RECALL4args opcbrecall;
   case OP_CB_NOTIFY:      CB_NOTIFY4args opcbnotify;
   case OP_CB_RECALLANY:   CB_RECALLANY4args opcbrecallany;
   case OP_CB_ILLEGAL:     CB_ILLEGAL4args opcbillegal;
   };

   union nfs_cb_resop4 switch (unsigned resop) {
   case OP_CB_GETATTR:     CB_GETATTR4res opcbgetattr;
   case OP_CB_RECALL:      CB_RECALL4res opcbrecall;
   case OP_CB_NOTIFY:      CB_NOTIFY4res opcbnotify;
   case OP_CB_RECALLANY:   CB_RECALLANY4res opcbrecallany;
   case OP_CB_ILLEGAL:     CB_ILLEGAL4res opcbillegal;
   };

   struct CB_COMPOUND4args {
           utf8str_cs      tag;
           uint32_t        minorversion;
           uint32_t        callback_ident;
           nfs_cb_argop4   argarray<>;
   };

   struct CB_COMPOUND4res {
           nfsstat4        status;
           utf8str_cs      tag;
           nfs_cb_resop4   resarray<>;
   };
```

12. IANA Considerations

The IANA considerations of NFSv4.0 apply to NFSv4.1.

13. Acknowledgements

David Noveck, Michael Eisler, Carl Burnett, Ted Anderson and Thomas Talpey for their constructive feedback and critical comments.

14. Normative References

[RFC3530]
S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck, "NFS version 4 Protocol", RFC 3530, April, 2003.

[talpey]
T. Talpey, S. Shepler, J. Bauman "NFSv4 Session Extensions", Internet-Draft, July, 2004. A URL for this Internet-Draft is available at http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-sess-00.txt

15. Informative References

None.

16. Author's Address

Saadia Khan
2324 Dubois Street
Milpitas, CA 95035
USA

Phone: 408-957-9626
EMail: saadiak@yahoo.com

17. IPR Notices

18. Copyright Notice