Introducing Network File System Version 4 on HP-UX 11i v3

Abstract

Beginning with HP-UX 11i v3, HP offers Network File System Version 4 (NFSv4). NFSv4 offers better interoperability with Microsoft® Windows®, improved security, and improved performance. This paper describes the NFSv4 features available in the initial and subsequent fusion releases of HP-UX 11i v3 and what an HP-UX system administrator needs to know to use NFSv4 clients and servers.

New features of NFSv4

The new NFSv4 features offered in HP-UX 11i v3 are:

- NFSv4 introduces state. NFSv4 is a stateful protocol unlike NFSv2 or NFSv3.
- NFSv4 introduces file delegation. An NFSv4 server can enable an NFSv4 client to access and modify a file in its cache without sending any network requests to the server.
- NFSv4 uses compound remote procedure calls (RPCs) to reduce network traffic. An NFSv4 client can combine several traditional NFS operations (LOOKUP, OPEN, and READ) into a single compound RPC request to carry out a complex operation in one network round trip.
- NFSv4 specifies a number of sophisticated security mechanisms including Kerberos 5 and Access Control Lists.
- NFSv4 can seamlessly coexist with NFSv3 and NFSv2 clients and servers.

NFSv4 changes

Five big differences between the NFSv4 protocol and earlier Network File System (NFS) protocol versions are:

- Compound RPCs (There are now only two RPCs, the NULL procedure, and the COMPOUND procedure.)
- Elimination of a separate mount, locking, and ACL protocols
- Introduction of state into the protocol
- Security
- Delegation

COMPOUND remote procedure call

A common complaint of earlier NFS protocols was that the request-response nature of NFS was inefficient. That is, if a request depended on the results of a previous request, then the client had to wait for a reply from the server before sending the second request. Consequently, there was unnecessary latency inherent in the protocol. COMPOUND requests provide a mechanism by which clients can bundle multiple NFS operations into a single over-the-wire request.

In this paper, NFS requests and replies are displayed using Ethereal traces. Ethereal, also known as Wireshark, is a multiprotocol network analyzer tool available from the HP Internet Express Bundle (http://software.hp.com) or by free download from http://www.ethereal.com. Ethereal displays the NFSv4 requests and replies as they are sent over the wire. Included in the output are all the fields specified by the protocol. Because the output can be quite long, sections of the output have been deleted and replaced with ellipsis.

Figure 1. COMPOUND RPC request

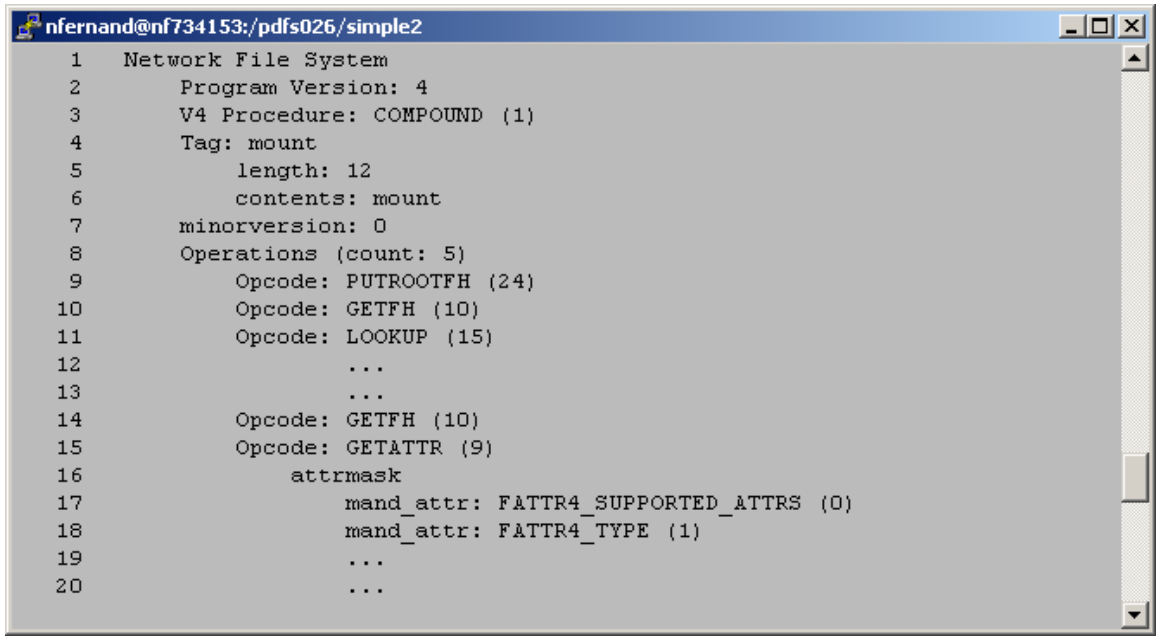


Figure 1 is an example of an Ethereal trace generated by mounting a file system over NFSv4.

Lines 1 and 2 denote that this is an NFS trace and that the protocol version is 4. Line 3 shows which NFSv4 RPC the client is making. Because the only other NFSv4 RPC is the NULL RPC, in almost all cases, the NFSv4 RPC will be the COMPOUND procedure. Lines 4 to 8 provide header information that describes the RPC. For example, line 4 is a tag describing this as a mount request. COMPOUND procedures consist of multiple operations, also called opcodes. Line 8 shows that there are five opcodes present within this request.

The remaining lines are a textual representation of the NFSv4 request. The indentation reflects the hierarchy of the data. For example, line 16 is indented relative to line 15 because attrmask is a field within the GETATTR opcode and lines 17 and 18 reflect bits turned on within the attrmask field. There are more bits turned on (lines 19 and 20) in the attrmask, but they have been replaced with ellipsis to shorten the output.

Within the COMPOUND request are the opcodes PUTROOTFH, GETFH, LOOKUP, a second GETFH, and GETATTR appear on lines 9, 10, 11, 14, and 15, respectively. These and all other opcodes are documented in the IETF RFC 3530 available at http://www.ietf.org/rfc/rfc3530.txt.

Figure 2. COMPOUND RPC reply

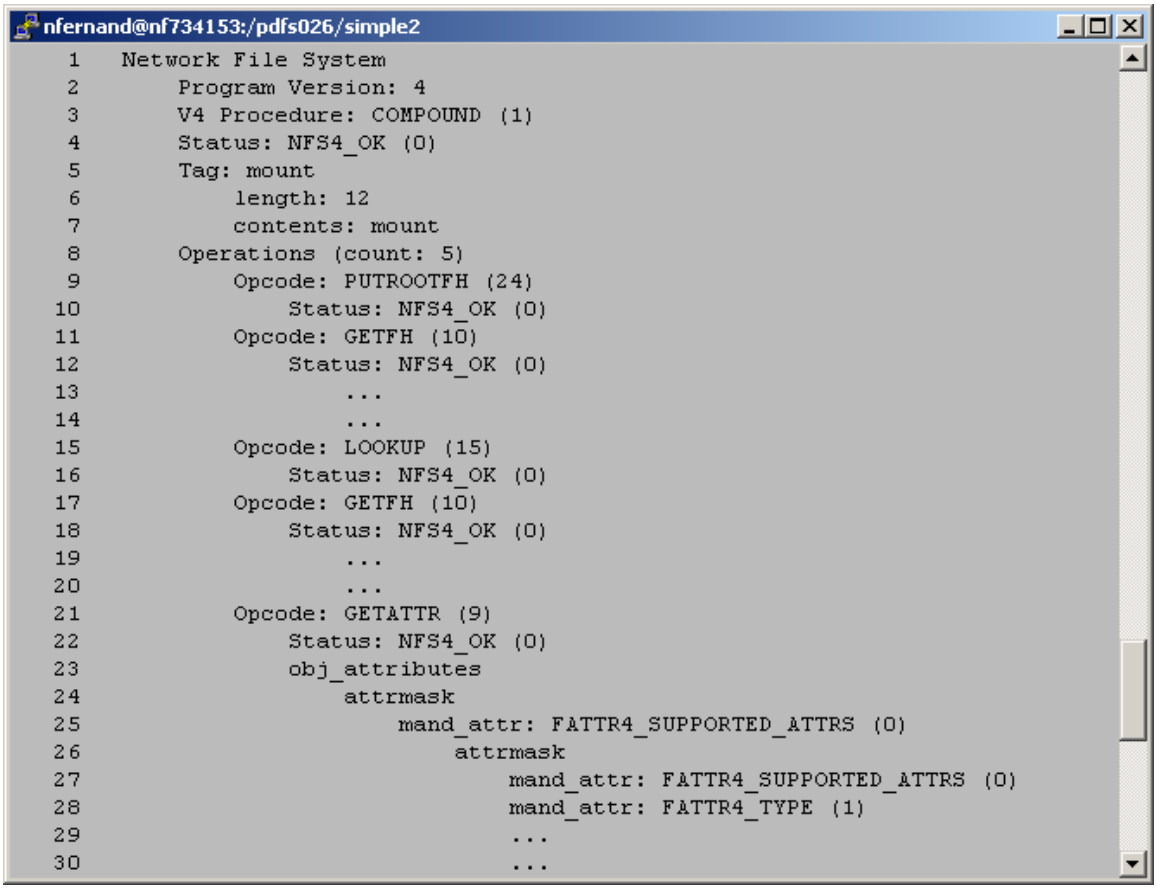


Figure 2 displays the server's reply to the COMPOUND RPC in Figure 1.

Unless an error occurs, each opcode has its own return status and its own return values. For example, the result of the PUTROOTFH opcode (lines 9 and 10) is the return status "NFS4_OK", indicating the operation succeeded without error. The results of the GETATTR opcode include the status (NFS4_OK on line 22) and the values of the different attributes requested by the GETATTR opcode.

If an error occurs in one of the opcodes, then execution of the remaining opcodes is terminated, and the accumulated results are returned to the client.

Note the request and response for FATTR4_SUPPORTED_ATTRS in Figure 1, line 17 and in Figure 2, lines 25 through 28. NFSv4 enhances the types of attributes supported by NFS. For example, there are mandatory attributes that are required for NFSv4 compliance, recommended attributes that should be supported for NFS interoperability, and named attributes that enable the server to maintain a value/pair assignment. Named attributes are optional and not currently supported in the NFSv4 implementation on HP-UX 11i v3.

The GETATTR operation enables NFSv4 clients to query a server as to which NFSv4 attributes it supports for a particular file. This functionality increases Windows and UNIX® interoperability by allowing Windows client applications and NFSv4 implementations to query an NFSv4 server as to which attributes it supports and include or exclude features accordingly.

No mount or locking protocols

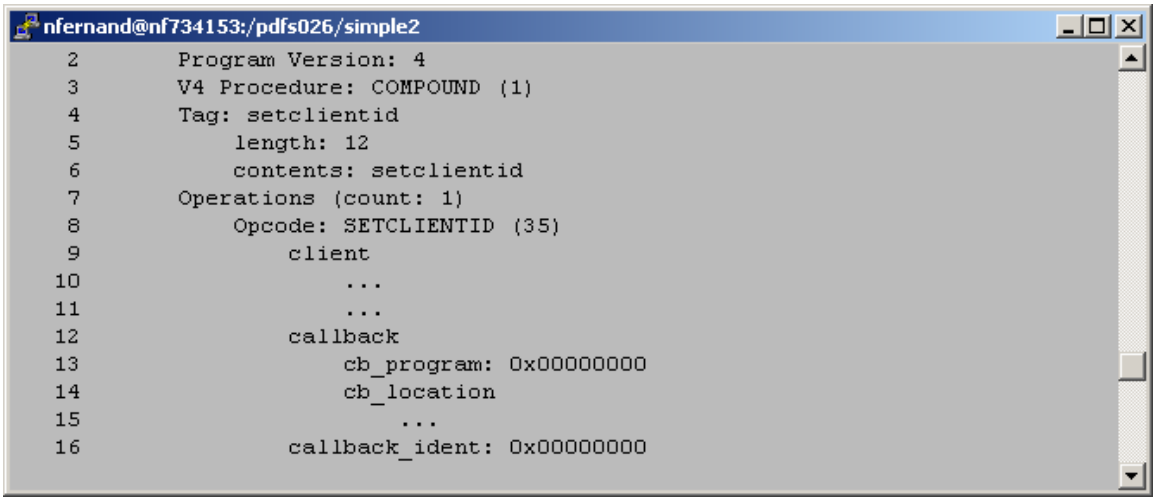
In Figure 1, there are no mount protocol messages; mounting is handled by the PUTROOTFH opcode, which is part of NFSv4. Similarly, there is no separate file locking protocol. Locking is handled by the NFSv4 opcodes LOCK, LOCKU, and LOCKT. These opcodes are discussed in the "File Locking as part of the NFSv4 protocol" section.

NFSv4 as a stateful protocol

Previous NFS protocols were stateless in the sense that the NFS server maintained no information or state about its clients. For example, previous NFS protocols define no open or close operations; NFS clients perform lookup operations on directories and files, and NFS servers respond with the appropriate directory or file handle. A file operation such as a `read()` would manifest itself as an NFS request to read a range of bytes from a particular file handle. Although opaque to the NFS client, this file handle often represents the disk-based data associated with a file.

NFSv4, however, is stateful; the NFSv4 server maintains information about its clients, the files they hold open, and locks. For example, a second COMPOUND procedure (shown in Figure 3) was also generated as part of the `mount` command. This COMPOUND procedure consists solely of the SETCLIENTID opcode.

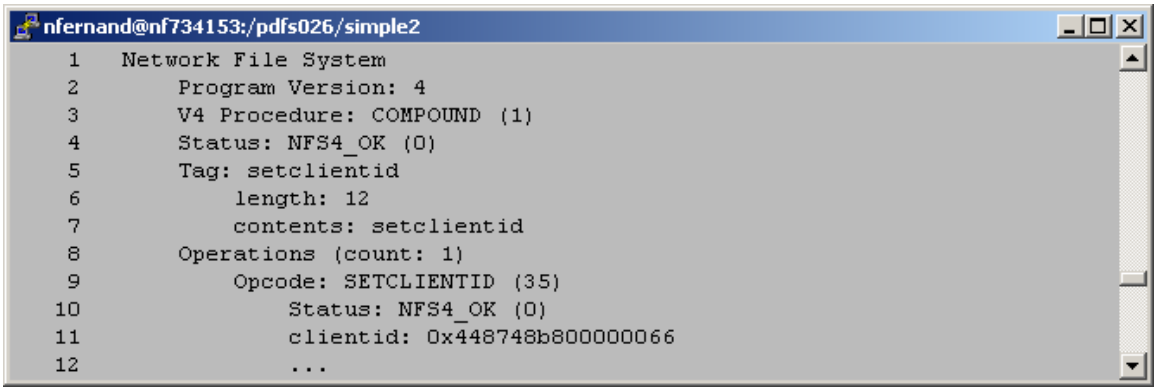
Figure 3. SETCLIENTID opcode request



The SETCLIENTID opcode is creates a data structure used to manage state associated with this client on the server. The SETCLIENTID opcode consists of the fields client, callback, and callback_ident. The callback fields shown in lines 9 through 14 in Figure 3 are associated with this client identifier on the server and are used to manage delegations. Delegations are covered in a subsequent section.

When the server receives a SETCLIENTID opcode, the server assigns a unique identifier to the client and returns that identifier to the client in its reply to the SETCLIENTID opcode. This reply is shown in Figure 4.

Figure 4. SETCLIENTID opcode reply

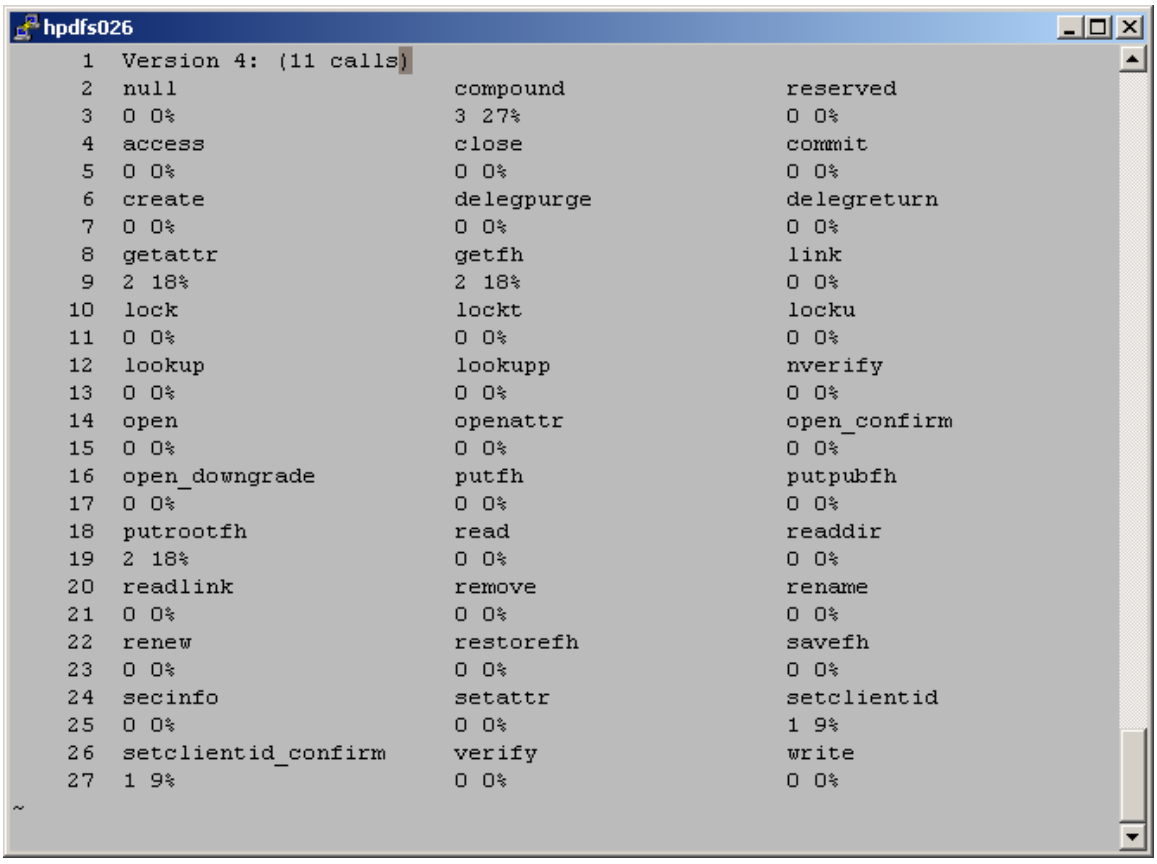


The client identifier assigned by the server to this client appears on line 11 in Figure 4.

Because NFSv4 is a stateful protocol, some opcodes require the client to identify itself by passing its client identifier to the server. For example, because file locks are associated with a client (and therefore a client identifier), the locking opcodes, LOCK, LOCKU, and LOCKT, all require the client to send its client identifier as part of the operation.

To see the list of opcodes defined as part of NFSv4, execute the `nfsstat` command. The `nfsstat` command has been enhanced to display NFSv4 statistics. For example, Figure 5 displays the NFSv4 opcodes executed after the initial NFSv4 mount.

Figure 5. `nfsstat` NFSv4 output



The output displays not just the NFSv4 procedures, NULL and COMPOUND, but also all opcodes that can be sent as part of a COMPOUND procedure. The opcodes listed include GETFH (line 8), PUTROOTFH (line 18), and SETCLIENTID (line 24). These opcodes were used in the initial mount request seen in Figure 1 and in identifying the client as seen in Figure 3. See the RFC for details on SETCLIENTID_CONFIRM.

Some other opcodes of interest in Figure 5 are file-open related opcodes, OPEN (line 14) and CLOSE (line 4), and locking-related opcodes, LOCK, LOCKT, and LOCKU (line 10).

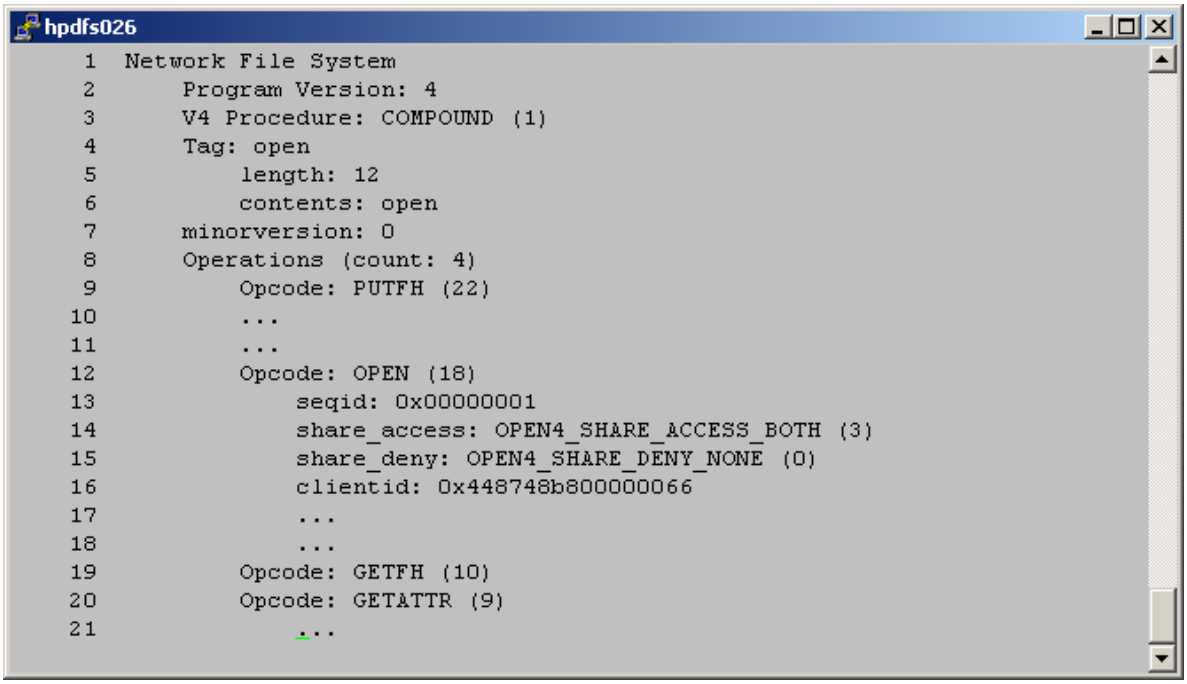
Open and close

Other important stateful operations in NFSv4 are file opens and closes. Figure 6 shows the Ethereal trace generated after an `open()` system call was made.

On an OPEN, the server returns a state identifier that identifies this instance of this client's file open. The state associated with this file open is maintained on the server.

The client identification returned to the client by the server is validated on each NFSv4 operation.

Figure 6. OPEN opcode request

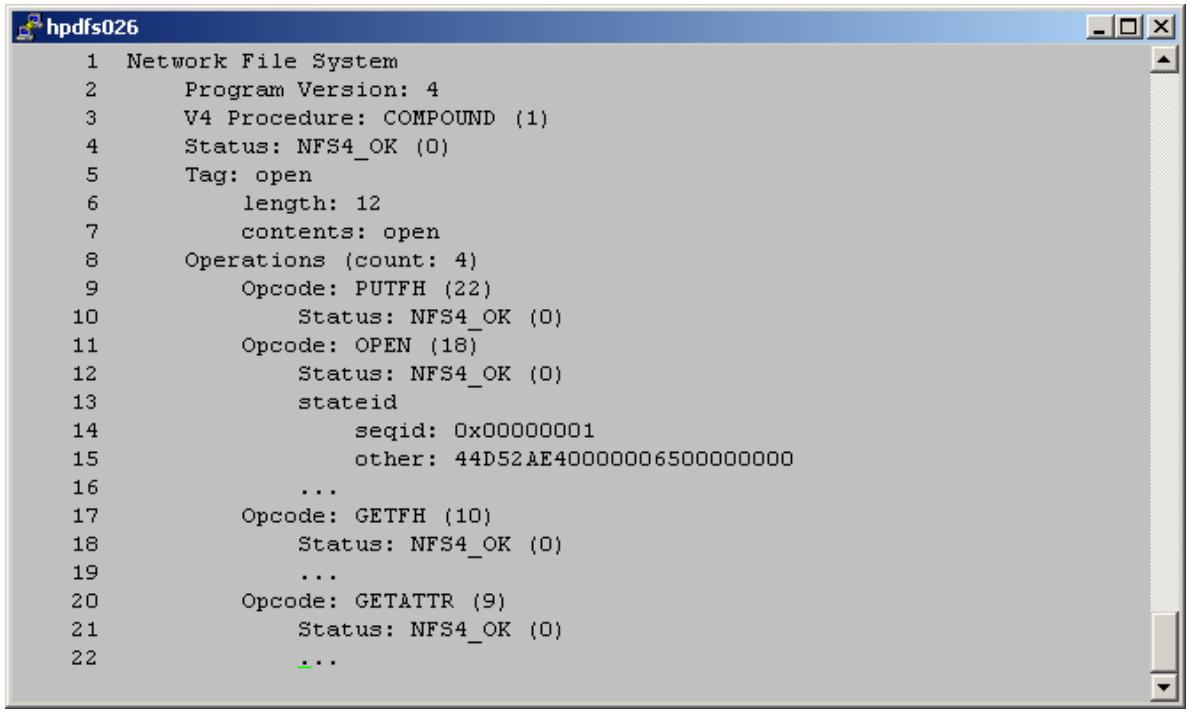


On line 12, the OPEN opcode appears. A parameter of the OPEN opcode is the client identifier (line 16). As previously mentioned, the client identifier is required to identify the client performing the open. Note how the client identifier on line 16 matches the identifier that was returned to the client as part of the SETCLIENTID opcode in Figure 4, line 11.

Note the share_access and share_deny fields in the OPEN opcode on lines 14 and 15. These fields are used to support share lock semantics, which are familiar in the Windows environment and are now available in NFSv4. Share lock support improves Windows and UNIX interoperability by enabling Windows applications that rely on share locks to work with HP-UX 11i v3 NFSv4 servers.

Share locks enable clients open access to a file while at the same time denying future open accesses to other client processes. If the server is unable to acquire the requested locks or deny future lock requests, the open fails. For example, if a client wants to prevent updates to a file while the client reads the file, then the client would open the file with a share read lock and a share deny write lock. If no other clients are writing the file, then the open succeeds. The share locks are in effect until the client closes the file. Subsequent attempts by other clients to open the file with share write access fail until the client closes the file.

Figure 7. OPEN opcode reply



The reply from the server is shown in Figure 7. The state identifier on line 13 represents this client's file open. It is returned by the server as part of the reply to the OPEN opcode. This identifier is used to associate state with this instance of this file open.

Delegation

Normally, an NFS server is responsible for managing file locks and file data across a network. The centralized responsibility is necessary when multiple client machines are accessing a remotely served file.

However, if only one client is writing to a file or multiple clients are only reading a file, then management of that file's locks and data can be delegated (management of locks and data for that file are performed locally on that client or clients). Delegation results in performance gains because network traffic between the NFS server and NFS client necessary to manage that file is avoided.

Because clients take on the responsibility for managing a file's locks and data, the server must verify that the client is still up and reachable. The NFSv4 server verifies the client's availability by sending RPC requests to the NFSv4 client's callback daemon (nfs4cbd). The server uses the callback and callback_ident fields that were sent in the SETCLIENTID request (see Figure 3) to determine where the client's nfs4cbd is listening.

When a delegation must be revoked, the nfs4cbd is responsible for returning to the server any locking or file data managed on the client. For example, a delegation must be revoked if a local application attempts to write to a file while the NFS client is writing to the same file, otherwise there will be data corruption.

However, until ONCplus version B.11.31.03, NFSv4 servers on HP-UX 11i v3 support delegation with the caveat that no local file access should occur on that file.

NOTE: ONCplus version B.11.31.03 (or later) will be included in the HP-UX 11i v3 Update 3 fusion release planned for September 2008.

For example, if a directory, /dir, is being exported by a server, then no local processes on the server should be accessing files in /dir. With the introduction of the delegation stackable module in ONCplus B.11.31.03, this restriction is no longer applicable.

File Locking as part of the NFSv4 protocol

While previous versions of NFS were stateless, accompanying protocols and daemons were created to manage things like file lock state and node status (that is, whether a node was up, rebooting, or down). While these protocols and daemons were associated with NFS, they were not part of the core NFS protocol. Specifically, the lock manager protocol, along with the lock and node status daemons (rpc.lockd and rpc.statd, respectively), were created. These daemons are still present on HP-UX 11i v3 in support of NFSv2 and NFSv3 clients.

However, in Figure 5, which lists all the opcodes supported by NFSv4, there are several locking-related opcodes: LOCK, LOCKT, and LOCKU (line 10). The LOCK, LOCKT, and LOCKU opcodes support creating locks, testing for locks, and unlocking files, respectively. Rather than continuing to support locking outside NFS, these opcodes were added to support locking from within the protocol.

The Ethereal trace in Figure 8 was generated by a C program that opens a file and then requests a read lock on the first 3 bytes. The request contains the stateid associated with the file (line 18) and the client identification (line 23). These match the state identifier granted by the OPEN opcode in Figure 7, line 15, and the client identifier granted by the SETCLIENTID opcode in Figure 4, line 11. Both are necessary because the lock request is associated with both this client and this instance of the file's open.

Figure 8. LOCK opcode request

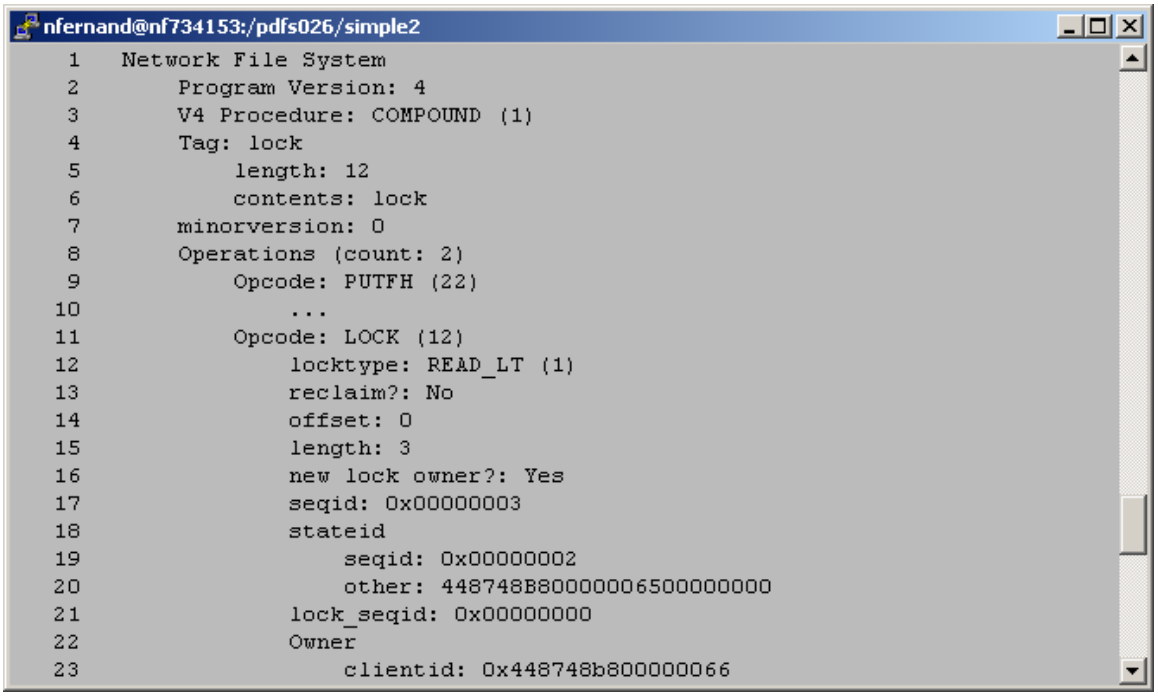
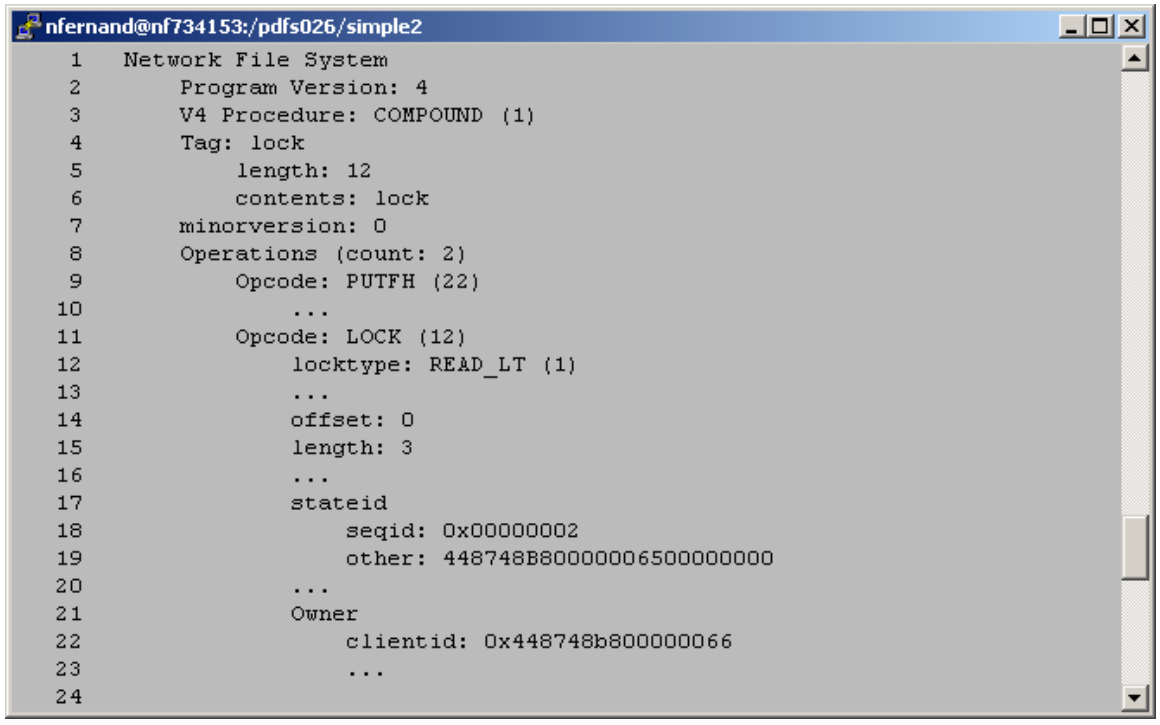


Figure 9. LOCK opcode reply



The reply from the server is shown in Figure 9. Notice how the server returns to the client a state identifier (lines 17 through 19) that identifies this particular lock. Also, the client identifier is returned to the client (line 22) as well because the server maintains state about the client. In this case, the server is keeping state about this client's specific locks.

When the client attempts to release a lock through the LOCKU opcode, the stateid associated with a lock is passed to the server so that the server can uniquely identify and process this unlock request.

Security

Previous NFS protocol versions did little to encourage use of advanced security. Instead, NFS clients normally used regular integer-based UNIX authentication (AUTH_SYS) or none at all (AUTH_NONE). NFSv4, however, ensures the availability of improved security by mandating that compliant NFSv4 implementations include advanced security mechanisms such as Kerberos.

The NFSv4 specification dictates that clients and servers negotiate a security flavor. The client sends the SECINFO opcode to the server. The server then replies with a list of available security flavors, listed in preferential order with the most preferred listed first. The client is then free to select which flavor it prefers. The client and server then use that flavor for the duration of the mounted filesystem.

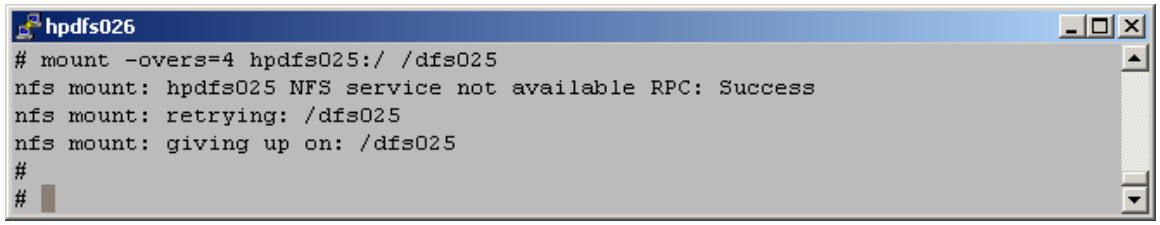
The AUTH_SYS and AUTH_NONE security flavors are still available on 11i v3 for those customers wishing to continue using them.

The NFSv4 specification also includes native support for access control lists (ACLs). The ACLs specification in NFSv4 is modeled on the ACLs present in Windows environments and, consequently, is closer to what is available in the Windows world. This is another feature that helps NFSv4 interoperate better with Windows systems.

Enabling NFSv4 mounts on your HP-UX 11i v3 server

By default, NFSv4 mounts are disabled on HP-UX 11i v3 servers. If the server does not enable NFSv4 mounts, then attempting to mount generates the following error message.

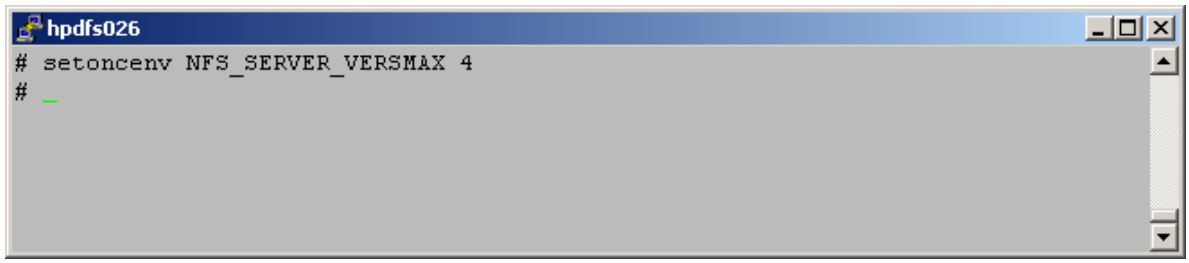
Figure 10. `mount` command fails



To enable NFSv4 mounts, the NFS configuration file, /etc/default/nfs, must be modified on the server. This file contains various configuration variables for NFS clients, NFS servers, and lock manager.

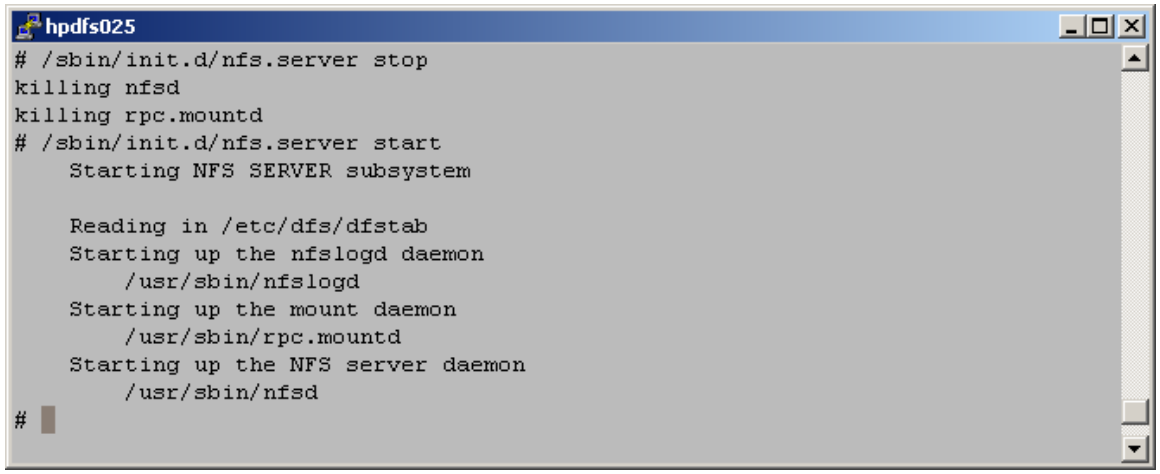
To enable clients to mount your NFSv4 file system, change the value of NFS_SERVER_VERSMAX to 4 in the /etc/default/nfs file as in Figure 11 by executing the `setoncenv(1M)` command.

Figure 11. Enabling NFSv4 on a server



Restart the NFS server services on the server using the startup/shutdown script, /sbin/init.d/nfs.server, with the `stop` and `start` parameters as in Figure 12.

Figure 12. Restarting NFSv4 server services



By default, NFS clients and servers negotiate which protocol version to use. The client and server use the highest numbered protocol version that both the client and server support. For example, if the client supports NFSv2, NFSv3, and NFSv4 and the server supports only NFSv2 and NFSv3, then the client and server agree to use NFSv3. In this way, NFSv4 clients and servers work seamlessly with other clients and servers that do not support NFSv4.

To enable only NFSv4 mounts, set the configuration parameters NFS_SERVER_VERSMIN and NFS_SERVER_VERSMAX to 4. The configuration parameters NFS_SERVER_VERSMIN and NFS_SERVER_VERSMAX tell the NFS server the range of NFS protocol versions to enable. For example, to enable only NFSv3 and NFSv4 mounts, set the configuration parameters NFS_SERVER_VERSMIN and NFS_SERVER_VERSMAX to 3 and 4, respectively.

After restarting the NFS server, check the /var/adm/syslog/syslog.log file for any error messages.

Lastly, share the directories that are to be made mountable by executing the `share` command. See man page `share(1M)` for details.

nfsmapid: Name-based user and group IDs

Whereas previous NFS protocol versions used 32-bit integers to represent user identifiers (UIDs) and group identifiers (GIDs), NFSv4 servers and clients exchange file owner and group attributes in the form of strings.

Figure 13. GETATTR opcode reply

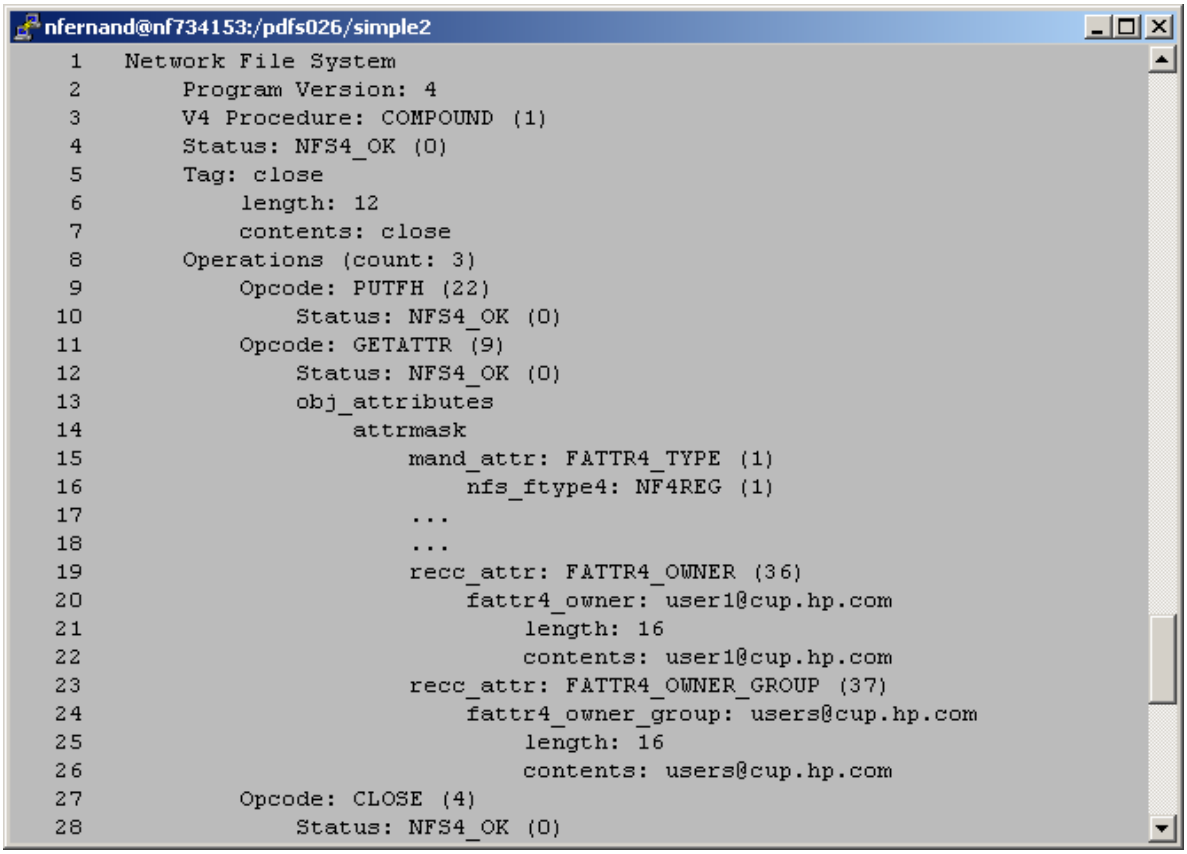


Figure 13 contains the results of a GETATTR opcode request. The GETATTR opcode was sent as part of the `close()` system call. The results to the GETATTR opcode begins on line 11. The value of the file's owner and group appear on lines19 through 26. In this case, the UID and GID are "user1@cup.hp.com" and "users@cup.hp.com," respectively.

Because the underlying file systems on HP-UX systems continue to use integer-based UIDs and GIDs, NFSv4 must perform translations between the string identifiers and their corresponding 32-bit integer UID and GID values. For example, if a file is being created, the NFS server must map the string user1@cup.hp.com to the 32-bit UID that matches user1 on the system. This translation is done by the nfsmapid daemon.

Mounting an NFSv4 file system on your HP-UX 11i v3 client

NFSv4 on HP-UX 11i v3 clients is disabled by default. Similar to enabling NFSv4 on HP-UX 11i v3 servers, enabling NFSv4 on HP-UX 11i v3 clients requires setting the configuration variable NFS_CLIENT_VERSMAX to 4 in the configuration file, /etc/default/nfs, and restarting the NFS client services using the startup and shutdown script, /sbin/init.d/nfs.client, with the stop and start parameters.

To enable only NFSv4 on HP-UX 11i v3 clients, set both the configuration variables NFS_CLIENT_VERSMIN and NFS_CLIENT_VERSMAX to 4.

Similar to NFS_SERVER_VERSMIN and NFS_SERVER_VERSMAX, NFS_CLIENT_VERSMIN and NFS_CLIENT_VERSMAX control the range of protocol versions that your HP-UX 11i v3 client uses. For

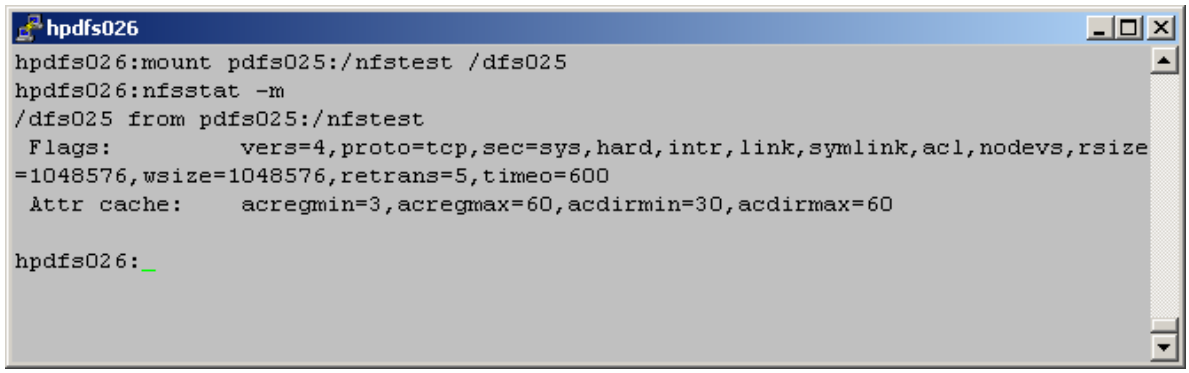
example, to enable your HP-UX 11i v3 client to use NFSv2, NFSv3, or NFSv4, set the configuration variables NFS_CLIENT_VERSMIN and NFS_CLIENT_VERSMAX to 2 and 4, respectively.

After the client has been restarted, NFSv4 mounts can be performed with a simple mount command:

```
mount server:/server_mount /client_mountpoint
```

If both the client and the server are configured to support NFSv4, then the remote file system is mounted using NFSv4. Otherwise, the client and server negotiate and accept the highest numbered protocol that both client and server support. To verify the NFS protocol in use, execute the `nfsstat` command with the "-m" option. Issuing this command displays the mount options, including the protocol version in use, as shown in Figure 14.

Figure 14. Implicit NFSv4 mount

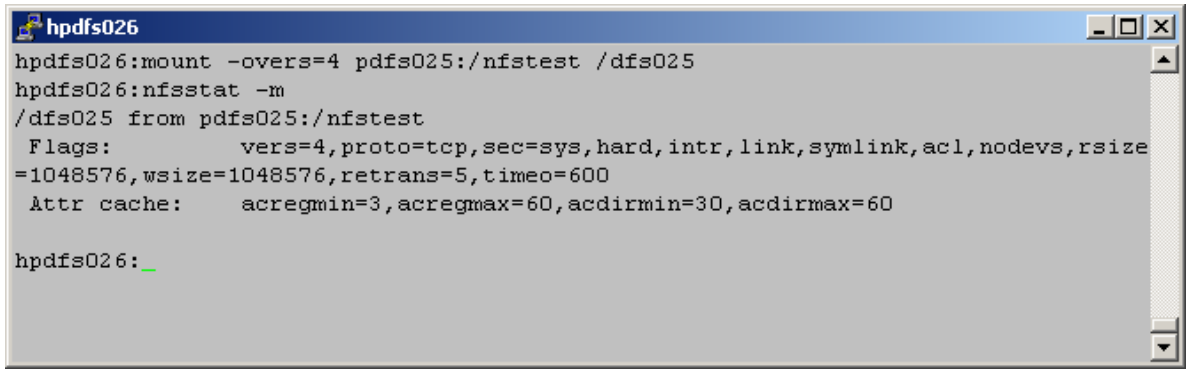


Alternately, the NFS protocol version can be specified explicitly on the command line with syntax:

```
mount –overs=4 server:/server_mount /client_mountpoint
```

If "vers=4" is specified explicitly and NFSv4 is not supported by the client or the server, then an error message appears and the mount request fails. Issuing the `nfsstat –m` command displays the NFS protocol version in use. See Figure 15.

Figure 15. Explicit NFSv4 mount



In either case, NFSv4 is shown in the output from the `nfsstat –m` command.

Additional Features in HP-UX 11i v3 ONCplus B.11.31.03

HP-UX 11i v3 ONCplus release 11.31.03 provides support for the following additional NFSv4 features:

- Cross Mounts
- Referrals
- Delegation Stackable Module for Local Access

Cross Mounts

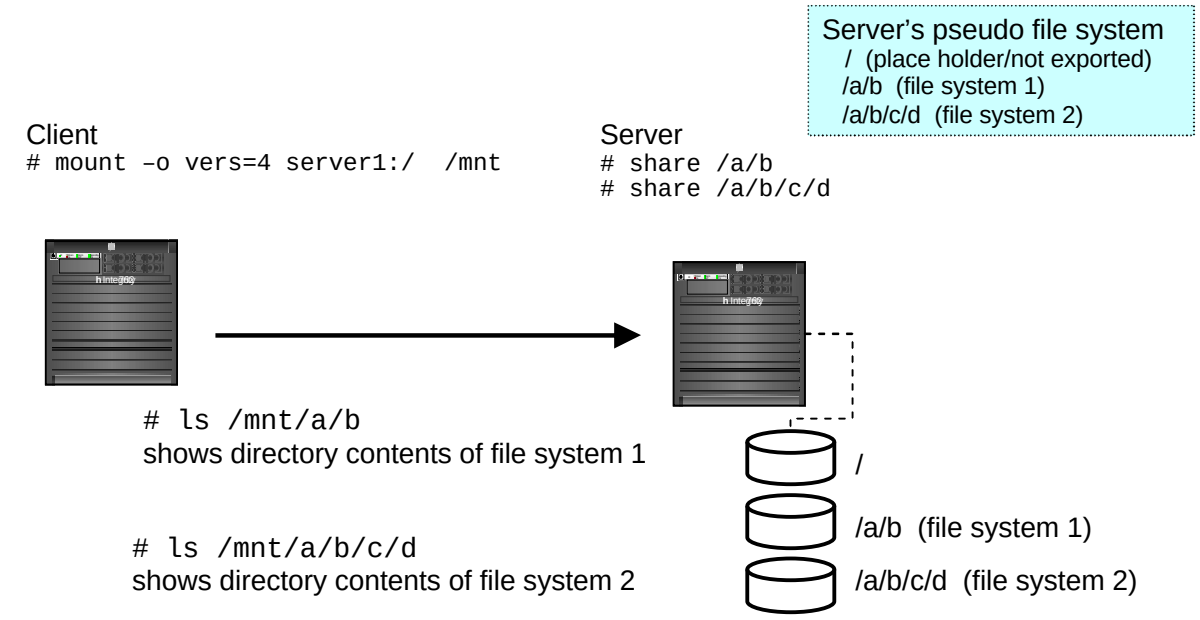
The NFSv4 protocol allows clients to seamlessly traverse the server's shared directories and cross the physical file system boundaries on the server without having to explicitly mount each shared file system independently. For example if the server is sharing the two file systems "/" and "/a/b" respectively, the client, after mounting the root file system of the server, can traverse the file system "/a/b" on the server without mounting the file system explicitly.

In prior releases an NFSv4 client was unable to traverse different file systems on the server by crossing the file systems' boundaries. The client was required to mount each of the server's shared file systems separately. HP-UX 11i v3 ONCplus version B.11.31.03 enables this functionality on the client. The client detects the crossing of the physical file system by checking the 'fsid' attribute returned by the server against the 'fsid' of the server's file system stored during the mount.

The functionality overview of the client crossing file systems on the server is shown in Figure 16. NFSv4 protocol does not require a MOUNT protocol to provide the initial mapping between path name and filehandle. The server provides a ROOT filehandle that represents the logical root of the file system tree of the server. In Figure 16, the server provides the pseudo filesystem with the root directory "/" as a place holder to represent the top of the file system tree and to fill in the gaps in the path names between real filesystems.

In Figure 16, the server has shared two file systems and the client mounts the server's root file system onto "/mnt" via NFSv4. With the cross mount functionality, the client is able to traverse, lookup and access the contents of both the server's file systems without mounting them separately. Note that the client can alternately mount the server's "/a/b" file system directly and then traverse into the "a/b/c/d" file system.

Figure 16. NFSv4 Cross Mount



Note that the NFSv2/NFSv3 clients will still need to mount the server's file systems separately to access the contents of both shared file systems.

Example of Cross Mounts

Figure 17 and Figure 18 illustrate the use of the Cross Mounts feature. The setup consists of:

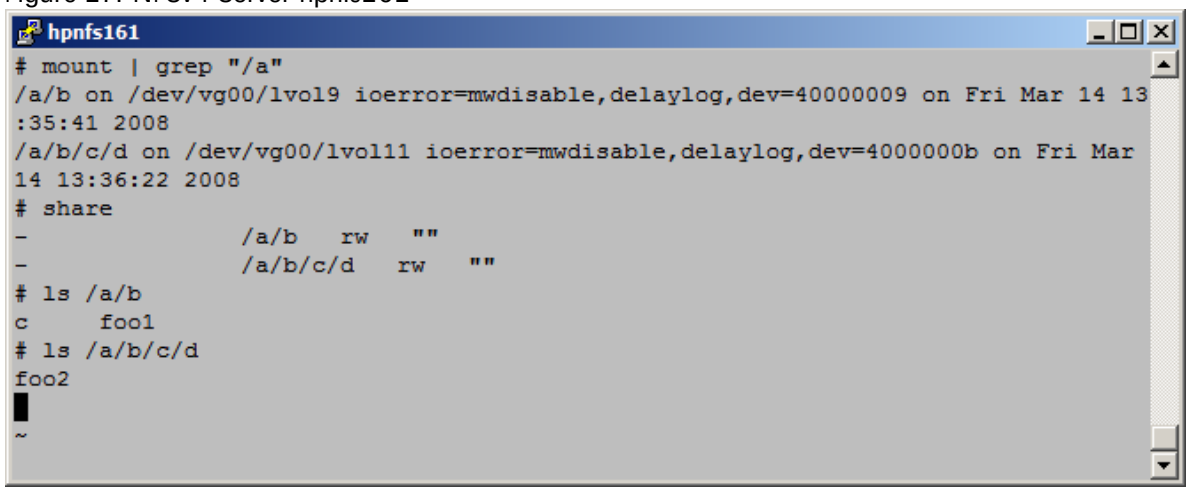
- NFSv4 Server: hpnfs161
  ```
  # share /a/b
  # share /a/b/c/d
  ```

where `/a/b` and `/a/b/c/d` are two different file system partitions as shown in Figure 17.

- NFSv4 Client: hpnfs163
  ```
  # mount  –o vers=4 hpnfs161:/ /mnt
  # cd /mnt/a/b; # ls
  # cd /mnt/a/b/c/d; # ls
  ```

Figure 17 shows the server setup and the directory contents of the two shared file systems.

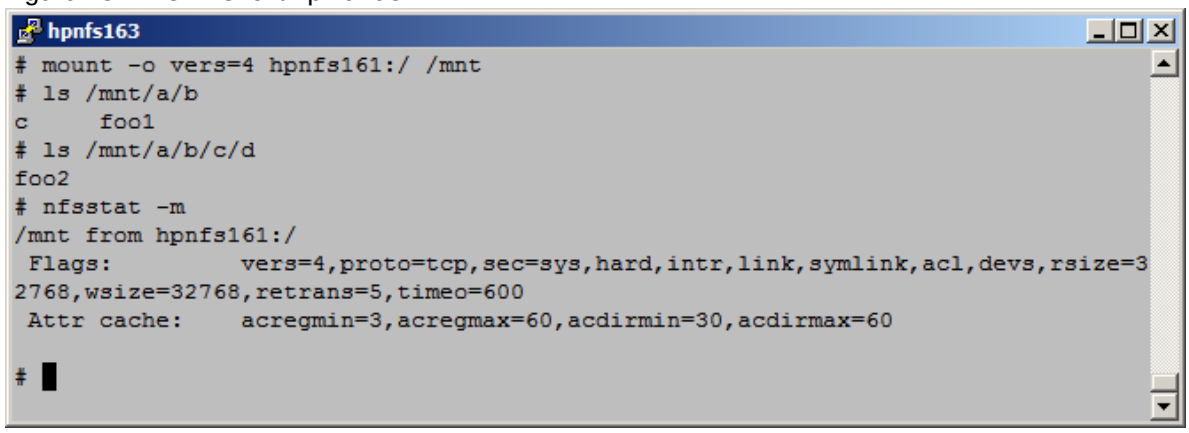
Figure 17. NFSv4 Server hpnfs161



In Figure 18, the ls command on directories "/mnt/a/b" and "/mnt/a/b/c/d" shows the contents of the exported file systems (hpnfs161:/a/b and hpnfs161:/a/b/c/d) respectively, illustrating the client is able to cross the server's file system mount points without having to mount each shared file system separately.

Since the server is not explicitly sharing the "c" portion of the "/a/b/c/d" directory structure, the NFSv4 server creates an empty, read-only placeholder at "/a/b/c" in order to bridge the gap between the "/a/b" and "/a/b/c/d" shared directories.

Figure 18. NFSv4 Client hpnfs163



Referrals

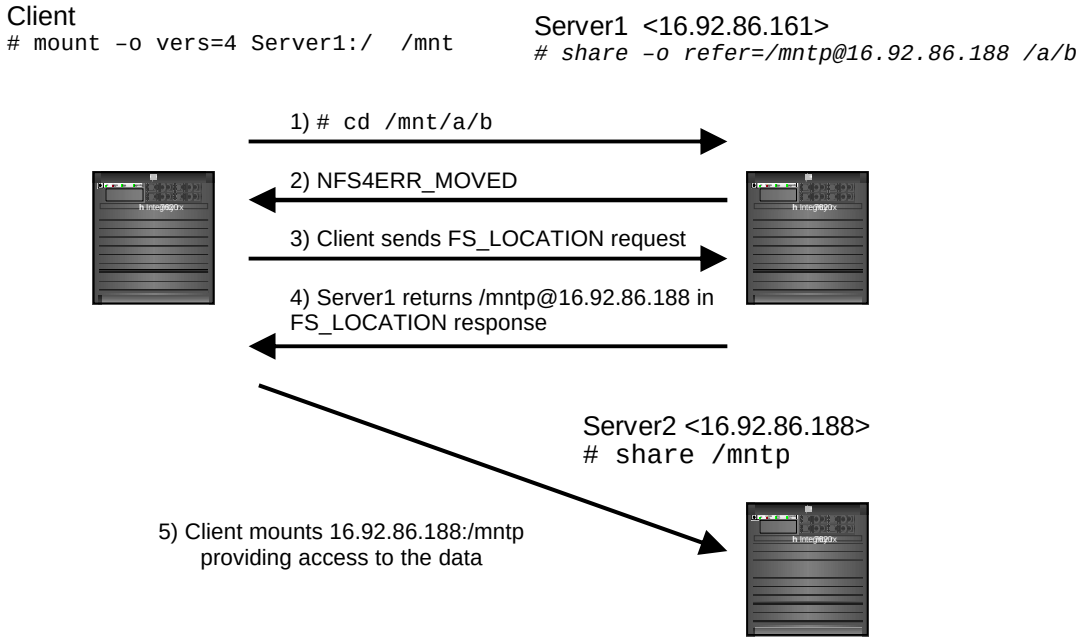
The Cross Mounts feature allows an NFSv4 client to traverse the server's shared directories and seamlessly cross the physical file system boundaries on the server. The Referrals feature allows an NFSv4 client to traverse shared directories and seamlessly cross the physical file systems located on different servers. In other words, a referral defines a way for the NFSv4 server to direct an NFSv4 client to a file system which resides on a different server. The combination of cross mounts and referrals can be used to construct a global namespace.

HP-UX 11i v3 ONCplus version B.11.31.03 enables the referral functionality on the client and the server. When a client tries to access a file system located on a different server, the NFSv4 server refers the client to the new server through a special error, NFS4ERR_MOVED, and the fs_locations attributes. The client automatically and seamlessly mounts to the new server to access the file system. The referral mount is transparent to the user and applications.

Figure 19 illustrates the referral functionality. As shown in the figure, Server1 is the referral server which is referring the file system "/a/b" to another server, Server2. The client attempts to mount a shared filesystem from Server1 via NFSv4. The following lists the sequence of steps as depicted in the figure:

1. Client tries to access the remote file system by executing the "cd /mnt/a/b" command.
2. Server1 returns NFS4ERR_MOVED for the lookup when it detects that the file system in the lookup entry ("/a/b") is a referral, and thus resides on a different server.
3. Client sends FS_LOCATION request to obtain the file system information of the referral.
4. Server1 responds with the referred server information in fs_locations attribute.
5. Client mounts to the referred server to obtain the initial file handle. The mount is performed inside the kernel and is transparent to the user.

Figure 19. NFSv4 Referrals



Server Support for Referrals

Server support for referrals is provided by the "refer" option of the share(1M) command. For example the following share command exports the "/mnt" file system, which is referring to the "/tmp" directory residing on the NFSv4 server whose IP address is 16.92.87.140.

```
# share –o refer=/tmp@16.92.87.140 /mnt
```

Syntax of share command

```
# share –o refer=<path>@<ipaddr> <dir>
```

Note:

1. The <dir> entry must be a file system partition or logical volume since clients recognize a change in the file handle when using referrals.
2. The <ipaddr> entry must be an IP address as client referral requests are not able to use hostname resolution mechanisms to resolve server hostnames.

Referrals can be defined with multiple referral hosts and multiple referral entries as:

```
# share –o refer=<path>@<ipaddr1>+<ipaddr2>+… <dir>
# share –o refer=<path1>@<ipaddr1>:<path2>@<ipaddr2> <dir>
```

The subsequent entries are used by the referral client when the previous entry fails to mount successfully.

Client Support for Referrals

Mounting of the referral file system on client is triggered by the lookup (ls or cd command) of the shared file system on the referred server. The following lists some important notes regarding the referral mounted systems on the client that users should be aware about:

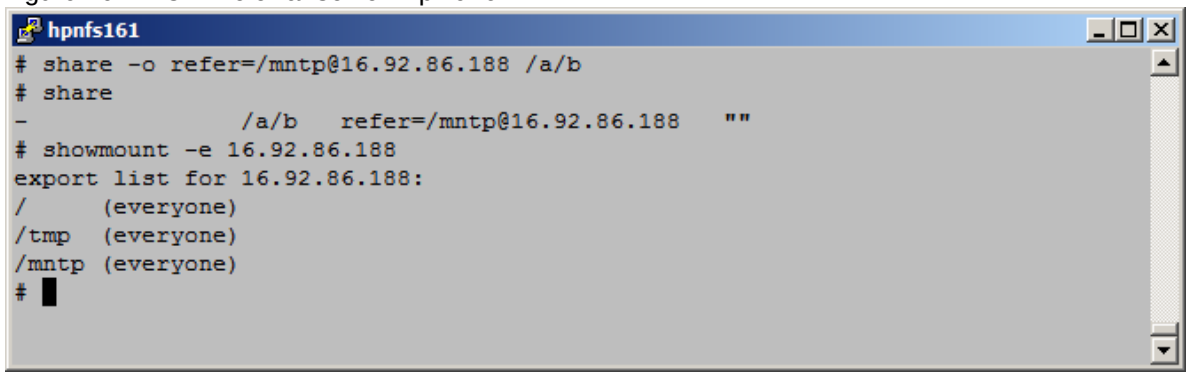
- Referral mounts and unmounts are automatic and transparent to the user.
- Referral mounted file systems will not show up in mount or bdf commands.
- nfsstat –m will show the mounted referral file systems.
- Only IP address may be used as the referral server since hostname resolution is not supported during referral mounts.
- Client allows referral IP addresses from the same address family only (i.e. IPv4 or IPv6).
- Client supports nested referrals where a referred file system location has objects that are referred to another remote server file system
- Client returns an error if it tries to directly mount to a referred location and the referred location is referred to another server.
- Once the referral file system is mounted, it will remain mounted unless server does an unshare or reshares the file system to another server or if it is unmounted explicitly.
- Resharing of the referral file system on the server is detected at the client when client updates the directory containing the referral file system or when the directory cache expires. HP-UX server will touch the parent directory when sharing a file system to indicate that the parent directory is changed.
- Client automatically unmounts the previously mounted referral file system and mounts to the new referral location when it detects a change to the referral location.

Example of Referral

Figure 20 and Figure 21 illustrate the Referral functionality. The figures use the following setup:

- NFSv4 Server1: hpnfs161 (Referring Server):
  `# share –o refer=/mntp@16.92.86.188 /a/b`
- NFSv4 Server2: hpnfs188 <16.92.86.188> (Referred-to Server):
  `# share /mntp`
- NFSv4 Referral Client: hpnfs163
  `# mount –o vers=4 hpnfs161:/ /mnt`
  `# cd /mnt/a/b`

Figure 20. NFSv4 Referral Server: hpnfs161



As shown in Figure 21, ls command on directory "`/mnt/a/b`" shows the contents of the exported file system of referred Server2 (hpnfs188:/mntp). Note that the "`nfsstat –m`" command shows the mounted referral system ("`mnt/a/b`").

Figure 21. NFSv4 Referral Client: hpnfs163

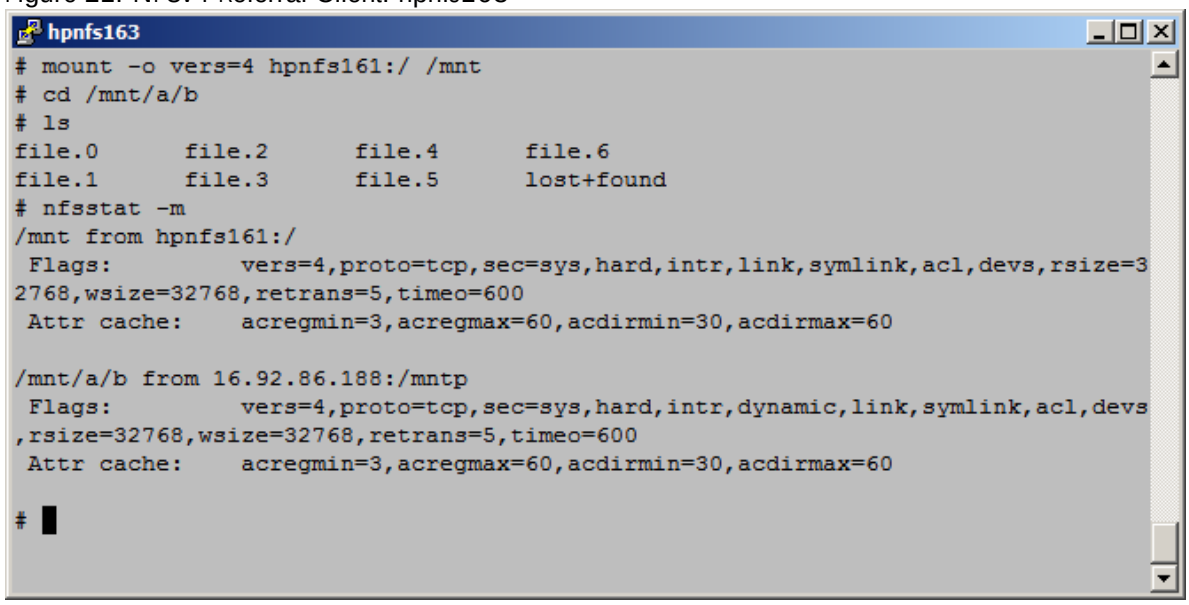


Figure 22 to Figure 24 show the network traces of the example of Figure 21.

Figure 22 shows the network trace of the LOOKUP response from the server when Client is executing the "cd /mnt/a/b" command. Line 7 shows Server1 returning the error status NFS4ERR_MOVED in the LOOKUP response. Lines 23 to 42 show the trace of FS_LOCATION request sent by the client upon receiving the NFS4ERR_MOVED error.

Figure 22. NFS4ERR_MOVED and FS_LOCATION request

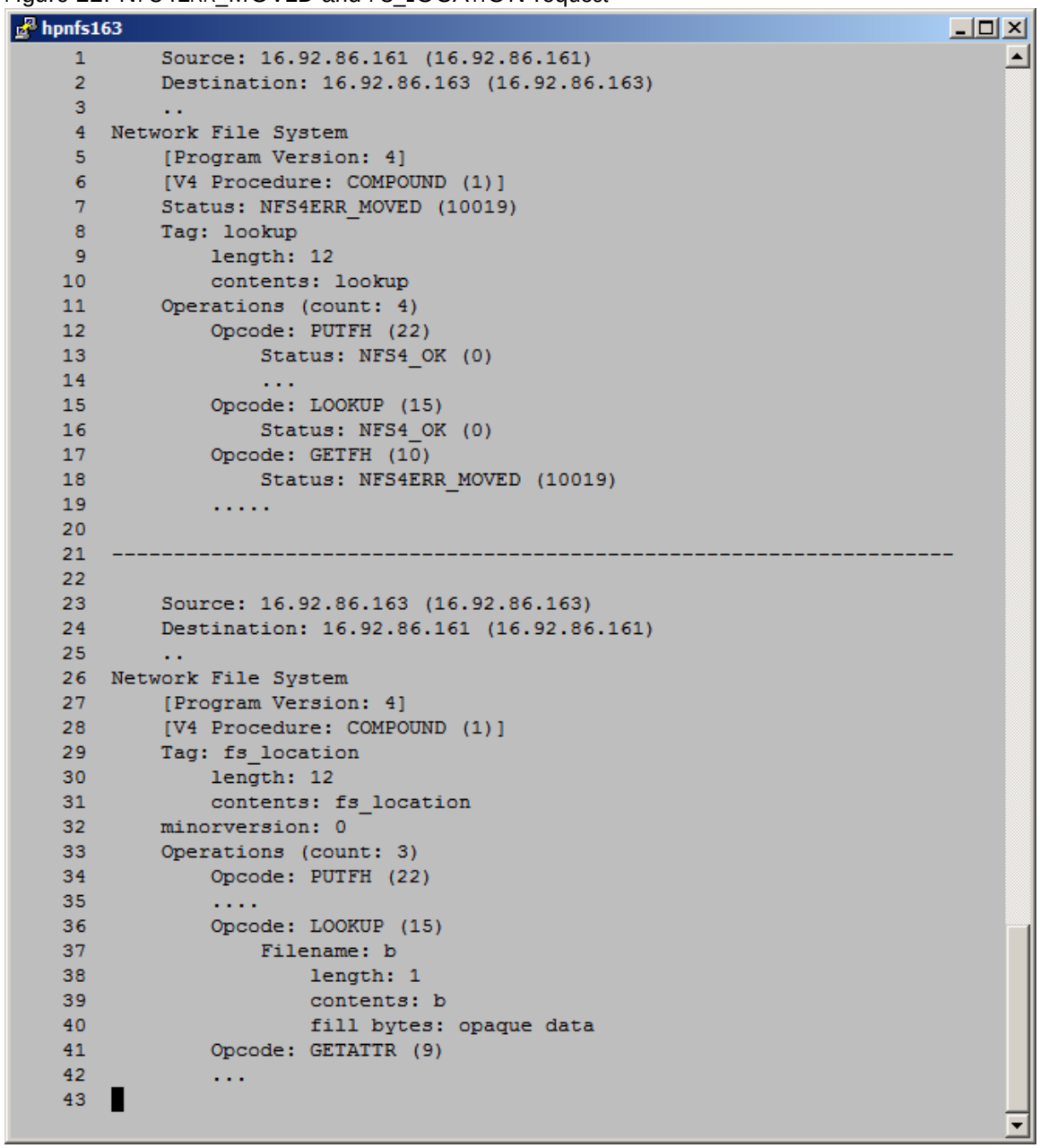


Figure 23 shows the network trace of the FS_LOCATION response from Server1. Lines 21 to 38 show the values in the fs_locations attribute.

Figure 23. FS_LOCATION Response

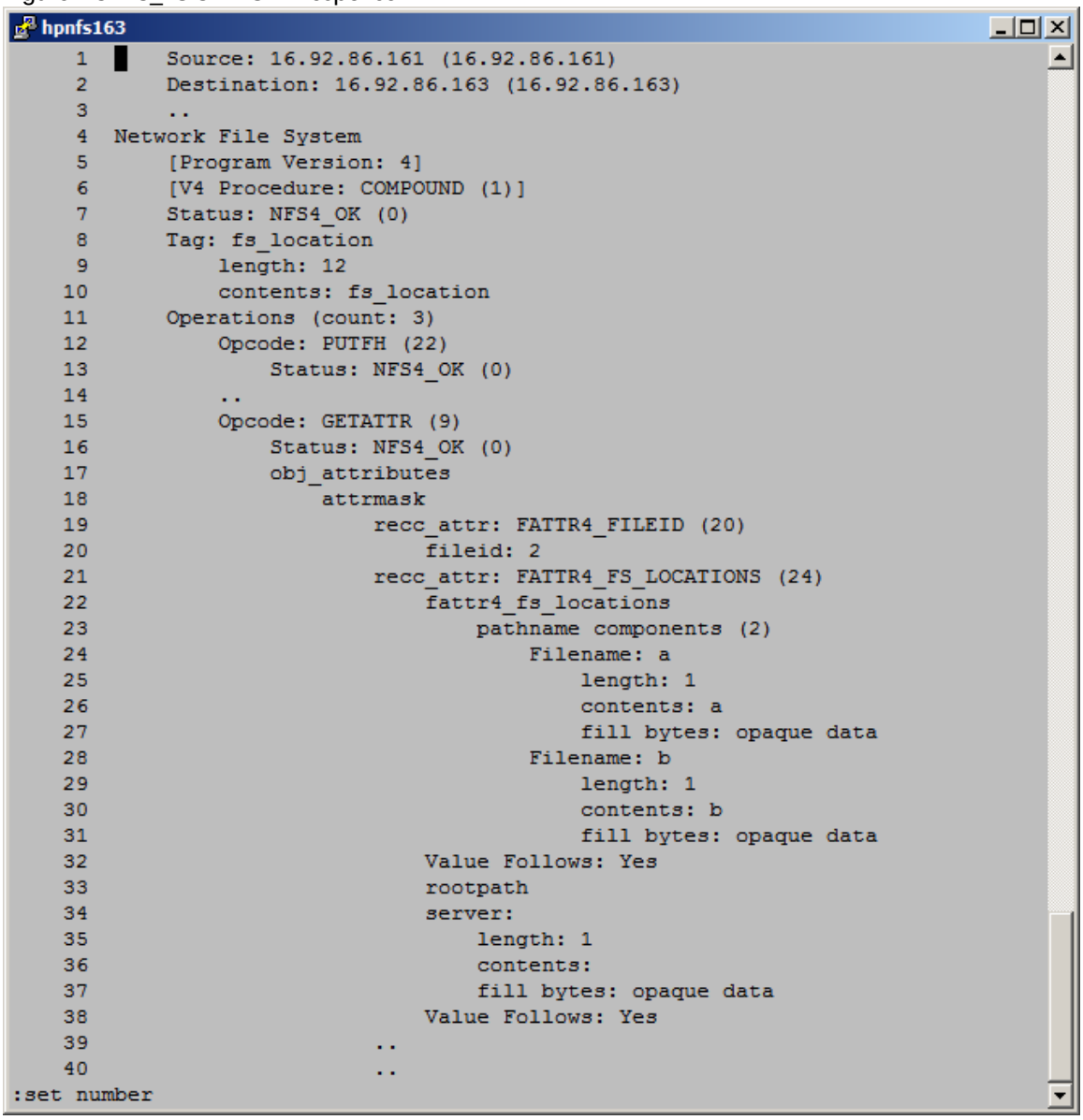
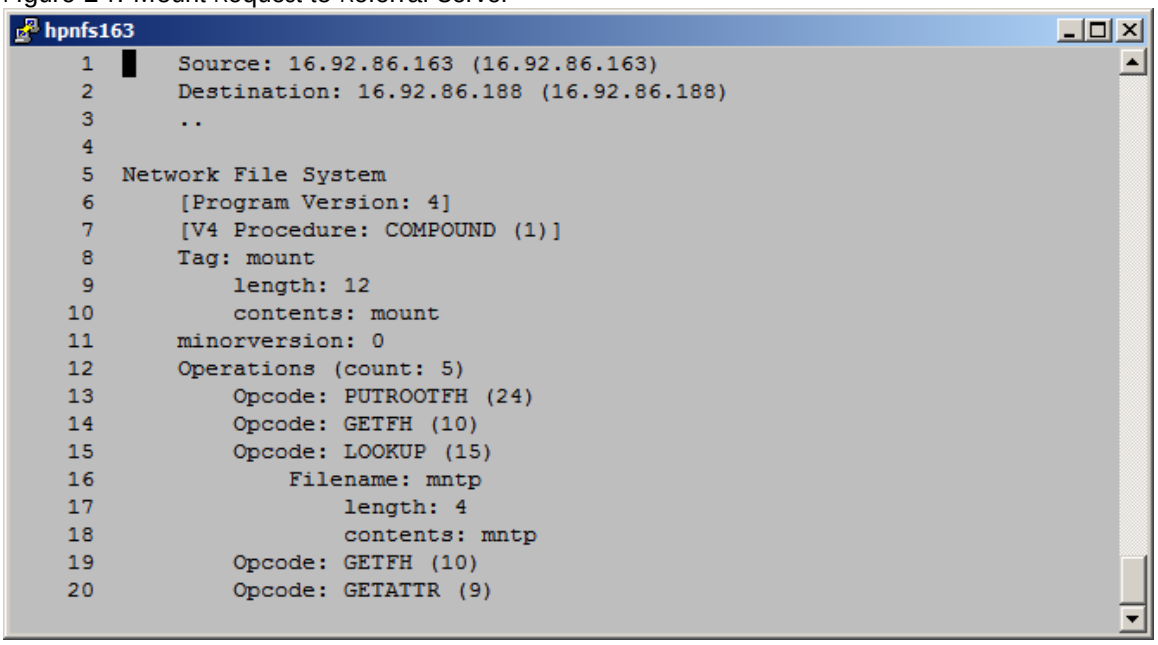


Figure 24 is the network trace of the client sending the mount request to Server2 after processing the FS_LOCATION response received from Server1. Note the destination address at line 2 is the IP address of Server2.

Figure 24. Mount Request to Referral Server



Delegation Stackable Module for Local Access

NFSv4 clients support delegation on HP-UX 11i v3. However, until the release of HP-UX 11i v3 ONCplus version B.11.31.03, NFSv4 servers support delegation with the caveat that no local file access should occur on any delegated file. For example, if the server grants a delegation to the file "/a/b/foo" then any local users on the server need to avoid accessing file "foo" while the delegation is in effect. If both local and remote users modify the delegated file then the data in "foo" could become corrupted.

With HP-UX 11i v3 ONCplus version B.11.31.03, support was added for local file access with delegations through a file system stackable module called nfs4deleg. This module provides a way to recall delegations issued by the NFS server when a file is modified locally on a machine by intercepting the local operation which triggers a callback to recall the delegation from the NFS client. The NFS client will write out locally cached data to the server before returning the delegation.

Using the nfs4deleg stackable module requires special configuration of the physical file system on the NFSv4 server.

- Create a volume and new file system to mount the stackable module.

  ```
  # lvcreate –L <size> <volgrp>
  # mkfs –F vxfs <volume created>
  ```

- Create a text file containing the nfs4deleg stackable module name.

  ```
  # echo "nfs4deleg" > <filename>
  ```

- Create a stackable module template using `fstadm(1M)`.

  ```
  # fstadm create –f <filename> –n <template>
  ```

  This will create a template with the name <template>

- Mount the newly created file system with the nfs4deleg stackable module.

  ```
  # mount –F vxfs –o stackfs=<template> <volume created> <directory>
  ```

- Share the new file system that has been mounted with the nfs4deleg module.

  ```
  # share <directory>
  ```

Figure 22. Example Setting Up Physical File system with nfs4deleg Stackable Module

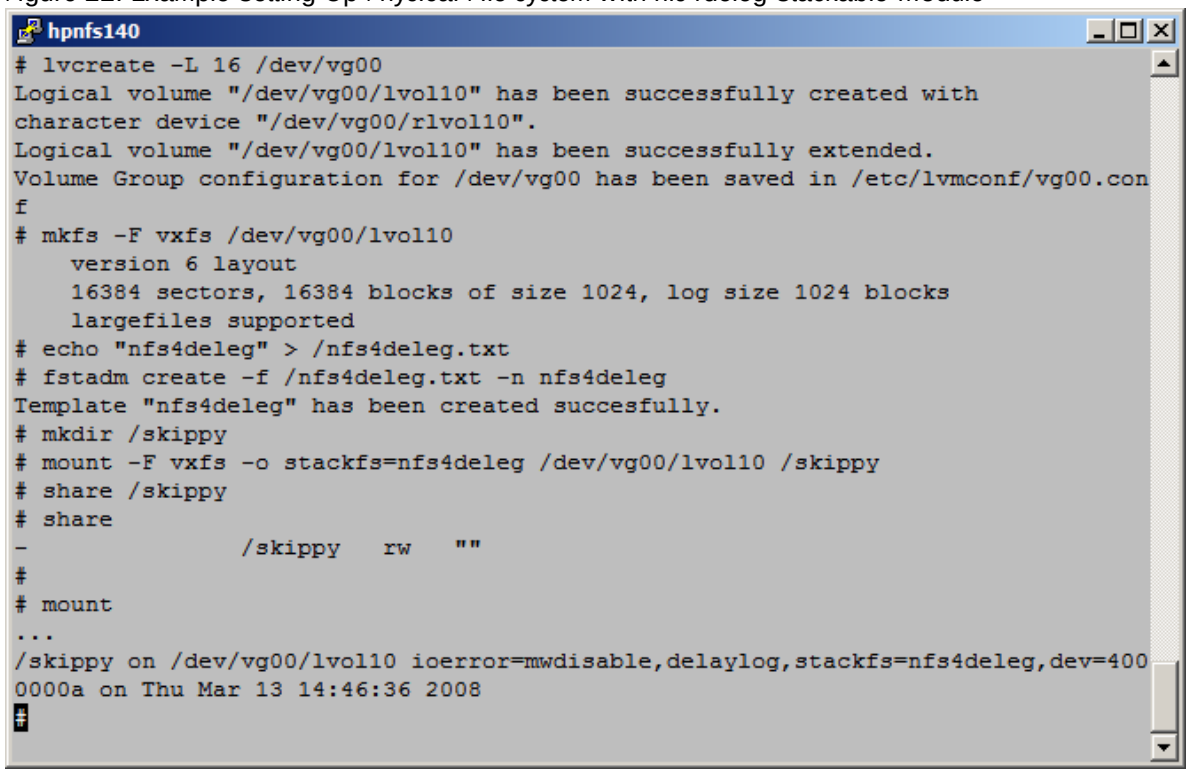


Figure 22 shows an example using the steps provided to create a mount with the stackable module. For convenience, once the physical file system is set up, the `stackfs=` mount option can be used in the `/etc/fstab` to mount the file system at boot time with the nfs4deleg module. For example:

```
/dev/vg00/lvol10  /skippy     vxfs  stackfs=nfs4deleg,delaylog 0 2
```

Once the physical file system is ready to use, the NFS server must be configured to support NFS version 4 and the delegation feature must be enabled.

- Setup NFS_SERVER_VERSMAX and NFS_SERVER_DELEGATION

  ```
  # setoncenv NFS_SERVER_VERSMAX 4
  # setoncenv NFS_SERVER_DELEGATION on
  ```

- Restart the NFS server

  ```
  # /sbin/init.d/nfs.server stop
  # /sbin/init.d/nfs.server start
  ```

Once the NFS server is restarted, the physical file system `/skippy` will be able to support delegations with local access. The stackable module is not supported on the root file system ("/") and is discouraged for use on `/stand`. All physical file systems that support NFSv4 delegations with local file access must have the nfs4deleg stackable module mounted through the stackfs= option.

In general, the delegation stackable module for local access does the following if an NFS delegation has been granted on the file:

1. The nfs4deleg module will intercept all vnode operations
2. If vnode operation does not issue a delegation recall, then directly call the physical file system

3. If the operation supports recalls and there is no delegation to revoke, pass the operation on to the physical file system
4. If there is a delegation that needs to be recalled, call the server function to recall the delegation
5. A callback is issued to the client where the client writes its data to server, updates the server state, and a DELEGRETURN will be issued. Local access is blocked on a delegated file until the delegation is returned or the lease times out.

Figure 23 is a diagram that shows an example of what happens when a delegation is recalled by the stackable module when an NFSv4 client has file "foo" open when a user on the local machine opens the same file. Once the delegation is recalled, the NFS client will flush the data to the server and the local user will have the most recent data that has been written out by the NFS client.

Figure 23. Recalling a Delegation on a Local File Open

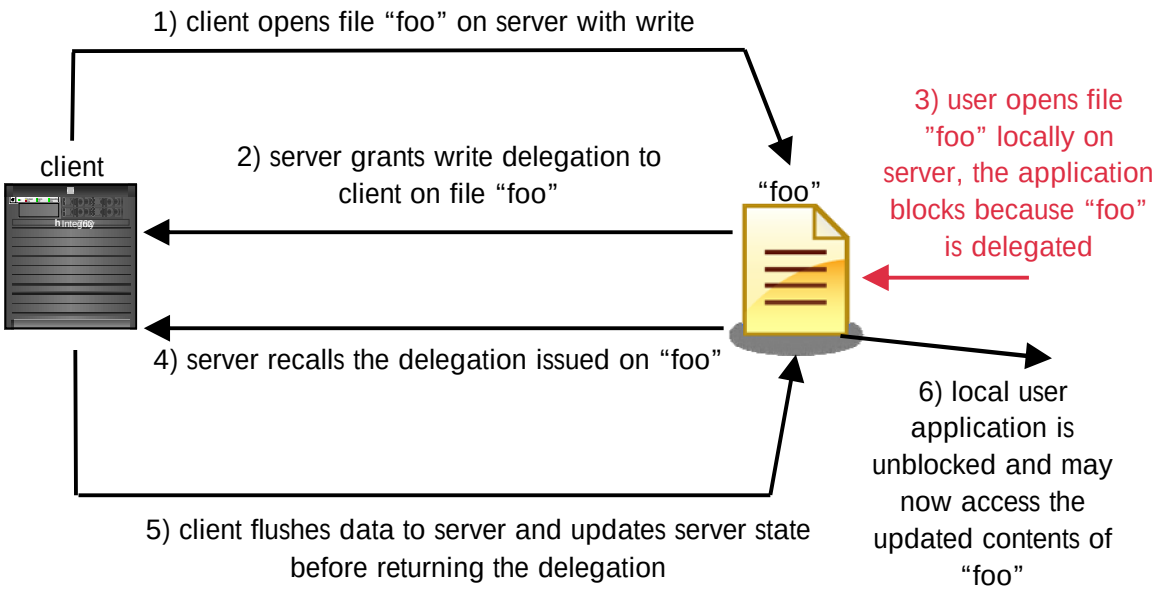


Figure 24 shows the network trace of the OPEN reply from the NFS server (Step 2 in Figure 23) where the write delegation (OPEN_WRITE_DELEGATE on line 16) has been granted to the NFS client for the file "foo".

Figure 24. NFSv4 OPEN opcode reply showing delegation granted

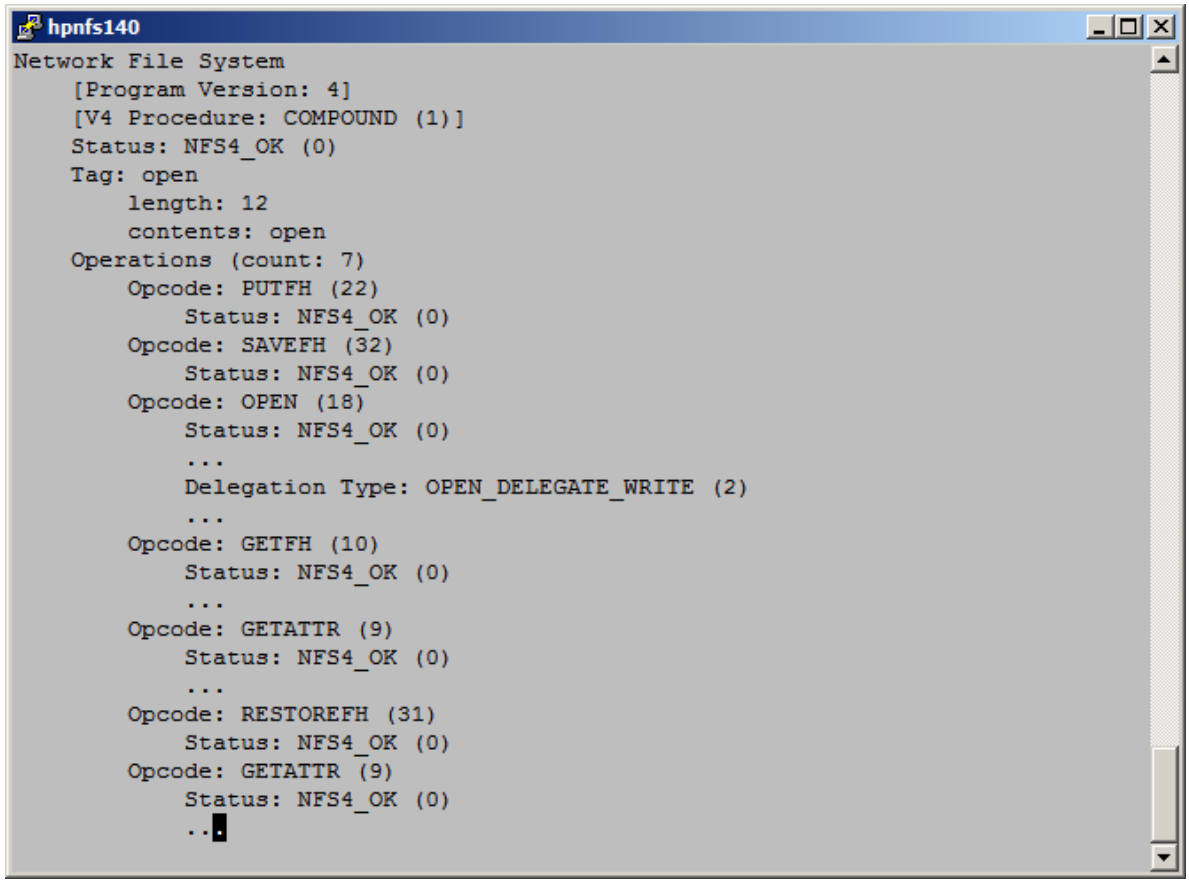
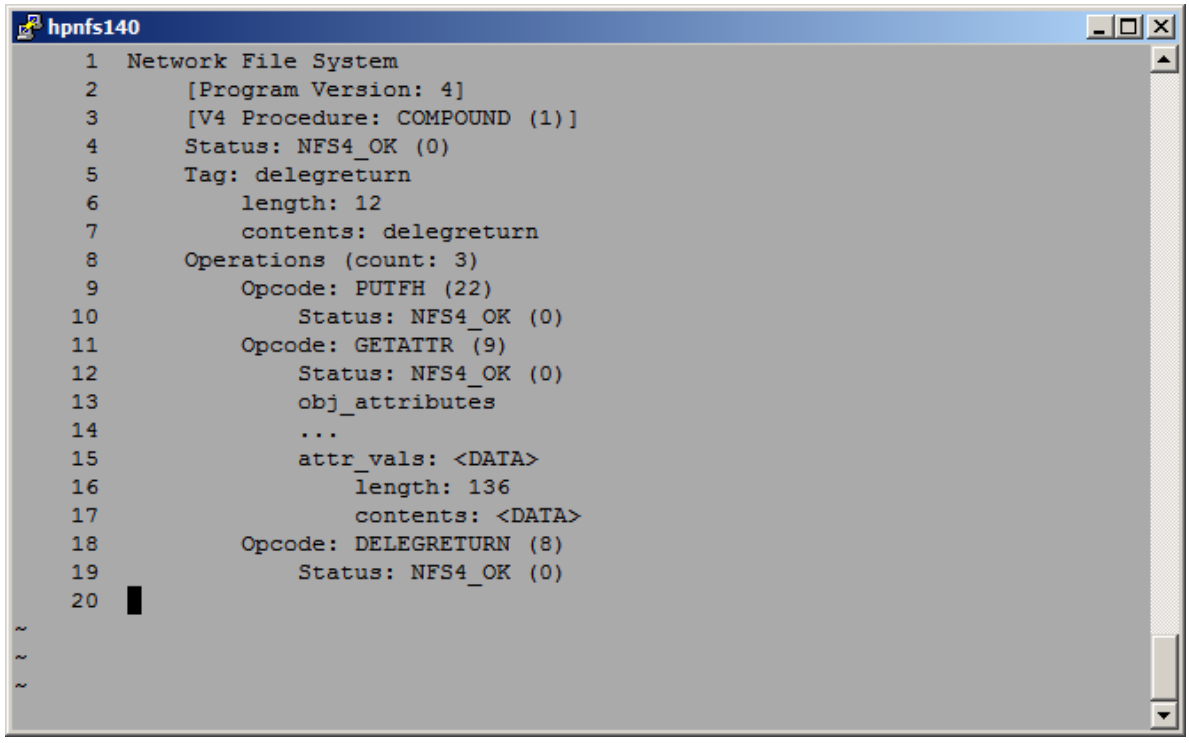


Figure 25 shows a network trace of a successful reply of the DELEGRETURN opcode from the NFS server (Step 5 in Figure 23).

Figure 25. NFSv4 DELEGRETURN opcode reply



Conclusion

The Network File System (NFS) has existed in one version or another since 1985. NFSv4 marks a big departure from previous NFS protocol versions because it introduces state into the NFS protocol and folds ancillary mechanisms such as ACLs, file locking, and mounting, into the core protocol. In addition, NFSv4 improves Microsoft Windows interoperability with an enhanced attributes mechanism, share locks, ACL support, and stronger security. Using this latest NFS protocol version provides high reliability and improved performance, as well as complete interoperability with systems that implement previous NFS protocol versions only.

With ONCplus version B.11.31.03, the introduction of features such as cross mounts, referrals, and delegation stackable module for local access provides a more complete NFSv4 offering on HP-UX 11i v3.

For more information

- http://www.hp.com/go/hpux11i
- http://www.hp.com/go/integrity
- http://www.ietf.org/rfc/rfc3530.txt

4AA0-7689ENW, April 2008