NFS Version 4

Technical Brief

Printed in the United States of America.
901 San Antonio Road, Palo Alto, California 94303 U.S.A

Please
Recycle

NFS Version 4

The NFS™ distributed file system is an enterprise solution that provides secure, high-performance, transparent access to information on worldwide, heterogeneous networks.

When the NFS protocol was designed, machines were far less powerful than today, networks were more commonly local area networks (LANs) than wide area networks (WANs), and available security mechanisms were relatively easy to exploit. Additionally, although NFS was designed to interoperate between different operating systems, many of the protocol features have favored UNIX® file semantics. This history has led to problems in using NFS solutions over the Internet, where security, performance, and interoperability are key. Version 4 of NFS is designed to address these concerns by providing:

- Improved access and good performance on the Internet
- Strong security, with security negotiation built into the protocol
- Enhanced cross-platform interoperability
- Extensibility of the protocol

Stronger in its support of intranets and the Internet, NFS Version 4 will also provide excellent service in environments currently standardized on earlier versions of NFS. Unlike versions 2 or 3, NFS Version 4 can be an important element in the strategy of enterprises to provide better support for global networks.

This document contains an overview of the features and benefits that are planned for NFS Version 4. It assumes the reader has a basic understanding of NFS, and includes a glossary and a reference list for those readers wishing to learn more.

Introduction

Worldwide, transparent access to information is a requirement for corporations wishing to compete successfully in today's global economy. As workgroups become distributed around the world, they demand access to file systems that are not dependent on geography. With the explosion of Internet-enabled applications, distributed file services must be able to function across the Internet as well as within private enterprise networks.

NFS on the Internet

Until now, there has been no reasonable solution to the lack of a functional distributed file system on the Internet. Historical constraints on NFS have limited the use of NFS over the Internet — it was designed for high bandwidth, low latency networks; its feature set was derived from the UNIX® operating system; and it lacks good security tools.

Despite these constraints, NFS is now the distributed file system of choice. Since its first introduction, NFS has continued to evolve, changing to meet the distributed file sharing requirements of global enterprises. With NFS Version 4, it is once again evolving to meet the demands of fast, reliable, secure, and transparent service over the Internet.

Sun's Commitment to Open Standards

Users are increasingly unwilling to pay the high costs of proprietary technology. Understanding this, Sun pioneered the concept of open systems in the early 1980s. Sun believes that open availability is critical to technological success. Key to this strategy are multiple vendors and implementations to foster competition and innovation, and broad technology licensing to ensure unencumbered access. Together, these methodologies increase the open availability of both specifications and implementations.

Sun believes that by employing basic, standardized technologies, more parties will develop, use, and innovate around those technologies, driving costs down. By deploying standardized technologies, implementers can reduce development costs and shorten time-to-market by leveraging open standards for distributed file sharing.

Sun is committed to working with the Internet Engineering Task Force (IETF) to increase the open availability of both the interface specifications and implementation of NFS Version 4. The IETF is an open international community concerned with the evolution of the Internet architecture. As such, it is the principal body engaged in the development of new Internet standards and procedures.

Sun is an active participant in the IETF working group that is chartered to advance the state of NFS technology by producing a specification for NFS Version 4. This public process ensures that the NFS Version 4 protocol is openly available, its specifications are well written and freely available, and the implementations based on those specifications are interoperable.

The NFS Version 4 Charter

The IETF's NFS Version 4 working group is creating a protocol definition for a distributed file system that focuses on improved access and good performance over the Internet, strong security with built-in negotiation, improved cross-platform interoperability, and will easily accept protocol extensions.

Although NFS Version 4 owes its general design to previous versions of NFS, it is a self-contained protocol that does not have any dependencies on the previous versions. However, to address backward compatibility with the installed base, Versions 2, 3, and 4 can be supported concurrently. Current users of NFS Versions 2 and 3 may look forward to NFS Version 4 as a reasonable, solid migration strategy to serve their Internet and intranet distributed file system needs.

At the time of the publication of this white paper, the NFS Version 4 protocol is still under development by the IETF, so, the final standard may not incorporate all of the features exactly as they are described here.

Improved Internet Access and Performance

The NFS protocol was originally designed for LANs where latency is low, bandwidth is high, and the number of round-trip requests is not a significant problem. As a consequence, NFS began to perform poorly when subjected to the low bandwidths, high latency, and heavy congestion typical of the Internet.

As more and more applications are being deployed over the Internet, it has become imperative that NFS not only be able to support distributed file systems, but easily support them through firewalls and scale more effectively. In addition, the global use of the Internet escalates the importance of international file naming conventions and multi-language file descriptors.

Several new features are planned for implementation in NFS Version 4. These include reducing turnarounds by combining requests, support of file caching, use of a well-known port for NFS services, and support for a universal character set.

Better Performance on High Latency Networks

Compound Requests

NFS traditionally has provided good throughput for the reading and writing of file data. This throughput is commonly achieved by the client's use of pipelining or windowing multiple Remote Procedure Call (RPC) READ/WRITE requests to the NFS server. The flexibility inherent in the NFS and RPC protocols has allowed implementations to provide efficient use of network connections.

However, the large number of procedure calls required to accomplish some file system operations, combined with high-latency networks like the Internet, have led to sluggish single-user or single-client response times. By decreasing the absolute number of procedure calls, NFS Version 4 will reduce the amount of server processing overhead required, and lower the time spent on the client side waiting for the server's individual responses.

This reduction is accomplished by supporting compound requests, which group multiple procedure calls into a single traditional RPC request, decreasing the number of round trip messages required between client and server. The grouping utilizes a new COMPOUND procedure that can be used as a wrapper by all other procedures. Combined requests reduce protocol-induced latency, shrink transport and security processing overhead, and allow the client to complete more complex tasks by combining procedures into a single RPC request. The COMPOUND procedure will lead to great performance improvements on high-latency WANs without detracting from today's LAN environments.

Client File Caching

In order to speed response times and reduce overall network and server loads, NFS clients have always cached file and directory data. This was accomplished by using a memory cache until recently, when local disk caching was added.

Because it decreases overall response times, aggressive client-side caching is very visible to the end user. With more aggressive protocols, it is possible to cache or delay certain protocol requests on the client, further reducing the protocol traffic between client and server. A client can request that the server delegate the client responsibility for changes to file attributes and data, or file locks. If there are no ensuing conflicts with other clients for file access, the client is then free to make any changes to the file without the need for contact with the server, thereby reducing the amount of file locking traffic generated by user applications.

Client caching is increasingly important for Internet environments with limited bandwidth and correspondingly long response times. NFS Version 4 will support a variety of caching protocols as exemplified by current technologies.

Easy, Secure Transmission Through Firewalls

The firewalls that protect corporate networks from Internet attackers have been a barrier to the efficient use of NFS on the Internet. Firewalls filter network packets to determine whether to forward them to their destinations. Packet filtering firewalls are relatively easy to configure for protocols that use well-known port addresses. In NFS versions 2 and 3, an NFS client could not communicate at all with an NFS server unless it first obtained an initial file handle using another RPC service: the MOUNT protocol. Since MOUNT does not have a well-known port, the server's portmapper was used to dynamically assign a port for MOUNT. This meant that the port number would change each time the MOUNT program was started. Although some sophisticated firewalls can track these port negotiations, it is not a common feature.

NFS Version 4 will overcome this limitation by not using the MOUNT protocol for translation from string-based path names to a file handle. Instead, two special file handles will be used as a starting point for the NFS client: the ROOT file handle and a public file handle. (See the *File Handle* section).

Historically, NFS Version 2 and 3 servers have resided on port 2049. Port 2049 is an IANA registered port number for NFS, and will continue to be used for NFS Version 4. Using this well-known port for NFS traffic means that NFS

clients do not need to use RPC binding protocols, and firewalls can easily be configured for NFS traffic. With the stronger security and authentication of ONC™ RPC's RPCSEC_GSS, NFS Version 4 provides a more secure and efficient solution for use with firewalls.

Internationalization

In the NFS protocol, strings are used for file and directory names and symbolic link contents. The string representations in NFS versions 2 and 3 are limited to either seven-bit U.S. ASCII or eight-bit ISO Latin 1 character sets. There is no mechanism to tag the character strings that indicate which encoding is used by either the client or the server. The U.S. ASCII and ISO Latin 1 representations do not support all languages, limiting the usefulness of NFS in environments where clients utilize different character sets.

The possibility of using local character sets presents a problem when strings are viewed in differing locales. For example, a user creates a file with a name in the Swedish character set. Later, when a different user with a U.S. ASCII locale within a given pathname attempts to view the file name, it looks very different because the local character representation has changed. This problem becomes quite complex when various components within a pathname are created using differing locales, or when users in various locales modify components within a given pathname that were originally created in a different language.

However, NFS Version 4 uses a universal character set, so it doesn't matter if the user accessing the file has a different locale than the user who created the file. This universal encoding of the character makes it possible to determine what language the character is from and how to display it on the client. The server does not need to associate a locale with a pathname.

In NFS Version 4, the universal encoding is UTF-8, a Universal Character Set (UCS) that uses an eight-bit, octet transformation format. UTF-8 was chosen because it compactly encodes 16- and 32-bit characters, is an efficient encoding for wire transfers, has room to expand beyond characters longer than 31 bits, and provides for the direct encoding of existing stored objects that are described with seven- or eight-bit characters.

Strong Security

A primary goal of NFS Version 4 is to provide for strong security services, including authentication, integrity, and privacy; be independent of specific security mechanisms; and rely on existing standards for implementation. Previous versions of NFS have been severely limited by the weaknesses of available security protocols and the lack of their widespread implementation. By providing a security implementation within the NFS Version 4 protocol, users will be able to use strong security if their environment or policies warrant it.

Limitations of Previously Available Protocols

The security provided by previous versions of NFS are inadequate and relatively easy to exploit. They reply on security flavors, described in Table 1, that have been called authentication flavors — either because integrity and privacy were not seen as requirements, or the computational overhead was considered prohibitive. The authentication flavors vary in their capacity to fight exploitation. With AUTH_SYS, t is relatively easy to spoof IDs. AUTH_DH employs a Diffie-Hellman 192-bit, public key modulus that is too small for reasonable security. AUTH_KERB4 is better, but lacks a written specification.

Today, there is no fundamental reason to restrict the services performed by new security flavors. However, since integrity and privacy were not used, RPC client data formats were the same for all flavors. Additionally, existing security flavors embody different security mechanisms, and each flavor has its own API. It is likely that the number of security flavors will increase, as no single solution can satisfy the unique requirements of every distributed network environment. Therefore, NFS Version 4 chose a solution with the flexibility to support multiple authentication solutions in a standard, consistent way.

AUTH_NONE	Null authentication, or no security information passed
AUTH_SYS	UNIX-style user identifier, group identifier, and an array of supplemental group identifiers
AUTH_DH	DES-encrypted authentication parameters based on a network-wide string name, with session keys exchanged via the Diffie-Hellman public key scheme
AUTH_KERB4	DES-encrypted authentication parameters based on a network-wide string name (a Kerberos version 4 principle identifier) with session keys exchanged via Kerberos version 4 secret keys

Table 1 Authentication technologies available to NFS versions 2 and 3

A New Security Protocol: RPCSEC_GSS

Conforming implementations of NFS Version 4 employ the standards track ONC™ RPC security flavor, RPCSEC_GSS (RFC 2203), which allows RPC protocols to access the Generic Security Services Application Programming Interface (GSS-API). Using the GSS-API (standards track RFC 2078) allows for the use of varying security mechanisms by the RPC layer without the additional implementation overhead of adding RPC security flavors (Figure 1.) Although the RPCSEC_GSS security flavor was available to previous versions of NFS, it was not widely deployed.

The basic services offered by the GSS-API are integrity and privacy. For the integrity service, the GSS-API uses the underlying mechanism to authenticate messages exchanged between applications, while cryptographic checksums establish identities and the authenticity of the transmitted data. The privacy service includes the integrity service and also encrypts transmitted data to protect it from eavesdroppers.

RPCSEC_GSS is a single security flavor over which different security mechanisms can be used. Examples of security mechanisms include Kerberos V5, RSA public key, and PGP. Within a mechanism, the GSS-API provides for the support of multiple qualities of protection (QOPs), which are pairs of cryptographic algorithms. The QOP parameter determines the cryptographic algorithms to be used in conjunction with the integrity or privacy service.

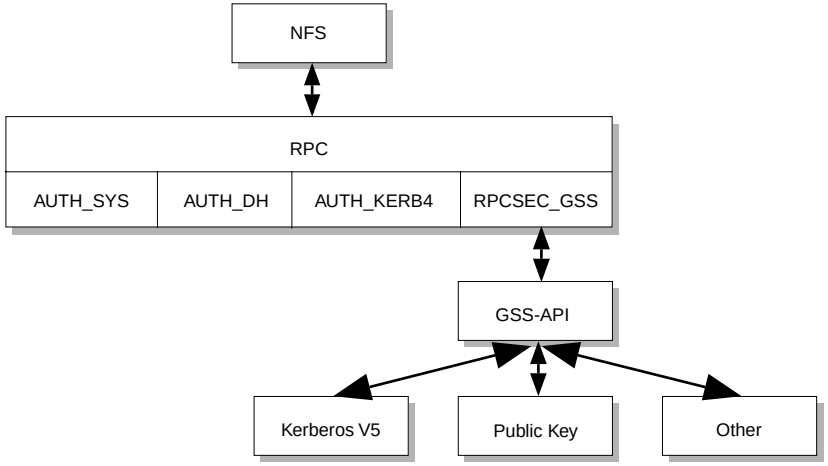

Figure 1 Using the GSS-API allows for the use of varying security mechanisms by the RPC layer without the additional implementation overhead of adding RPC security flavors.

The use of RPCSEC_GSS requires the selection of security mechanism, QOP, and service, which includes authentication, integrity, and privacy. These three parameters are known as the security triple.

Using the programming interface for the RPCSEC-GSS security flavor, NFS Version 4 specifies a security mechanism to be used on an RPC session, and requests desired security services (integrity, privacy, or none) for each RPC exchange. Security mechanism names and QOP values are passed through the RPCSEC_GSS layer to the GSS-API layer.

Since the RPCSEC_GSS security flavor uses the GSS-API interface, in the future any security mechanism that is supported by the GSS-API can be made available to the NFS protocol. RPCSEC_GSS is also extensible in that it provides for both public and private key security mechanisms as well as the ability to plug in various mechanisms without significantly disrupting ONC RPC or NFS implementations.

User Identification

Previous versions of NFS have been limited to the use of the UNIX user and group identification (UID and GID) numbers as a user identification mechanism. This has limited the capability for NFS to scale beyond large work groups, support string-based identification schemes, or scale to the Internet with respect to multiple naming domains and naming mechanisms.

The permission models in NFS Version 4 scale beyond the flat integer UID space of previous versions by using string-based user identifications. This permits integration into an external naming service or services and enables servers and clients to translate between the external string representation to a local internal numeric or other identifiers that match internal implementation needs.

Security Negotiation

With the new ability to provide multiple security mechanism choices, NFS Version 4 offers the client a security negotiation process to determine which mechanism should be used for communicating with the server. Within a given server implementation, multiple security mechanisms may be deployed across various file system resources. Therefore, the security negotiation process also takes into account the possibility of a change in policy, as an NFS client crosses certain file system boundaries on the server. Additionally, this negotiation is done over a secure channel to eliminate the possibility of third-party interception and resulting negotiation down to a lower level of security than required or desired.

The new procedure, SECINFO, allows NFS Version 4 clients to determine, on a per-file-handle basis, what security mechanism will be used for server access. In general, the NFS client does not have to use the SECINFO procedure except during initial communication with the NFS server or when the client crosses a policy boundary on the server. It is possible that the server's security policies could change during the client's interaction, forcing the client to negotiate a new security triple.

In summary, NFS Version 4:

- Supports multiple underlying security mechanisms and provides access to their services through a common API
- Does not impose excessive or complex administration on security-minded applications
- Is flexible enough to enable a variety of security requirements, while still being easy to use

Better Cross-platform Interoperability

NFS protocols are available for many different operating environments. However, due to its UNIX origin, the NFS protocol was difficult to implement in some environments. NFS Version 4 fixes this by creating a common set of features that do not favor one operating system over another.

For example, NFS versions 2 and 3 do not provide file and directory attributes beyond those found in the traditional UNIX environment. These include persistent file handles (unique identifiers of file system objects), UNIX uid/gid (user-ID/group-ID) mappings, directory modification times, accurate file sizes, and file and directory locking semantics.

To meet the requirements of extensibility and increased interoperability with platforms other than UNIX, NFS Version 4 introduces a new type of file handle, better file-attribute handling, a server namespace, and file locking mechanisms.

File Handles

In the NFS protocol, a file handle is a unique identifier for a file system object on a server. The contents of this handle are opaque to the client, so the server is responsible for translating the file handle into an internal representation of the file system object. Since the file handle is the client's only reference point to an object on the server, and the client may cache this reference, the server should not reuse the file handle for another file system object.

In NFS versions 2 and 3, the ancillary MOUNT protocol obtains the initial file handle when a filesystem is first mounted. As the MOUNT protocol is deficient with respect to firewalls (see *Secure Transmission Through Firewalls*, page 5), the use of a public file handle was introduced. Using a public file handle in

combination with the LOOKUP procedure in version 2 and 3 demonstrates that the MOUNT protocol is no longer necessary for viable NFS client and server interaction.

NFS Version 4 introduces two special file handles that are to be used as starting points for NFS clients, and two file handle types. The first is the ROOT file handle, the conceptual root of the file system name space on the NFS server. The second is a public file handle that can be used to bind or represent an arbitrary file system object on the server. Although the public and root file handles may not refer to the same system object, it is up to the server to define the bindings of the handles.

Versions 2 and 3 use the same type of file handle with a single set of semantics. This type relies on a unique, persistent identifier supported by UNIX but not by some other file systems. NFS Version 4 introduces two types of file handles that accommodate other server environments. The first type, persistent, has the same semantics as file handles in Versions 2 and 3. The second type is the volatile file handle.

Some server environments simply do not provide a file system invariant that can construct a persistent file handle. The volatile type addresses server functionality or implementation issues that prevent correct or feasible implementation of a persistent file handle. This eases the implementation of products such as hierarchical storage management systems and helps maintain uptime during file system reorganization and similar processes.

Since the client has different logic paths to handle persistent and volatile file handles, NFS Version 4 defines a new file attribute that determines what type of file handle is being returned by the server.

File Attributes

NFS Version 3 contains a fixed list of file attributes that is not supported by all clients and servers, had no mechanism to indicate non-support, and could not be extended as new needs arose. To increase interoperability, attributes must be handled in a more flexible manner. With NFS Version 4, the NFS client has to query the server about the attributes it supports, and be able to request only needed attributes.

The proposed attributes for NFS Version 4 are divided into three groups: mandatory (Table 2), recommended (Table 3), and named. Both mandatory and recommended attributes are supported in the protocol by specific, well-defined

encoding and semantics. Named attributes allow a client to name, store, and retrieve arbitrary or extra data and associate it as an attribute of a file or directory. This grouping of attributes, plus indicators of non-support, makes it possible to add new mandatory or recommended attributes to the NFS protocol between revisions.

The set of attributes classified as mandatory by NFS Version 4 is deliberately small, as servers must do whatever it takes to support them. Attributes are deemed mandatory only if the data is needed by a large number of clients and is not otherwise reasonably computable by the client when the server does not provide support.

Name	Description
supp_attr	Bit vector that retrieves all mandatory and recommended attributes
object_type	Type of the object (file, directory, symbolic link)
persistent_fh	Determines if the file handle for this objects persistent
change	Server created value flags if file data, directory contents, or attributes have been modified. Necessary for useful caching
object_size	Size of the object in bytes
link_support	Determines if the object's file system supports hard links
symlink_support	Determines if the object's file system supports symbolic links
named_attr	Determines if the object has named attributes
fsid.major	Unique identifier for the file system holding this object
fsid.minor	Identifier within the fsid.major file system identifier for the file system holding this object

Table 2 Mandatory attributes in NFS Version 4.

Name	Description
ACL	Access control list for this object
archive	Determines if the file been archived since it was last modified
cansettime	Determines if this object's file system can fill in the times on a SETATTR request without an explicit time

Table 3 Possible recommended attributes in NFS Version 4.

Name	Description
case_insensitive	Determines if file name comparisons on this file system are case insensitive
case_preserving	Determines if file name case is preserved on this file system
chown_restricted	Determines if a request to change ownership will be honored
filehandle	File handle of this object
fileid	Number uniquely identifying the file within the file system
files_avail	File slots available to this user on the file system containing this object
files_free	Free file slots on the file system containing this object
files_total	Total file slots on the file system containing this object
hidden	Determines if the file is considered hidden
homogeneous	Determines if this object's file system is homogeneous (i.e. if the pathconf is the same for all objects)
maxfilesize	Maximum supported file size for the file system of this object
maxlink	Maximum number of links for this object
maxname	Maximum filename size supported for this object
maxread	Maximum read size supported for this object
maxwrite	Maximum write size supported for this object
mime_type	MIME body type/subtype of this object
mode	UNIX-style permission bits for this object (deprecated in favor of ACLs)
no_trunc	If a name longer than name_max is used, determines if an error will be returned or the name truncated
numlinks	Number of links to this object
owner	String name of the owner of this object
owner_group	String name of the group of the owner of this object
quota_hard	Number of bytes of disk space beyond which the server will decline to allocate new space
quota_soft	Number of bytes of disk space at which the client may choose to warn the user about limited space
quota_used	Number of bytes of disk space occupied by the owner of this object on this file system.
rawdev	Raw device identifier

Table 3 Possible recommended attributes in NFS Version 4.

Name	Description
space_avail	Disk space in bytes available to this user on the file system containing this object
space_free	Free disk space in bytes on the file system containing this object
space_total	Total disk space in bytes on the file system containing this object
space_used	Number of file system bytes allocated to this object
system	Determines if this file is a system file
time_access	The time of the last access to the object
time_backup	The time of the last backup of the object
time_create	The time of the creation of the object
time_delta	Smallest useful server time granularity
time_metadata	Time of the last meta-data modification of the object
time_modify	Time since the epoch of the last modification to the object
version	Version number of this document
volatility	Approximate time until next expected change on this file system, as a measure of volatility

Table 3 Possible recommended attributes in NFS Version 4.

NFS Server Namespace

On a UNIX server, the namespace describes all the files reachable by the pathnames under the root directory "/". On a Windows NT server, the namespace consists of all the files on disks that are named by mapped disk letters. NFS server administrators rarely make the entire server's file system namespace available to NFS clients, but typically make pieces of the namespace available via an export feature.

The NFS Version 4 protocol provides a root file handle that clients can use to obtain file handles for these exports. A common experience is to use a graphical user interface (i.e. an Open dialog window) to find a file via progressive browsing through a directory tree. Clients must be able to move from one export to another export through progressive, look-up operations.

This style of progressive file browsing through directory trees was not supported well in NFS versions 2 and 3. A client application expects all look-up operations to remain within a single server file system. NFS versions 2 and 3 prevented a client from specifying namespace paths that spanned exports.

NFS Version 4 presents all exports within the framework of a single server namespace. Portions of the server namespace that are not exported are bridged via a pseudo file system that views only the exported directories. The pseudo file system behaves like a normal, read-only file system.

DOS, Windows 95, 98, and NT are sometimes described as having multiple roots, where file systems are commonly represented as drive letters. MacOS represents file systems as top-level names. NFS Version 4 servers will be able to construct a pseudo file system above root names on these servers, so drive letters or volume names are simply directory names in the pseudo-root.

The server's pseudo file system is a logical representation of the file system or systems available on the server. It is expected that the pseudo file system may not have an on-disk, physical counterpart from which persistent file handles can be constructed. Therefore, volatile file system handles are used.

Even if a server's root file system is exported, a pseudo file system may be needed. For example, in figure 2, "/a" lies on an unexported drive between the exported directories "/" and "/a/b". Because "/a" is not exported, "/a/b" cannot be reached by browsing from "/" via standard look-up mechanisms. In this case, the server must bridge the gap with a pseudo file system.

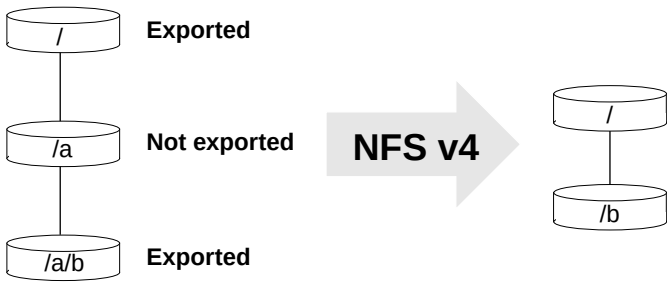

Figure 2 Portions of the server namespace that are not exported are bridged via a pseudo file system that views only the exported directories.

File Locking

NFS Versions 2 and 3 do not offer any file locking functions, but instead rely on an ancillary protocol, Network Lock Manager (NLM) to provide exclusive access to file data as well as lock recovery in the case of client or server failure.

Unfortunately, NLM is not widely supported. In addition to requiring the additional implementation of a separate protocol, NLM has minor design flaws relating to high network load and recovery, and is not well suited for Internet firewall traversals.

NFS Version 4 provides UNIX file locking functions and Windows share locking functions. A share reservation is the equivalent of file locking, providing the basis for sharing or exclusive access to file data without the risk of data corruption. Additionally, NFS Version 4 offers record or byte-range locking functions.

To correctly support share locks, NFS Version 4 supplants the existing mechanisms used to create or open a file (LOOKUP, CREATE, ACCESS) with mechanisms that can atomically open or create files. A new function, OPEN, completely subsumes the functionality of LOOKUP, CREATE, and ACCESS. The OPEN procedure obtains the initial file handle and indicates the desired access (read, write, both) and what, if any, access to deny to others (deny none, deny read, deny write, deny both). These access flags enable the implementation of policies ranging from advisory locking only to full mandatory locking.

However, because many operations require a file handle, the traditional LOOKUP will be preserved to map a file name to a file handle without establishing a lock state on the file. While ACCESS is just a subset of OPEN, the ACCESS will be preserved as a lighter weight mechanism to determine the access rights that a user has with respect to a file system object.

Designed for Protocol Extensions

A great strength of NFS is the simplicity of its protocol: straightforward, interoperable implementations can be accomplished with relative ease. This makes it easy to add or layer functionality when these changes do not require protocol revisions. When the NFS protocol is deficient or a minor modification to the protocol is the best solution, versioning or managed extensions would be helpful.

Previous versions of NFS did not accept extensions well. Small, functional additions would have greatly increased the overall value of the protocol. However, the perceived size and burden of revising the RPC version to introduce new functionality discouraged change.

The RPC protocol uses a version number to describe the set of procedure calls, replies, and their semantics and attributes. Any change in this set must be reflected in a new version number for the program. For instance, the addition of a new procedure requires a protocol revision, even though the RPC protocol already allowed procedures not to be implemented and provided an error mechanism.

NFS Version 4 allows for the infrequent instances where a protocol extension within the RPC version number is the most prudent course and revision would be unnecessary or impractical.

Summary

With the assistance of Sun's continued commitment to open standards and ongoing work with the IETF, the protocol for NFS Version 4 is openly available, interoperable, and provides excellent service in Internet and intranet environments. Some key NFS features are highlighted in Table 4.

Feature	Benefit	NFS V2	NFS V3	NFS V4
Local disk caching	Enhances client performance by increasing cache space available	✔	✔	✔
Automatic mounting	Makes global file systems continuously and transparently accessible to users	✔	✔	✔
Centralized administration	Reduces time and effort for routine administration tasks	✔	✔	✔
Global file system namespace	Creates file pathnames that are valid throughout the network so users and applications can move freely	✔	✔	✔
Client determination of protocol version supported by the server	Clients and servers negotiate which protocol to use based on what they both support. Backwards compatible with installed base.	✔	✔	✔
Reduced attribute requests	Increases scalability and performance		✔	✔
Reduced requests for lookup information	Increases scalability and performance		✔	✔
Asynchronous writes	Improves client write throughput		✔	✔
COMPOUND procedure	Better performance on high-latency networks			✔
Use of a well-known port, 2049, for NFS services	Easier transmissions through packet-filtering firewalls			✔

Table 4 Comparison of NFS Versions 2, 3, and 4 features and benefits

Feature	Benefit	NFS V2	NFS V3	NFS V4
UTF-8 universal character set	Supports internationalization with multiple languages in file and pathnames			✔
Built-in RPCSEC_GSS protocol	Strengthens security, supports multiple security mechanisms			✔
Volatile file handles	Accommodates server environments where persistent file handles are not implemented			✔
Flexible attributes	Provides extensibility for new attributes and indicators to indicate non-support			✔
Server namespaces	Client paths span exported file systems and cross-drive letter root names			✔
File locking	Supports share locks			✔

Table 4 Comparison of NFS Versions 2, 3, and 4 features and benefits

As the table clearly shows, NFS Version 4 meets the demands for fast, reliable, secure, and transparent file service over the Internet. NFS Version 4 implements key features to improve Internet access and performance, adds strong security, enhances cross-platform interoperability, and provides an extensible protocol that will carry NFS through the demands of the future. Current users of NFS Versions 2 and 3 will find that Version 4 is an important link in their migration strategy to an Internet- and intranet-centric world.

References A

General

Schneier, Bruce, Applied Cryptography, John Wiley and Sons, 1994

Stern, Hal, Managing NFS and NIS, O'Reilly, June 1991

NFS Protocol

Callaghan, B., Pawlowski, B. and P. Staubach, *"NFS Version 3 Protocol Specification,"* RFC 1813, June 1995, http://www.ietf.org/rfc/rfc1813.txt

Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D. and Hitz, D., *"NFS Version 3 Design and Implementation,"* Proceedings of the USENIX Summer 1994 Technical Conference

Sun Microsystems, Inc., *"The NFS Distributed File Service,"* NFS White Paper, March 1995, http://www.sun.com/software/white-papers/wp-nfs/

Sun Microsystems, Inc., *"NFS: Network File System Protocol Specification,"* RFC 1094, March 1989, http://www.ietf.org/rfc/rfc1094.txt

Sun Microsystems, Inc., *"NFS Version 2 and Version 3 Security Issues and the NFS Protocol's Use of RPCSEC_GSS and Kerberos V5,"* RFC 2623, June 1999, http://www.ietf.org/rfc/rfc2623.txt

Sun Microsystems, Inc., *"NFS Version 3 Security Protocol Specification,"* June 1995, http://www.ietf.org/rfc/rfc1813.txt

Sun Microsystems, Inc., *"NFS Version 4 Design Considerations,"* June 1999, http://www.ietf.org/rfc/rfc2624.txt

Sun Microsystems, Inc., *"WebNFS™ Client Specification,"* RFC 2054, October 1996. http://www.ietf.org/rfc/rfc2054.txt

Sun Microsystems, Inc., *"WebNFS Server Specification,"* RFC 2054, October 1996, http://www.ietf.org/rfc/rfc2054.txt

Sun Microsystems, Inc., *"XDR: External Data Representation Standard,"* RFC 1832, August 1995. http://www.ietf.org/rfc/rfc1832.txt

RPC Protocol

Srinivasan, R., *"RPC: Remote Procedure Call Protocol Specification Version 2,"* RFC 1831, August 1995, http://www.ietf.org/rfc/rfc1831.txt

Srinivasan, R. (1995). *"Binding Protocols for ONC RPC Version 2,"* RFC 1833, August 1995, http://www.ietf.org/rfc/rfc1833.txt

RPCSEC_GSS

Eisler, M., Chiu, A. and L. Ling, *"RPCSEC_GSS Protocol Specification,"* RFC 2203, September 1997, http://www.ietf.org/rfc/rfc2203.txt

Sun Microsystems, Inc., *"NFS Version 2 and Version 3 Security Issues and the NFS Protocol's Use of RPCSEC_GSS and Kerberos V5,"* RFC 2623, June 1999, http://www.ietf.org/rfc/rfc2623.txt

GSS-API

Jaspan, B. (1995). *"GSS-API Security for ONC RPC,"* 1995 Proceedings of The Internet Society Symposium on Network and Distributed System Security, pp. 144-151

Linn, J., *"Generic Security Service Application Program Interface,"* RFC 1508, September 1993, http://www.ietf.org/rfc/rfc1508.txt

Linn, J., *"Generic Security Service Application Program Interface, Version 2,"* RFC 2078, January 1997, http://www.ietf.org/rfc/rfc2078.txt

KERBEROS

Callaghan, B., Singh, S., *"The Autofs Automounter,"* Proceedings of the 1993 Summer USENIX Technical Conference, 1993

Linn, J., *"The Kerberos Version 5 GSS-API Mechanism,"* RFC 1964, June 1996, http://www.ietf.org/rfc/rfc1964.txt

Massachusetts Institute of Technology, *"Kerberos: The Network Authentication Protocol,"* MIT's implementation of Kerberos V5 including implementations of RFC 1510 and RFC 1964, 1998, http://web.mit.edu/kerberos/www/index.html

UTF-8

The Unicode Consortium, http://www.unicode.org

Glossary B

Access Control Lists (ACLs)

List that indicates what type of access is allowed for a file. For example, it indicates whether certain individuals or groups of individuals are allowed to perform file operations such as read, write, execute, etc.

Authentication Service

Security mechanism that determines if someone is who they claim to be before being allowed to use resources. Typically involves checking for information known only to that individual such as a password.

Authorization Service

Security mechanism that determines if someone has permission to do what they want to do, once they have been properly authenticated. Permission bits and ACLs are examples of authorization.

Automounter

Component of client-side NFS that is responsible for transparently mounting and unmounting server file systems on behalf of client requests.

Cache

Fast access, temporary storage area (physical memory and disk) utilized by an NFS client to increase the speed of access to file data. When file data is cached on the local system, client performance is improved because clients spend less time querying the server or waiting for the server to transfer data over the network.

Data Encryption Standard (DES) Authentication

Private key encryption algorithm originally developed by IBM which became a national standard in the 1970s.

Diffie-Hellman Authentication

Public key encryption algorithm developed by Whitfield Diffie and Martin Hellman in 1976 that is utilized by NFS on the Solaris™ Operating Environment.

Encryption

Mechanism for converting information into a non-readable form so it cannot be interpreted by an unauthorized recipient. This is accomplished by applying an encryption algorithm with a special value (known as a key) to the data.

Generic Security Service Application Program Interface (GSS-API)

Standard framework for uniting multiple authentication flavors under a single API.

Internet Assigned Numbers Authority (IANA)

Central coordinator for the assignment of unique parameter values for Internet protocols. Chartered by the Internet Society (ISOC) to act as a clearinghouse to assign and coordinate the use of numerous Internet protocol parameters.

Internet Engineering Task Force (IETF)

Large, open international community of network designers, operators, vendors, and researchers concerned with evolution of Internet architecture and smooth operation of the Internet.

Kerberos

Authentication service developed by MIT for project Athena that utilizes private and public key mechanisms to exchange authentication information contained in expanded packages (tickets).

Local Area Network (LAN)

Closely linked group of computer systems that communicate with one another via connecting hardware and software.

Mount

Accessing a directory from a disk attached to the machine making a mount request or to a remote disk on a network.

Mount Point

Point in the client file system tree at which a server file system will be mounted. Also called a mount or client name (pathname).

Namespace

Collective names or pathnames used by clients to access server file systems. (See mount point.)

ONC

Distributed applications architecture promoted and controlled by a consortium led by Sun Microsystems, Inc.

Portmapper

Network system service used by all other remote-procedure-call-based services. Keeps track of correspondence between ports (logical communications channels) and services on a machine, providing a standard way for clients to look up the port number of any remote procedure call program supported by the server.

Public Key Encryption Service

Encryption service in which the key or value used to encrypt data is different from that used to decrypt information.

Private Key Encryption Service

Encryption service in which the key or value used to encrypt data is the same as that used to decrypt information.

Protocol

Formal description of messages to be exchanged and rules to be followed for two or more systems to exchange information.

Root

In a hierarchy of items, the item from which all other items are descended. A root item has nothing above it in the hierarchy. See root disk, root file system, and root directory.

Root directory

Base directory from which all other directories stem, directly or indirectly.

Root disk

On UNIX servers, the disk drive where the operating system resides.

Remote Procedure Call (RPC)

Paradigm for implementing client-server, distributed computing. A request is sent to a remote system to execute a designated procedure, using arguments supplied, and the result is returned to the caller. Many different RPC protocols exist.

Root file system

File system residing on the root device (a device predefined by the system at initialization) that anchors the overall file system.

RPCSEC_GSS

ONC RPC security flavor that allows RPC protocols to access GSS-API.

Symbolic Link

File or directory that points to another file or directory, giving both files or directories the same contents.

Transmission Control Protocol (TCP)

Reliable transport layer service, developed by the Department of Defense, utilized by NFS for communication over LANs and WANs.

UDP

Unreliable transport layer service, developed by the Department of Defense, utilized by many existing NFS implementations to communicate over LANs.

UTF-8

Universal Character Set (UCS), eight-bit octet transformation format.

WAN (Wide Area Network)

Network consisting of many systems that provide file transfer services, it may cover a large physical area.

Sun Microsystems Incorporated
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300
FAX 650 969-9131
http://www.sun.com

Printed in USA