



We meanNetwork File System

NFS (Network File System)

- ❖ The most commercially successful and widely available remote file system protocol
- ❖ Designed and implemented by Sun Microsystems (Walsh et al, 1985; Sandberg et al, 1985)
- ❖ Reasons for success
 - ❖ The NFS protocol is public domain
 - ❖ Sun sells that implementation to all people for less than the cost of implementing it themselves
- ❖ Evolved from version 2 to version 3 (which is the common implementation today).

Introduction: Remote File-systems

- ❖ When networking became widely available users wanting to share files had to log in across the net to a **central machine**
- ❖ This **central machine** quickly become far more loaded than the user's local machine
- ❖ So there was a demand for a convenient way to share files on several machines
- ❖ The most easily understood **sharing model** is one that allow a server machine to **export** its file systems to one or more clients. The clients can then **import** these file systems and present them as local file systems

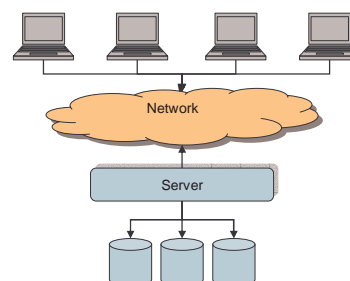
NFS Overview

- ❖ Views a set of interconnected workstations as a set of independent machines with independent file systems
- ❖ The goal is to allow some degree of sharing among these file systems (on explicit request)
- ❖ Sharing is based on client server relationships
- ❖ A machine may be both client and server
- ❖ The protocol is stateless
- ❖ Designed to support UNIX file system semantics
- ❖ The protocol design is transport independent

First Attempts

- ❖ UNIX United
 - ❖ Implemented near the top of the kernel
 - ❖ No caching → slow performance
 - ❖ Nearly identical semantics to a local system
- ❖ Sun Microsystems network disk
 - ❖ Implemented near the bottom of the kernel
 - ❖ Excellent performance
 - ❖ Bad UNIX semantic
- ❖ System V RFS
 - ❖ Excellent UNIX semantic
 - ❖ Slow performance
- ❖ AFS

The division of NFS between client and server

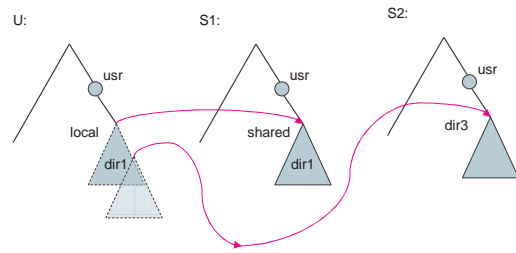


Mounting

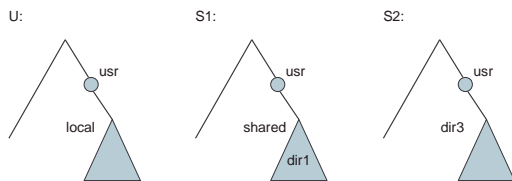
- ❖ A machine (M1) wants to access transparently a remote directory (On another machine M2)
- ❖ To do that a client on M1 should perform a mount operation
- ❖ The semantics are that a remote directory is mounted over a directory of a local file system
- ❖ Once the mount operation is complete, the mounted directory looks like an integral sub tree of the local file system
- ❖ The previous sub tree accessed from the local directory is not accessible anymore

Example 3: cascading mounts

The effects of mounting **S2:/usr/dir3** over **U:/usr/local/dir1**



Example 1: Initial situation

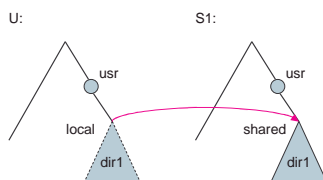


RPC/XDR

- ❖ One of the design goals of NFS is to operate in heterogeneous environments
- ❖ The NFS specification is independent from the communication media
- ❖ This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol

Example 2: Simple mount

The effects of mounting **S1:/usr/shared** over **U:/usr/local**



RPC

- ❖ A server registers a procedure implementation
- ❖ A client calls a function which looks local to the client
 - ❖ E.g., `add(a,b)`
- ❖ The RPC implementation packs (marshals) the function name and parameter values into a message and sends it to the server

RPC

- ❖ Server accepts the message, unpacks (un-marshals) parameters and calls the local function
- ❖ Return values is then marshaled into a message and sent back to the client
- ❖ Marshaling/un-marshaling must take into account differences in data representation

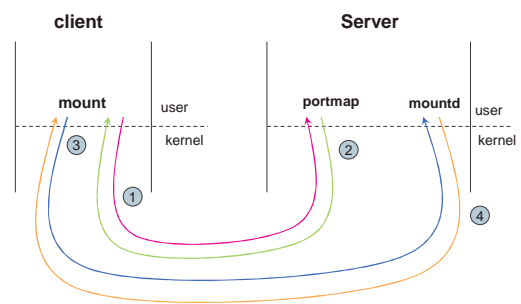
The mount protocol

1. Client's mount process send message to the server's portmap daemon requesting port number of the server's mountd daemon
2. Server's portmap daemon returns the requested info
3. Client's mountd send the server's mountd a request with the path of the file system it wants to mount
4. Server's mountd request a file handle from the kernel
 1. If the request is successful the handle is returned to the client
 2. If not error is returned
5. The client's mountd perform the mount() system call using the received file handle

RPC

- ❖ Transport: both TCP and UDP
- ❖ Data types: atomic types and non-recursive structures
- ❖ Pointers are not supported
- ❖ Complex memory objects (e.g., linked lists) are not supported
- ❖ NFS is built on top of RPC

The mount protocol

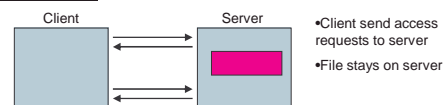


The mount protocol

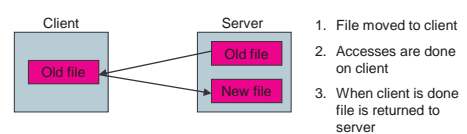
- ❖ Is used to establish the initial logical connection between a server and a client
- ❖ Each machine has a server process (daemon) (outside the kernel) performing the protocol functions
- ❖ The server has a list of exported directories (/etc/exports)
- ❖ The portmap service is used to find the location (port number) of the server mount service

Access Models

Remote access mode:



Upload/download mode:



NFS Protocol Requests

- ❖ Provides a set of RPCs for remote file operations

RPC request	Action	Idempotent
GETATTR	Get file attributes	Yes
SETATTR	Set file attributes	Yes
LOOKUP	Look up file name	Yes
READLINK	Read from symbolic name	Yes
READ	Read from file	Yes
WRITE	Write to file	Yes
CREATE	Create file	Yes
REMOVE	Remove file	No
RENAME	Rename file	No
LINK	Create link to file	No
SYMLINK	Create symbolic link	Yes
MKDIR	Create directory	No
RMDIR	Remove directory	No
REaddir	Read from directory	Yes
STATFS	Get file system attributes	Yes

File handle

- ❖ Built from the file system identifier , an inode number and a generation number
- ❖ The server creates a unique identifier for each local fs
- ❖ A generation number is assigned to an inode each time the latter is allocated to represent a new file
- ❖ MOST NFS implementations use a random number generator to allocate generation numbers
- ❖ The generation number verifies that the inode still references that same file that it referenced when the file was first accessed
- ❖ 32 bits in NFS-V2, 64 bits in NFS-V3, 128 bits in NFS-V4

Idempotent Operations

- ❖ We can see that most operations are *idempotent*
- ❖ An idempotent operation is one that can be repeated several times without the final result being changed or an error being caused
- ❖ For example *writing the same data to the same offset in the file*
- ❖ However *removing the file is not* idempotent
- ❖ This is an issue when RPC acknowledgments are lost and the client retransmit the request
- ❖ *If a non idempotent request is retransmitted and is not in the server recent request cache we get an error*

Statelessness

- ❖ *If you noticed*, open and close are missing from the "requests table"
- ❖ NFS servers are *stateless*, which means that they don't maintain information about their clients from one access to another
- ❖ Since there is no parallel to open files table, every request has to provide a full set of arguments, including a unique file identifier and an absolute offset (self contained)
- ❖ The resulting design is robust, no special measures need to be taken to recover the server after a crash

The NFS File handle

- ❖ Each file on the server can be identified by a unique file handle
- ❖ Are globally unique and are passed in operations
- ❖ Created by the server when pathname translation request (lookup) is received
- ❖ The server find the requested file or directory and ensure that the user has access permissions
- ❖ If all is O.K the server returns the file handle
- ❖ It identify the file in future requests

Drawback to statelessness

- ❖ Semantics of the local file system imply state
 - ❖ When a file is unlinked, it continue to be accessible until the last reference to it is closed
 - ❖ Advisory locking
- ❖ NFS doesn't know about clients so it cannot properly know when to free file space (.nfsAxxxx4.4)
- ❖ Performance: All operations that modify the file-system must be committed to stable storage before the RPC can be acknowledged
- ❖ In NFS-V3 new asynchronous write RPC request eliminates some of the synchronous writs

The NFS Architecture

- ❖ Operate in a remote access model (as opposed to upload/download)
- ❖ Consists of tree major layers
 - ❖ unix file system interface (read, write, open, close)
 - ❖ virtual file system (VFS) layer
 - ❖ NFS protocol implementations
- ❖ The VFS is based on a file representation structure called vnode

Path name translation

- ❖ Done by breaking the path into component names and performing a separate NFS lookup for every pair of component name and directory vnode
- ❖ Once a mount point is crossed every lookup causes a separate RPC request to the server
- ❖ This is since at any point there can another mount point for the client which the server is not aware of
- ❖ The client maintain a directory name lookup cache holds the vnodes for remote directories names

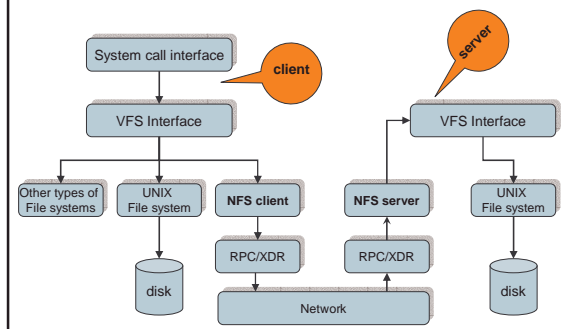
NFS Architecture

- ❖ Vnodes contain a numeric designator for files that is network-wide unique
- ❖ The VFS distinguish local files from remote files (and also deferent kind of local files (deferent local file-systems))
- ❖ VFS activate file-system specific operations to handle local requests according to the fs type and call NFS protocol procedures for remote requests

Path translation (cont)

- ❖ Entries are pruned from the cache when attributes change
- ❖ A server cannot act as an intermediary between the client and another server
- ❖ The client must establish direct client server connection
- ❖ When a client perform lookup on a directory on which the server has mounted a file system, the clients sees the underling directory instead of the mounted directory

Schematic View of the NFS Architecture



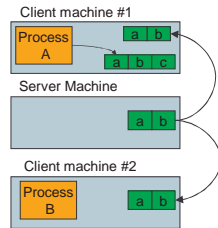
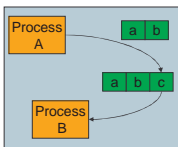
Semantics of file sharing

- ❖ When two or more users share the same file it is necessary to define the semantics of read/write exactly to avoid problems
- ❖ In **single-processor** systems that allow file sharing, the semantics normally state that when a **read** operation follows a **write** the value read is the value stored in the last write
- ❖ the system enforce **absolute time ordering** on all operations
- ❖ We will refer to this model as **UNIX semantics**

Semantics of file sharing

- ❖ In a distributed system, UNIX semantics can be achieved as long as there is not client caching
- ❖ This might usually result in poor performance
- ❖ The performance problem is solved by allowing clients to cache files

Single machine:



NFS Caching

- ❖ There are 2 caches
 - ❖ File attributes
 - ❖ File blocks
- ❖ On a file open the kernel checks with the remote server whether to fetch or revalidate the cached attributes
- ❖ The cached blocks are used only if the cached attributes are up-to-date
- ❖ Both read-ahead and delayed write techniques are used between the server and the client
- ❖ There are many more optimization methods

Semantics of file sharing

- ❖ To solve the obsolete values we can propagate all changes to cached file immediately
- ❖ Another option is to relax the semantics of file sharing

Changes to an open file are initially visible only to the process (or machine) that modified the file
Only when the file is closed are the changes made visible to other processes or machines

We call this **session semantics**

- ❖ In session semantics the previous behavior is correct

Summary

- ❖ NFS is widely used and is implemented in almost any environment
- ❖ Performance are good due to caching and other optimizations methods
- ❖ File sharing semantics is not UNIX semantics
- ❖ NFS-V4 is out there and is not stateless and offer many improvements in many fields

File sharing semantics

- ❖ NFS implement something in the middle of **UNIX semantics** and **session semantics**
- ❖ There are also problems with session semantics
 - ❖ What if 2 clients are simultaneously caching and modifying the same file.
- ❖ There are other sharing semantics
 - ❖ Immutable files
 - ❖ Transactions
- ❖ But enough of this for now ☺