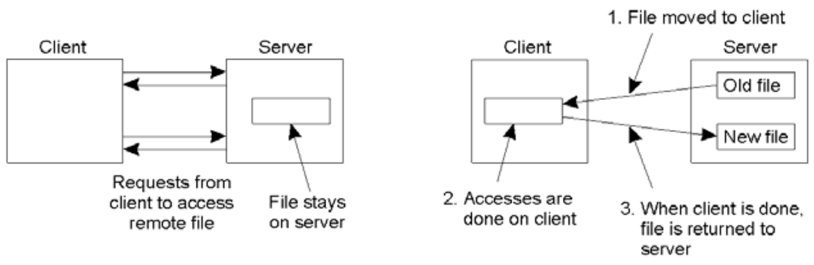
COP 6611 Advanced Operating System

Distributed File Systems

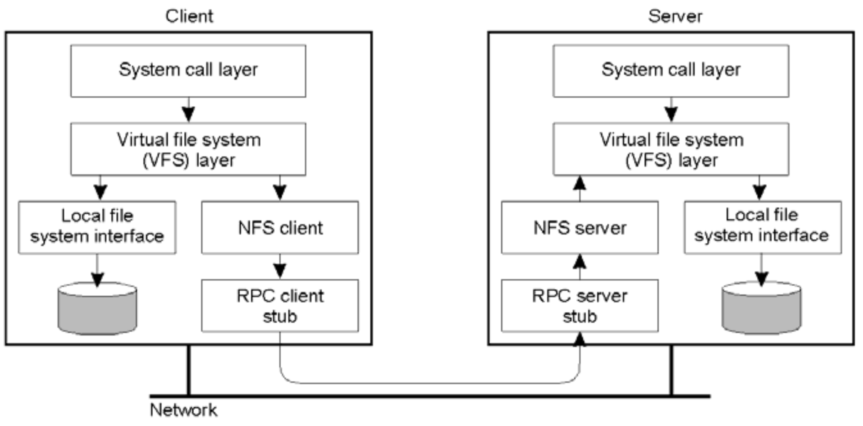
Chi Zhang
czhang@cs.fiu.edu

NFS Architecture (1)


Client
Server
Requests from client to access remote file
File stays on server
1. File moved to client
Client
Server
Old file
New file
2. Accesses are done on client
3. When client is done, file is returned to server

a) The remote access model. (like NFS)
b) The upload/download model (like FTP)

NFS Architecture (2)


Client
Server
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

The basic NFS architecture for UNIX systems:

Hide differences between various file systems (e.g., local vs. remote)

3

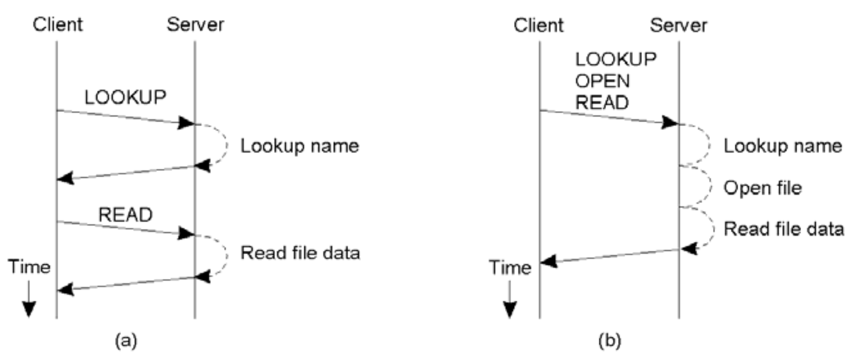
File System Model

Operation	v3	v4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Read the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

An incomplete list of file system operations supported by NFS.

4

Communication: RPC



a) Reading data from a file in NFS version 3.
b) Reading data using a compound procedure in version 4. (If *lookup* fails, the succeeding *open* is not even attempted)

5

Stateful vs. Stateless Servers

- NFS v3 is stateless
- NSF v4 is stateful
  - Locking
  - Authentication
  - Strict cache consistency
  - Call back functions

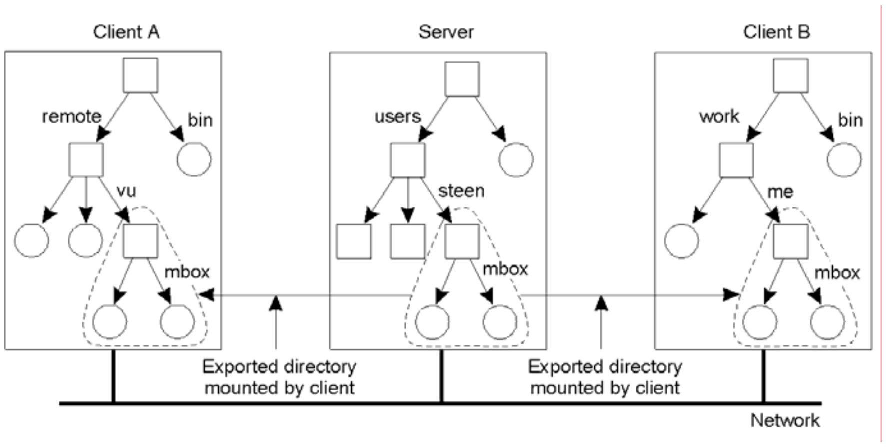
6

File Handles

- On a local file system, a file descriptor maps to an i-node number.
- In NFS, a file handle usually consists of dev number, i-node number and i-node generation number (for i-node reuse, because of client caching)
- 64 bytes in v3 and 128 bytes in v4, only makes sense to the server.
- Clients *lookup* the file handle for a given file name under a directory (given by its file handle), and cache the handle locally.
- How to get the initial file handle? *putrootfh*

7

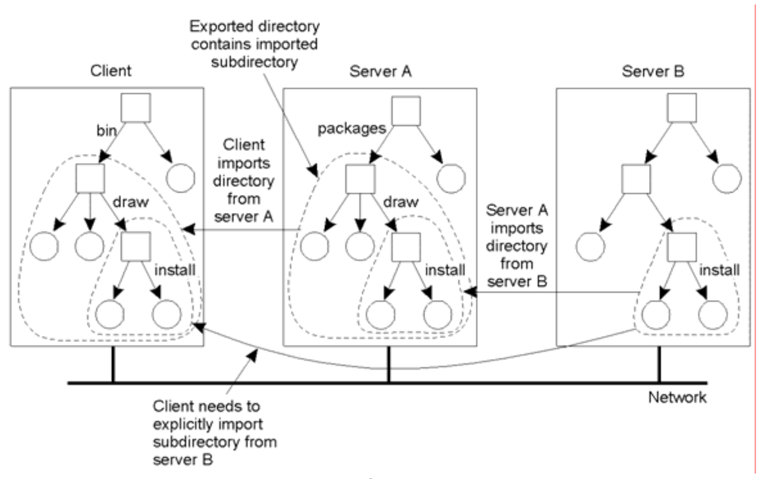
Naming (1)


Client A
Server
Client B
remote
bin
users
work
bin
vu
steen
me
mbox
mbox
mbox
Exported directory mounted by client
Exported directory mounted by client
Network

Mounting (part of) a remote file system in NFS.
Clients A and B have different paths names for the same file on the server, unless the name space on clients is partly standardized

8

Naming (2)


Exported directory contains imported subdirectory
Client
Server A
Server B
bin
packages
Client imports directory from server A
draw
draw
Server A imports directory from server B
install
install
install
Network
Client needs to explicitly import subdirectory from server B

Mounting nested directories from multiple servers in NFS.

The client needs to explicitly mount the nested directory, since there is no special file handle that includes the server ID.

File Locking in NFS

Operation	Description
Lock	Creates a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the leas on a specified lock

NFS version 4 operations related to file locking.

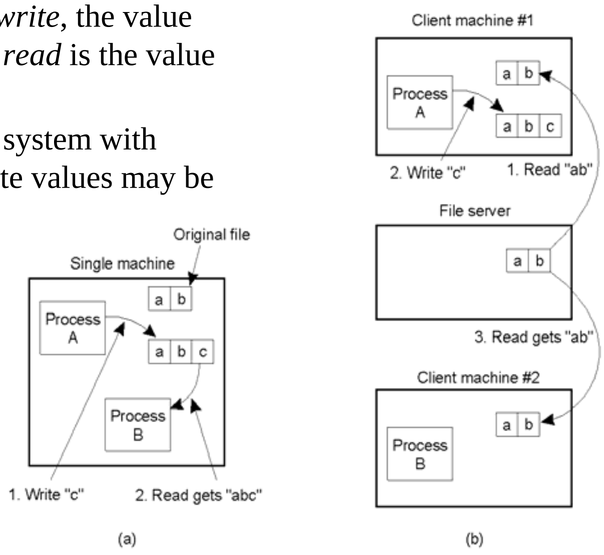
Lock operations can be non-blocking (clients have to poll) or blocking.

Locks are granted for a specific time (in case a client crashes). The client needs to renew the lease.

Semantics of File Sharing (1)

a) On a single processor, when a *read* follows a *write*, the value returned by the *read* is the value just written.

b) In a distributed system with caching, obsolete values may be returned.



Semantics of File Sharing (2)

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics (NFS)	No changes are visible to other processes until the file is closed. Invalidate the local cache when the file is re-opened later.

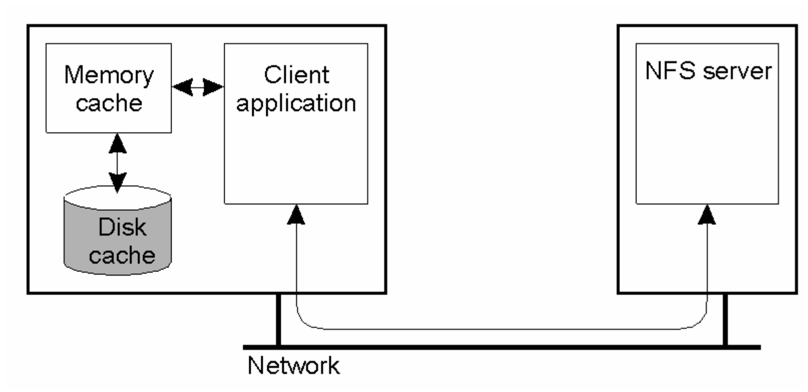
Four ways of dealing with the shared files in a distributed system.

Session semantics: propagating updates on cache immediately back to the server is inefficient. Just relax the semantics of file sharing in NFS.

If two clients simultaneously cache and modify the file, the final result depends which one closes more recently.

12

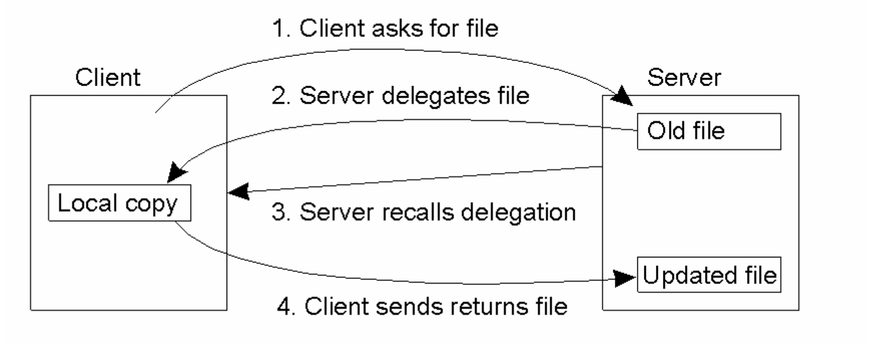
Client Caching (1)



Client-side caching in NFS.
Read / Write to the local cache.

13

Client Caching (2)



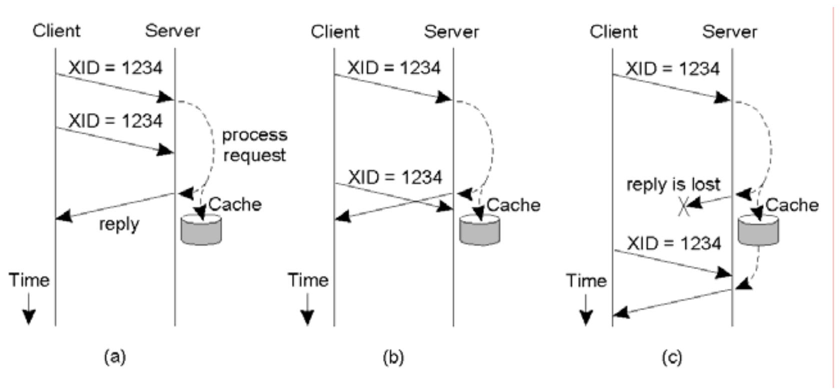
Using the NFS v4 callback mechanism to recall file delegation.

The client machine can locally candle *open/close* operations from other clients on the *same* machine.

The (stateful) server can recall the delegation, for example, when another client on a different machine needs to access the file.

14

RPC Failures



Each RPC request from a client carries a unique transaction ID (XID). Three situations for handling retransmissions.

a) The request is still in progress
b) The reply has just been returned
c) The reply has been some time ago, but was lost.

File locking and Delegation in the Presence of Failures

- A lease on every lock to solve client crashes.
- After a server recovers from a crash, it enters a *grace period*, during which new locks are not granted, clients can reclaim locks granted before the crash.
  - The server rebuilds lock information.
- If a client reclaims a file delegation after a server recovers from a crash, the server forces the client to flush all modifications back to the server, effectively recalling the delegation.