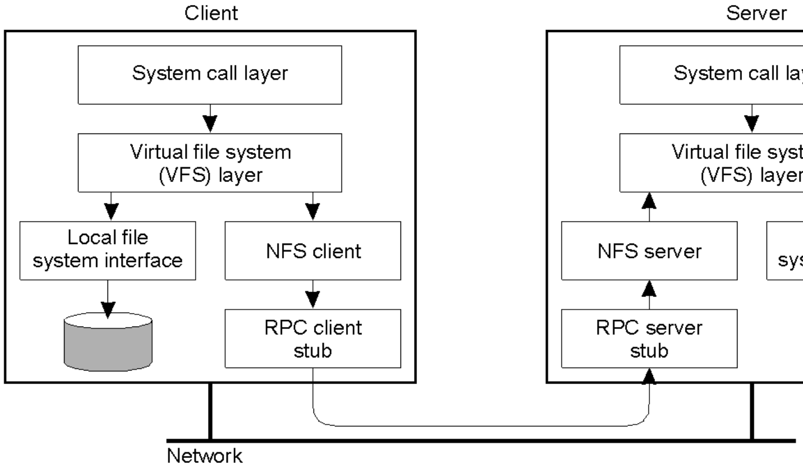
Today: Distributed File Systems

- Issues in distributed file systems
- Sun's Network File System case study

NFS Architecture

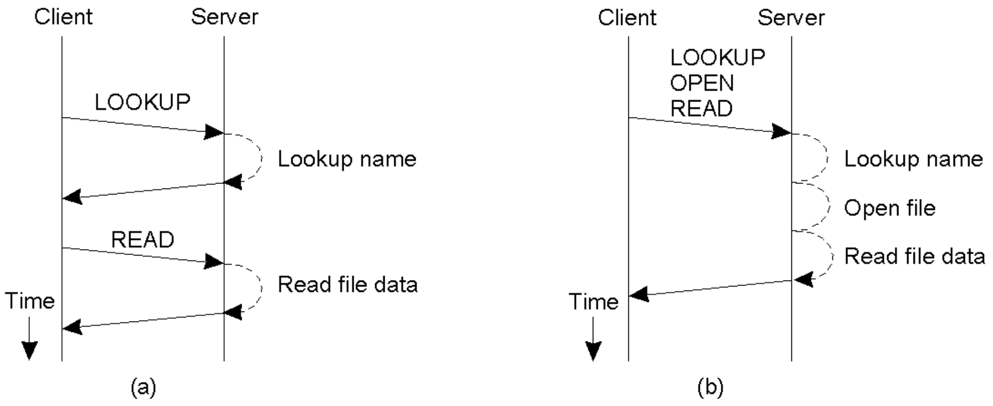
- Sun's Network File System (NFS) – widely used distributed file system
- Uses the virtual file system layer to handle local and remote files


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

NFS Operations

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

Communication



a) Reading data from a file in NFS version 3.

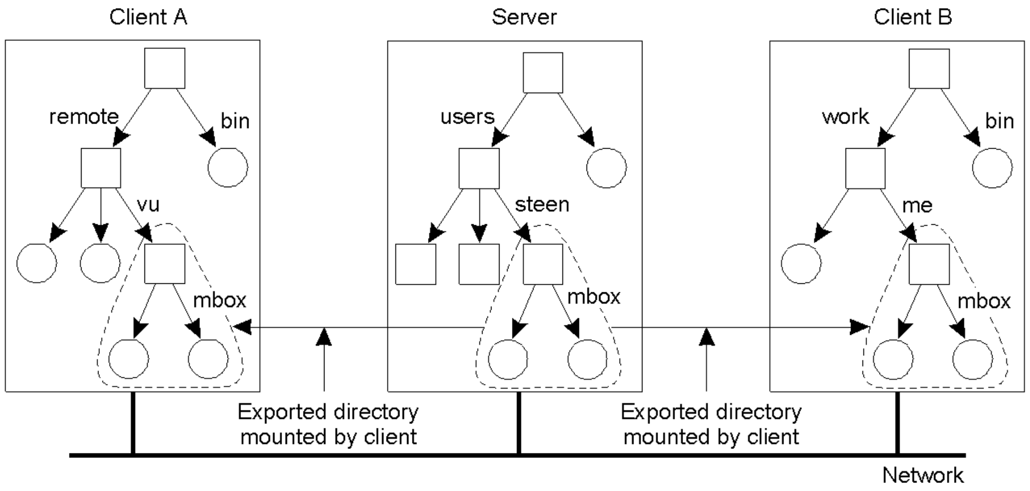
b) Reading data using a compound procedure in version 4.

Both versions use Open Network Computing (ONC) RPCs

- One RPC per operation (NFS v3); multiple operations supported in v4.

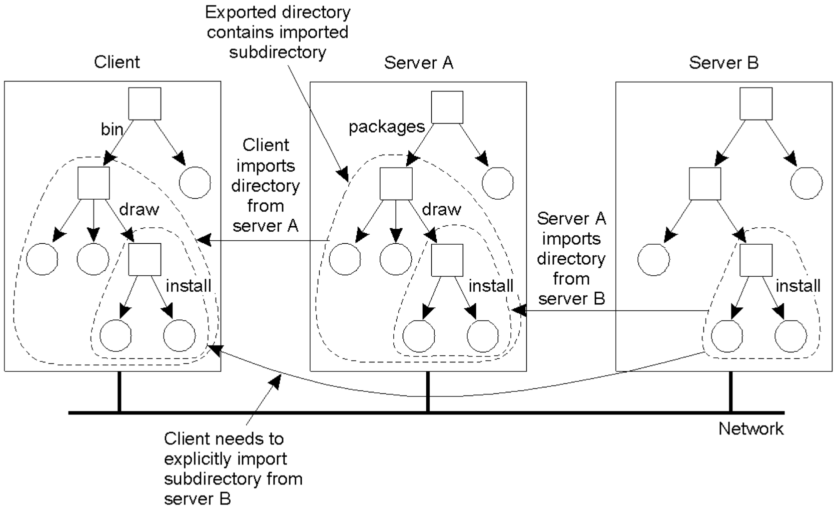
Naming: Mount Protocol

- NFS uses the mount protocol to access remote files
  - Mount protocol establishes a local name for remote files
  - Users access remote files using local names; OS takes care of the mapping

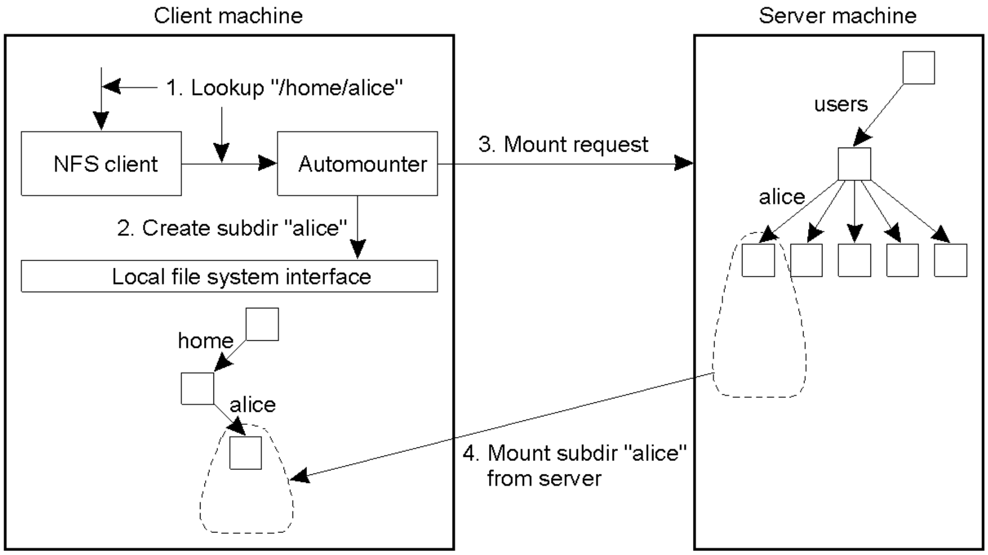

Client A
Server
Client B
remote
bin
vu
mbox
users
steen
mbox
work
bin
me
mbox
Exported directory mounted by client
Exported directory mounted by client
Network

Naming: Crossing Mount Points

- Mounting nested directories from multiple servers
- NFS v3 does not support transitive exports (for security reasons)
  - NFS v4 allows clients to detects crossing of mount points, supports recursive lookups


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

Automounting



- Automounting: mount on demand

File Attributes (1)

Attribute	Description
TYPE	The type of the file (regular, directory, symbolic link)
SIZE	The length of the file in bytes
CHANGE	Indicator for a client to see if and/or when the file has changed
FSID	Server-unique identifier of the file's file system

- Some general mandatory file attributes in NFS.
  - NFS modeled based on Unix-like file systems
    - Implementing NFS on other file systems (Windows) difficult
  - NFS v4 enhances compatibility by using mandatory and recommended attributes

File Attributes (2)

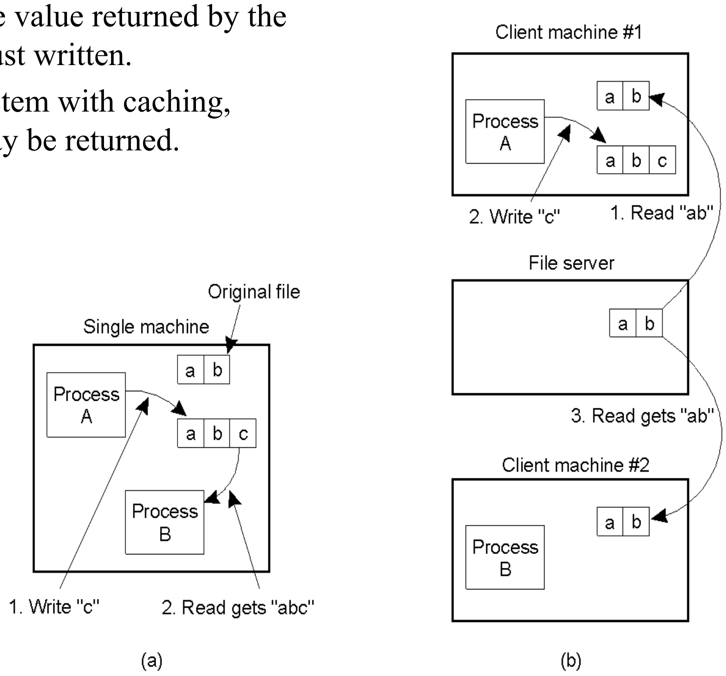
Attribute	Description
ACL	an access control list associated with the file
FILEHANDLE	The server-provided file handle of this file
FILEID	A file-system unique identifier for this file
FS_LOCATIONS	Locations in the network where this file system may be found
OWNER	The character-string name of the file's owner
TIME_ACCESS	Time when the file data were last accessed
TIME_MODIFY	Time when the file data were last modified
TIME_CREATE	Time when the file was created

- Some general recommended file attributes.

Semantics of File Sharing

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

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transaction	All changes occur atomically

- Four ways of dealing with the shared files in a distributed system.
  - NFS implements session semantics
    - Can use remote/access model for providing UNIX semantics (expensive)
    - Most implementations use local caches for performance and provide session semantics

File Locking in NFS

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

- NFS supports file locking
    - Applications can use locks to ensure consistency
    - Locking was not part of NFS until version 3
    - NFS v4 supports locking as part of the protocol (see above table)

File Locking: Share Reservations

		Current file denial state			
		NONE	READ	WRITE	BOTH
Request access	READ	Succeed	Fail	Succeed	Fail
	WRITE	Succeed	Succeed	Fail	Fail
	BOTH	Succeed	Fail	Fail	Fail

(a)

		Requested file denial state			
		NONE	READ	WRITE	BOTH
Current access state	READ	Succeed	Fail	Succeed	Fail
	WRITE	Succeed	Succeed	Fail	Fail
	BOTH	Succeed	Fail	Fail	Fail

(b)

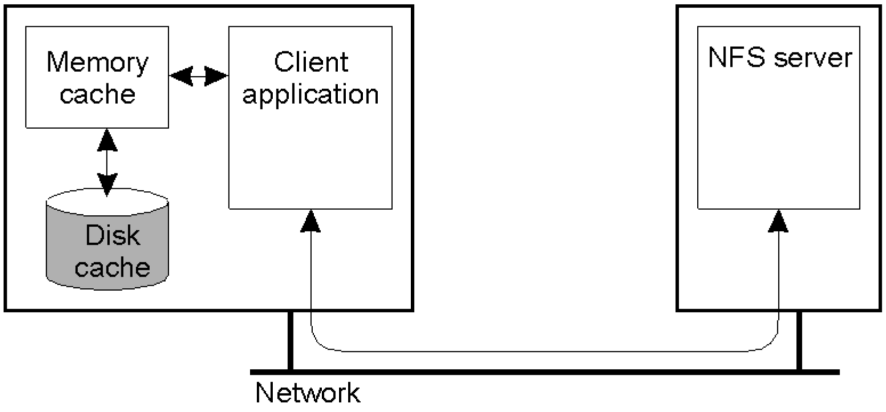
- The result of an *open* operation with share reservations in NFS.

a) When the client requests shared access given the current denial state.

b) When the client requests a denial state given the current file access state.

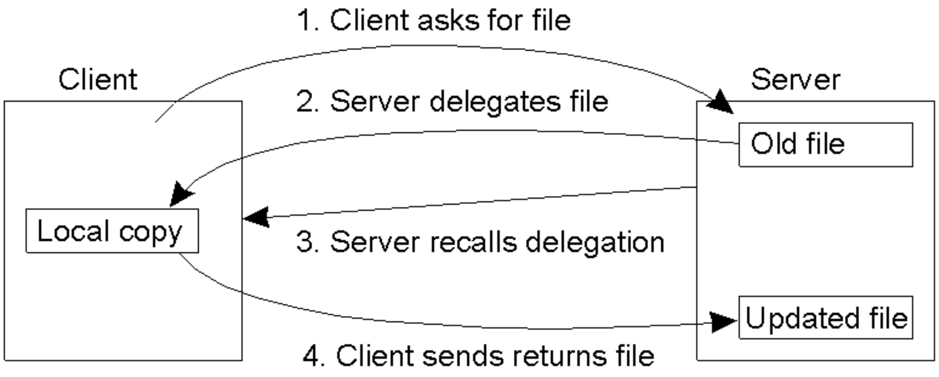
Client Caching

- Client-side caching is left to the implementation (NFS does not prohibit it)
  - Different implementation use different caching policies
    - Sun: allow cache data to be stale for up to 30 seconds

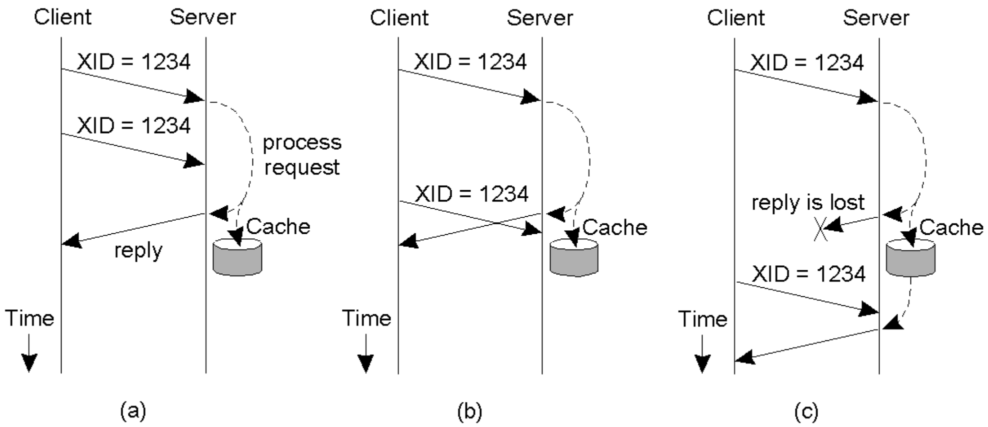


Client Caching: Delegation

- NFS V4 supports open delegation
  - Server delegates local open and close requests to the NFS client
  - Uses a callback mechanism to recall file delegation.



RPC Failures



- Three situations for handling retransmissions: use a duplicate request cache

a) The request is still in progress

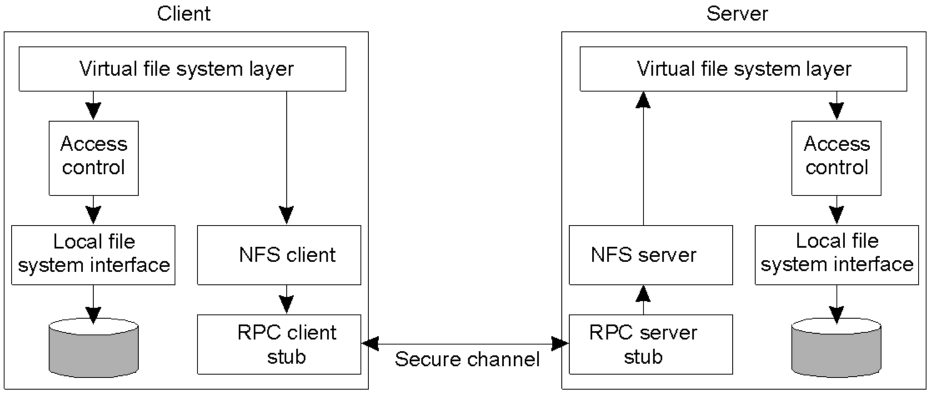
b) The reply has just been returned

c) The reply has been some time ago, but was lost.

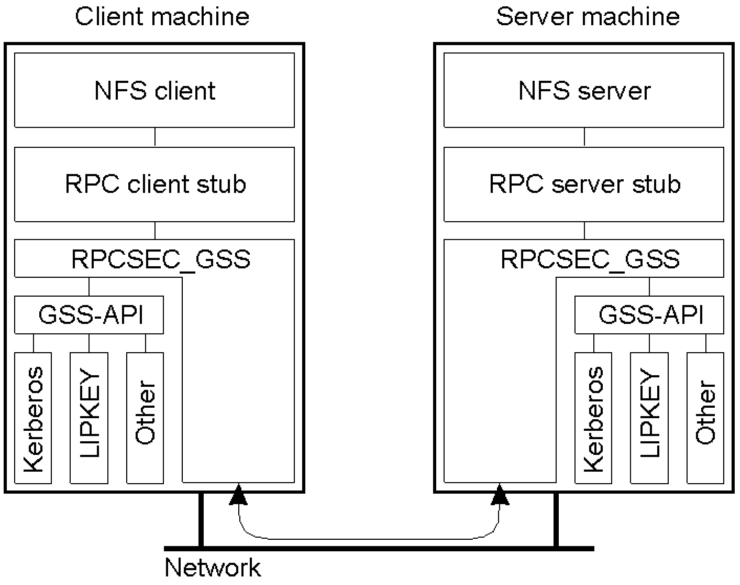
Use a duplicate-request cache: transaction Ids on RPCs, results cached

Security

- The NFS security architecture.
  - Simplest case: user ID, group ID authentication only


Client
Server
Virtual file system layer
Access control
Local file system interface
NFS client
RPC client stub
Secure channel
Virtual file system layer
Access control
NFS server
Local file system interface
RPC server stub

Secure RPCs



- Secure RPC in NFS version 4.

Replica Servers

- NFS ver 4 supports replications
- Entire file systems must be replicated
- FS_LOCATION attribute for each file
- Replicated servers: implementation specific