# Distributed file System (File system support for Mobility)

→ Two main purposes of using files:
  ↳ Permanent storage of information
  ↳ Sharing of information

→ A file system is a subsystem of an operating system that performs file management activities
  - Organization
  - Storing
  - Retrieval
  - Naming
  - Sharing
  - Protection

→ A file system provides an abstraction of a storage device; that is, it is a convenient mechanism for storing and retrieving information from the storage device (without knowing the details of space allocation and layout of the secondary storage device).

→ A Distributed file system provides similar abstraction to the users of a Distributed system and makes it convenient for them to use files in a Distributed environment.

  ↳ Design is more complex than a conventional file system.

following:

+ Remote Information Sharing
  - ↳ file to be transparently accessed by processes of any node of the system irrespective of the file's location.

+ User Mobility
  - ↳ User should not be forced to work on a specific node
    - ↳ Flexibility to work on different nodes at different times.

+ Availability
  - ↳ For better fault tolerance, files should be available for use even in the event of temporary failure of one and more nodes of the system.
    - ↳ Replication of information

+ Diskless Workstation
  - ↳ A distributed file system, with its transparent remote file-accessing capability, allows the use of diskless workstations in a system.

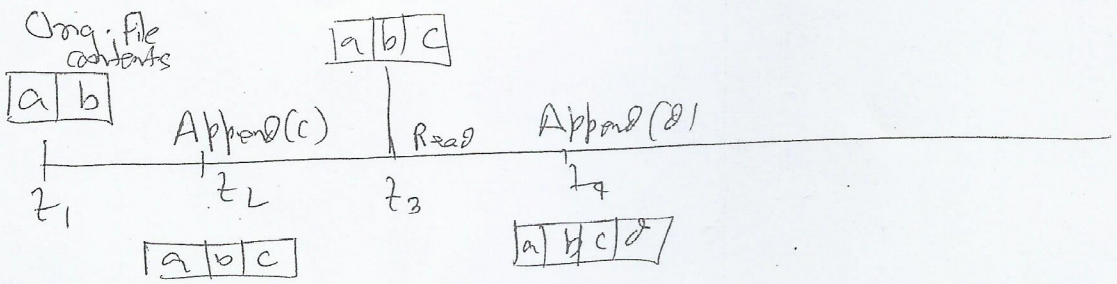
→ A DFS typically provides the following three types of services:

+ Storage Service —
  ↳ It deals with the allocation and mgmt of space on a secondary storage device that is used for storage of files in the file system.

+ True file Service
  ↳ It is concerned with the operations on individual files, such as operations for accessing and modifying the data in files and for creating and deleting files.

+ Name Service
  ↳ It provides a mapping between text names for files and references to files, that is, file IDs.
    ↳ It is also known as a Directory service.

→ File sharing Semantics
  + UNIX semantics



# Distributed File Systems

Chapter 10

# Distributed File System

a) A **distributed file system** is a file system that resides on different machines, but offers an integrated view of data stored on remote disks.

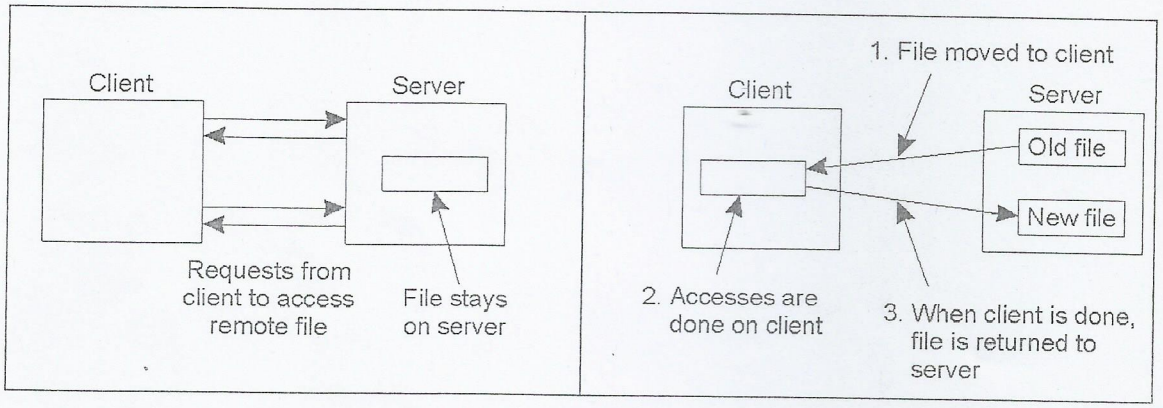
b) Examples of distributed file systems
   a) NFS
   b) AFS
   c) Coda
   d) Plan9
   e) xFS

# Network File System (NFS)

- Developed originally at Sun Microsystems for UNIX workstations.
- It is a model to integrate different file systems.
- Based on the idea that each file server provides a standardized view of its local file system.
- NFS runs on heterogeneous groups of computers.

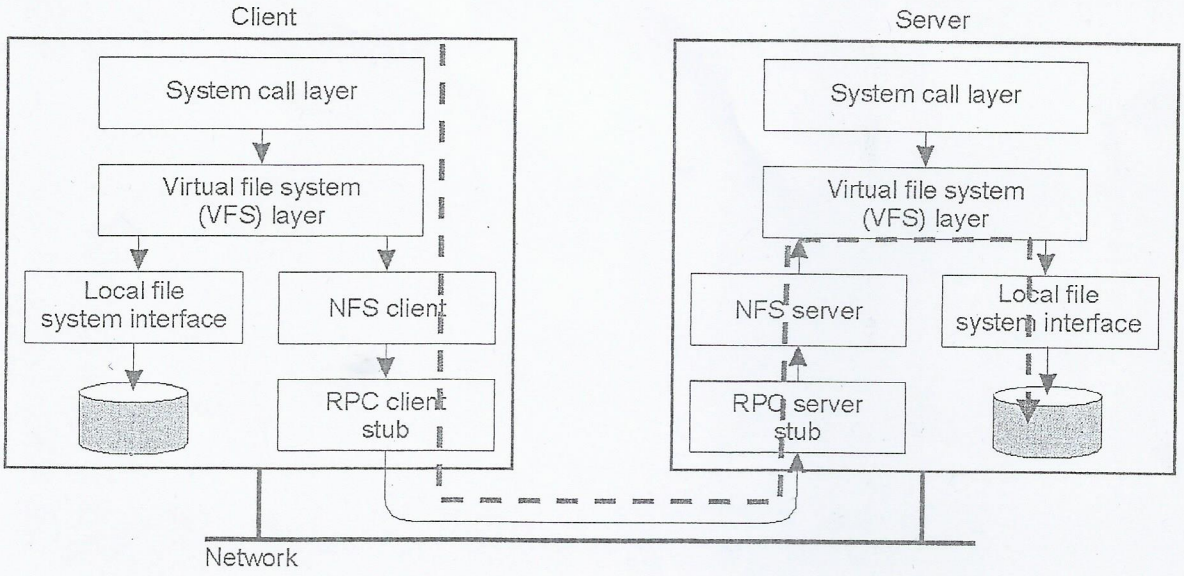
# NFS Architecture (1)

- NFS uses a **remote access model**: clients are unaware of file locations.
- Servers export a set of operations on files.




**The remote access model.** **The upload/download model.**

# NFS Architecture (2)




The basic NFS architecture for UNIX systems.

# NFS Architecture (3)

- NFS is independent from local file system organization.
- It integrates file systems used in UNIX, Linux, Windows, and other operating systems.
- The model offered is similar to UNIX-like file systems based on files as sequences of bytes.

# 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.

An important issue in the design of NFS is the independence of OS, N/w arch, and transport protocols.

# Communication (1)

a) In NFS all communications between servers and clients are implemented using Remote Procedure Call (RPC).

b) The used protocol is the Open Network Computing RPC.

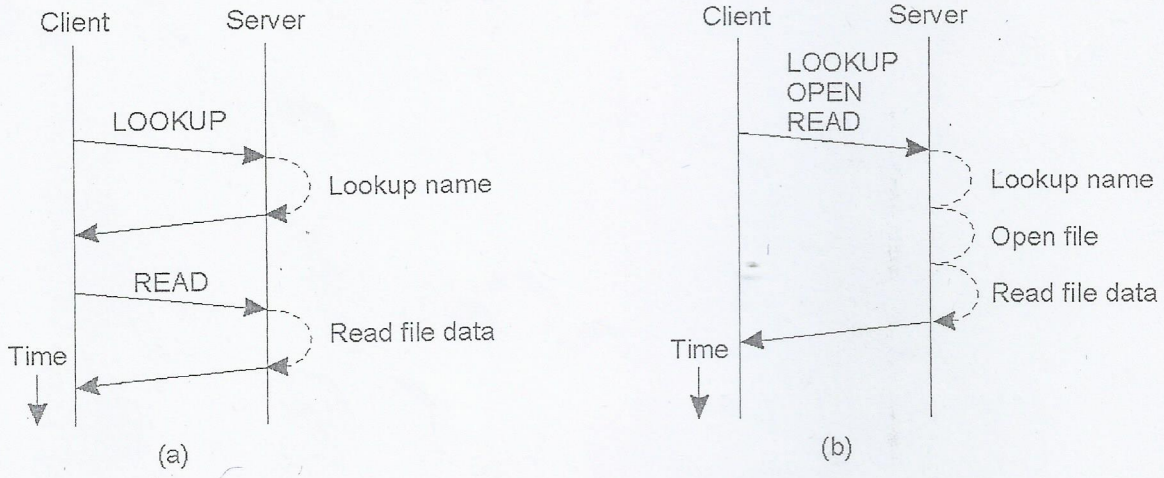
c) Before version 4, NFS used **stateless** servers.

d) The clients were in charge to maintain the status of current operations on a remote file system.

# Communication (2)

- In version 4, NFS introduced **compound operations** to improve the reduce the number of RPC calls and improve communication performance.
- This is appropriate for wide-area file systems.
- Compound operations are not handled as transactions.
- If one operation in a compound procedure fails successive operations are not executed.

# Communication (3)




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

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

principle, users do not share name spaces.

- The file named /remote/vu/mbox at client A is named /work/me/mbox at client B.
- A file's name therefore depends on how clients organize their own local name space, and where exported directories are mounted.
- The drawback of this approach in a DFS, is that sharing files becomes much harder.
- Solution: Provide each client with a name space that is partly standardized.

  e.g., each client may be using the local directory /usr/bin to mount a file system containing a std. collection of programs that are available to everyone.

  Likewise, the directory /local may be used as a std. to mount a local FS that is located on the client's host.

# Communication (4)

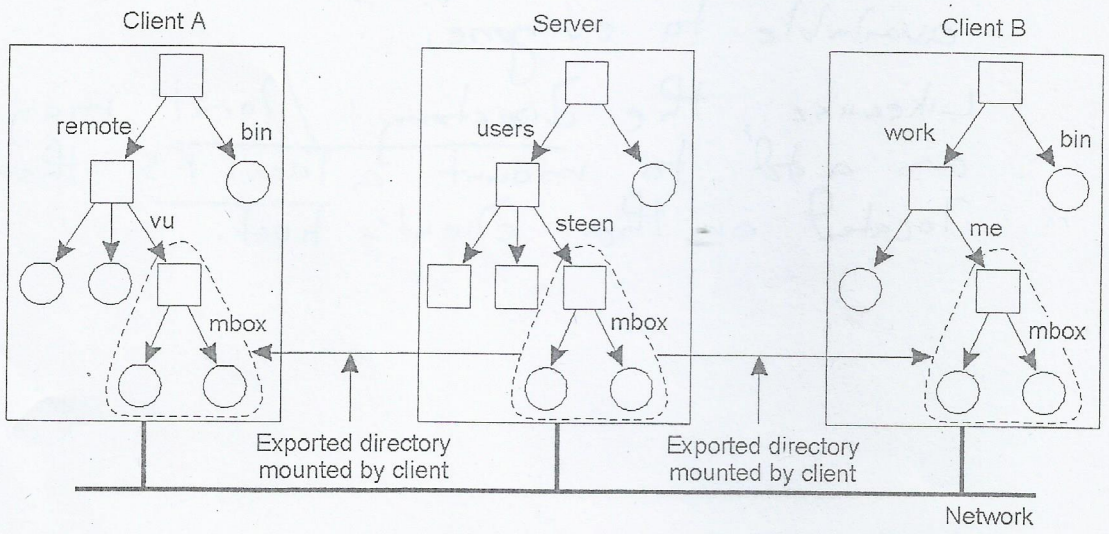
- In version 4, NFS servers maintain the status of some operations.

- This model was introduced to handle with wide-area network operations such as
  - file locking
  - cache consistency protocols
  - callback procedures.

The fundamental idea underlying the NFS naming model is to provide clients complete transparent access to a remote file system as maintained by a server.

# Naming (1)

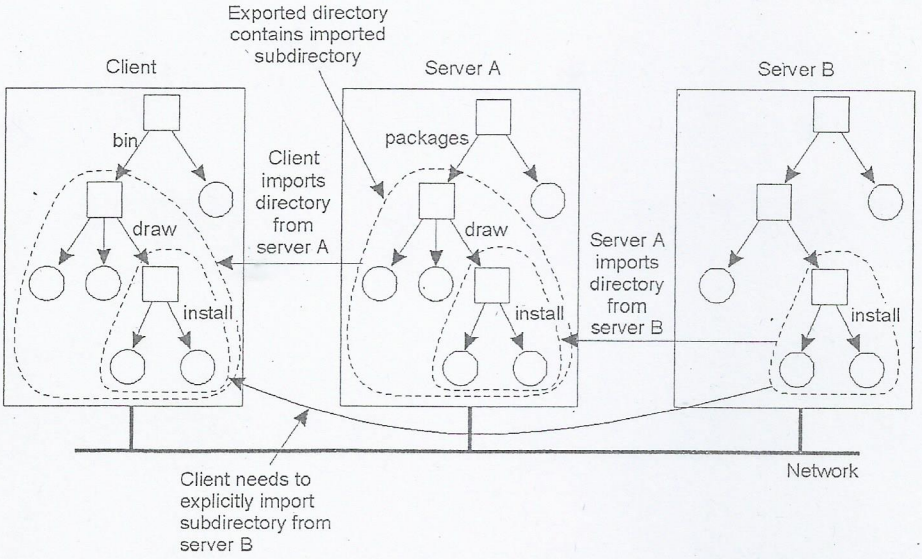
File sharing is based on **mounting** operations.




Mounting (part of) a remote file system in NFS.

# Naming (2)

An NFS server can mount directories exported from other servers, but these cannot be exported to clients.



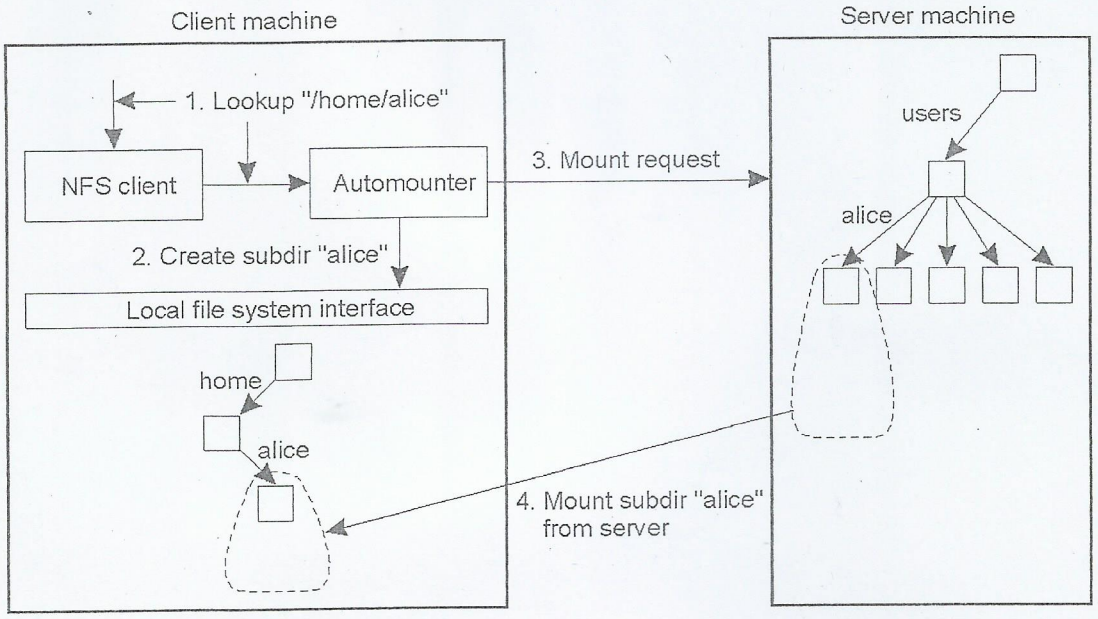

**Mounting nested directories from multiple servers in NFS.**

# Automounting (1)

- When a file system should be mounted on a client ?

- An automatic procedure is implemented by an **automounter** for NFS that
  - mount home directories of users when they log into the client and
  - mount other file system on demand (when files are accessed).

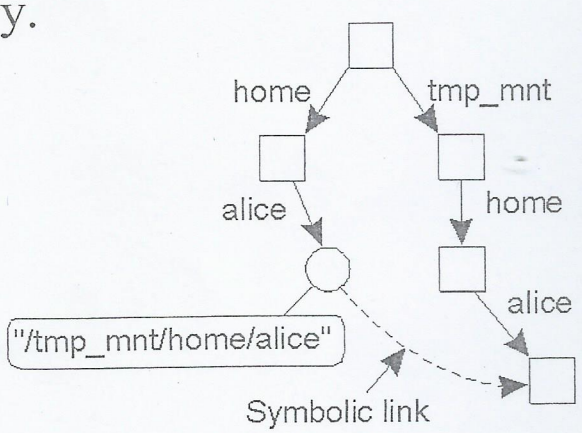
# Automounting (2)




A simple automounter for NFS.

# Automounting (3)

- To avoid to call the automounter whenever a file is read, directories can be mounted on a special sub-directory and using a symbolic link to each mounted directory.




Using symbolic links with automounting.

# File Attributes (1)

- NFS file attributes are divided between two groups: 12 **mandatory** (supported by every implementation) and 43 **recommended** attributes.

| 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.**

# 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.**

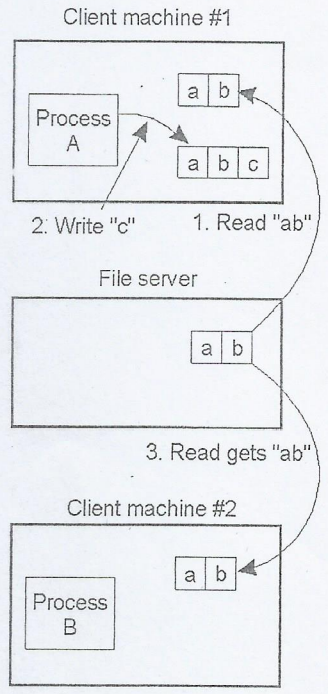
# Semantics of File Sharing (1)

- According to the **UNIX semantics** in a sequential system that allows to share files
  - a read after a write, returns the value just written
  - after two successive writes a read operation returns the value stored by the last write.

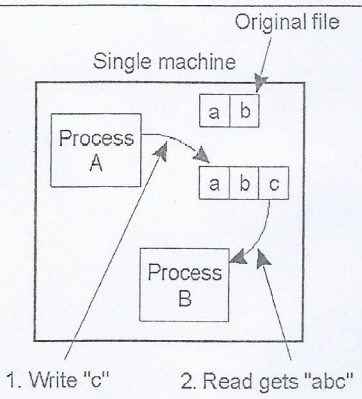
- In a distributed system, UNIX semantics can be assured if there is only one file server and clients do not cache files.

# Semantics of File Sharing (2)

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




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



# Semantics of File Sharing (3)

- Although NFS in theory uses the remote access model, most implementation use local caches, so they in practice use the upload/download model.

- NSF implements the **session semantics**:

  *changes to an open file are initially visible only to the process that modified the file . When the file is closed all the changes are visible to other processes (or machines).*

- What happens when two processes caches and modify a file?

# Semantics of File Sharing (4)

| 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.**

# File Locking in NFS (1)

- NFS version 4 use a file locking method.

# File Locking in NFS (1)

- NFS version 4 use a file locking method.
- Read locks are not mutually exclusive.
- Write lock is exclusive.

| 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.**

# File Locking in NFS (2)

- NFS implements an implicit way to lock a file: **share reservation**

Current file denial state

Request access

| | NONE | READ | WRITE | BOTH |
|---|---|---|---|---|
| READ | Succeed | Fail | Succeed | Fail |
| WRITE | Succeed | Succeed | Fail | Fail |
| BOTH | Succeed | Fail | Fail | Fail |

(a)

Requested file denial state

Current access state

| | NONE | READ | WRITE | BOTH |
|---|---|---|---|---|
| READ | Succeed | Fail | Succeed | Fail |
| WRITE | Succeed | Succeed | Fail | Fail |
| BOTH | Succeed | Fail | Fail | Fail |

(b)

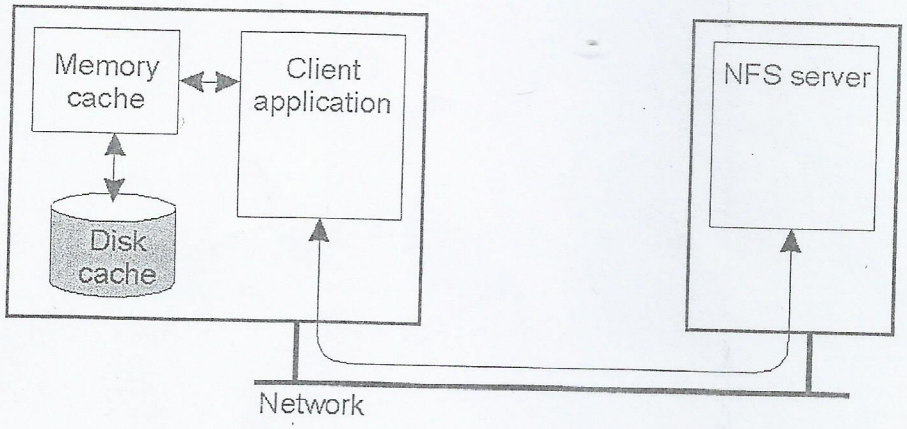
The result of an *open* operation on an already opened by another client with share reservations in NFS.

(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.**

# NFS Client Caching (1)

- NFS version 4 provides Client-side caching including a Memory cache and a Disk cache.
- File data, attributes, handles, and directories can be cached.

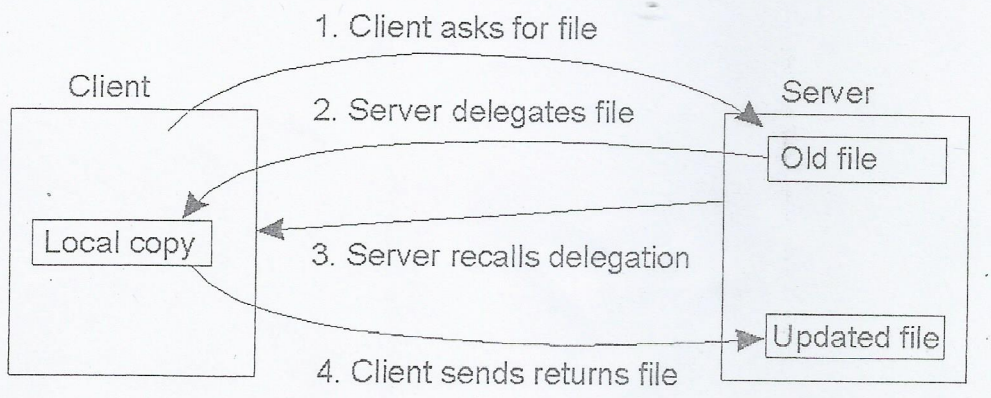


# NFS Client Caching (2)

- Caching of file data uses the session semantics: modification of cached data must be flushed to the server when a client closes the file.
- Data can be retained in the cache, but if the file will be re-opened they must be revalidated.
- NFS uses **open delegation** to delegate some rights to a client that opened a file.
- The client can take some decisions without asking the server. Some other decisions remain to the server.

# NFS Client Caching (3)

- An NFS may need to recall a delegation when another client on a different machine asks for access rights to a file.
- The **callback** mechanism is used to recall file delegation.



# NFS Client Caching (4)

- Attribute values, file handles, and directories can be cached, but modifications to those values must be sent to the server.

- Cache entries are automatically invalidated after a certain amount of time. This oblige clients to revalidate them before to use them again.

- NFS v4 provides a support for file system **replication** through a list of locations of a file system.

# NFS Fault Tolerance

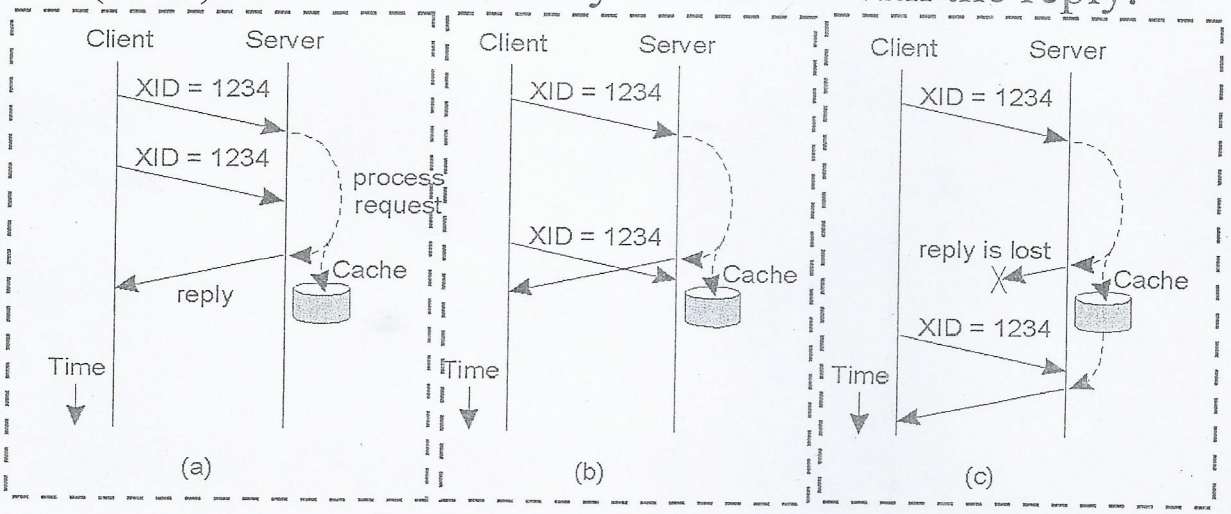
- As NFS v4 provides stateful servers (e.g., file locking, open delegation), fault tolerance and recovery mechanisms need to be designed to handle with RPC failures.

- RPC may use TCP or UDP protocols.

- RPC may incurs in duplicate requests when an RPC reply is lost; so the server can carry out the request more than one time.

# Duplicate-Request Cache

Each RPC request from a client carries a unique transaction id (**XID**) and it is cached by the server with the reply.




**Three situations for handling retransmissions.**

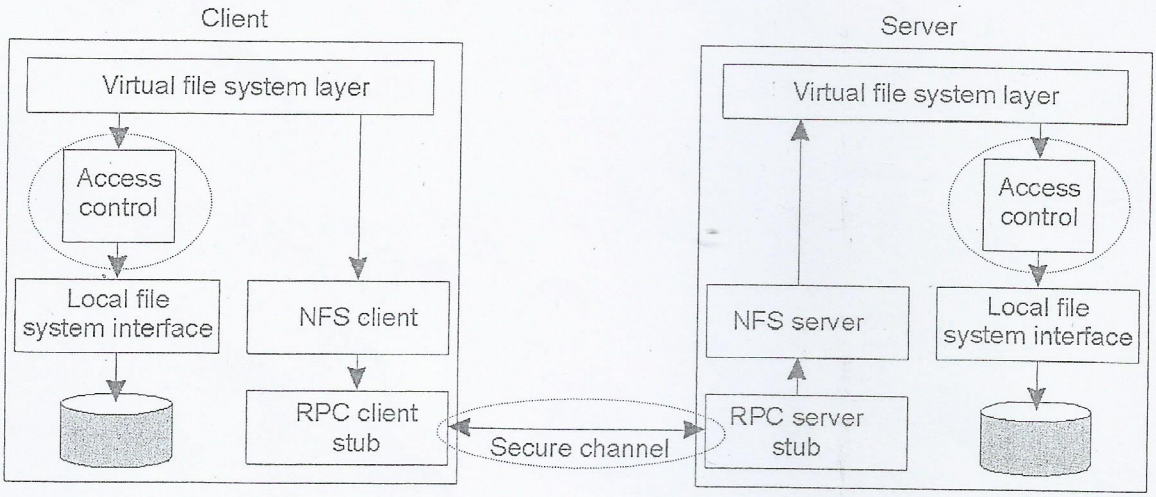
**(a) The request is still in progress**

**(b) The reply has just been returned**

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

# NFS Security

Security in NFS is mainly based on **secure channels** and **file access control**.




The NFS security architecture.