

CSMC 412

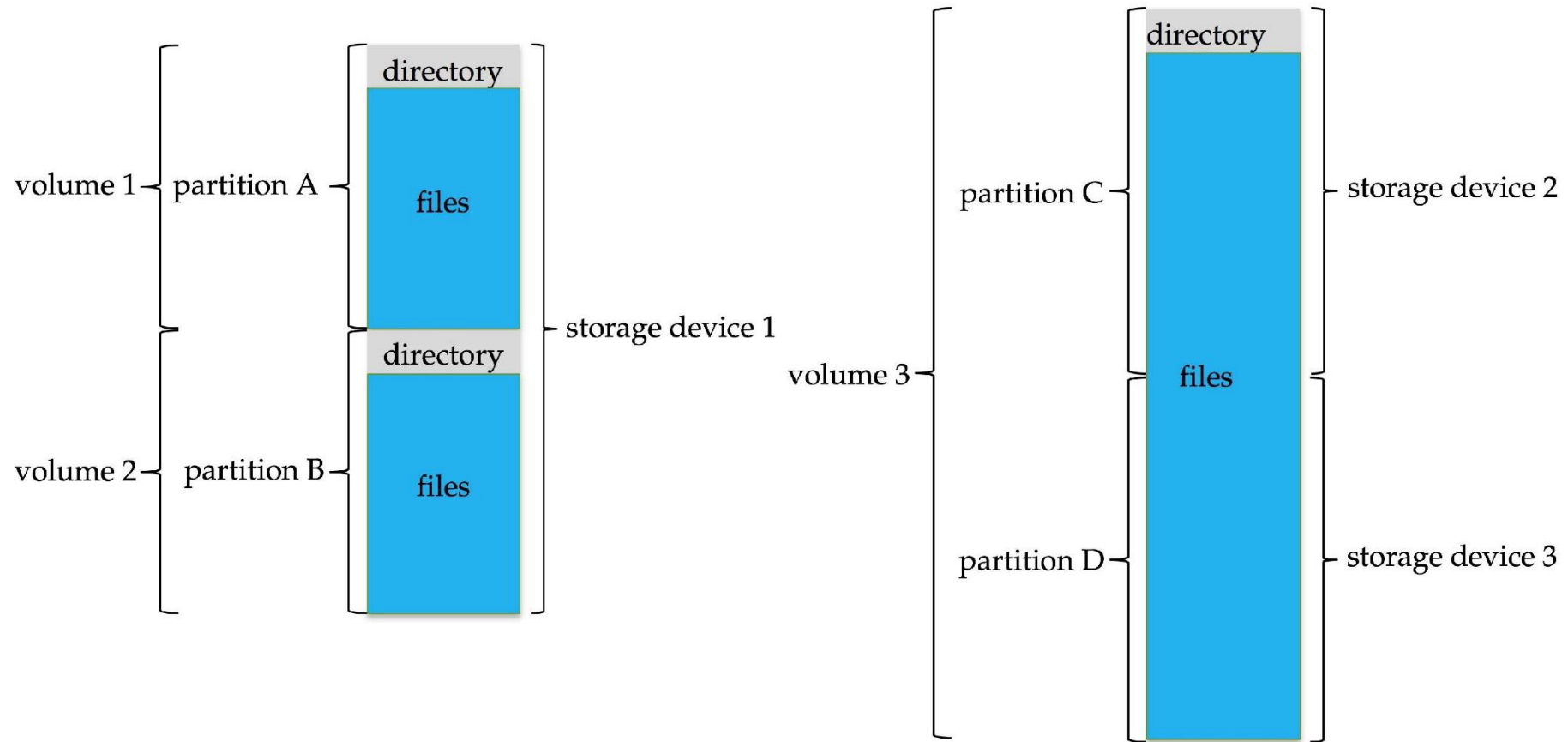
Operating Systems Prof. Ashok K Agrawala

Set 16

File System Internals

- File Systems
- File-System Mounting
- Partitions and Mounting
- File Sharing
- Virtual File Systems
- Remote File Systems
- Consistency Semantics
- NFS

A Typical Storage Device Organization



File Systems

- Files are stored on secondary storage devices
 - SSD
 - Hard Disks
 - Optical Drives, Etc.
- System may have multiple storage devices – sliced up in partitions
- Partitions hold volumes
- Volumes hold File Systems
- Volume may span multiple partitions

File Name Space

- Naming structure for files
- Names have to be unique
- Support for grouping files in folders/directories
- Full name -- Path name
- Local name – with respect to the current directory

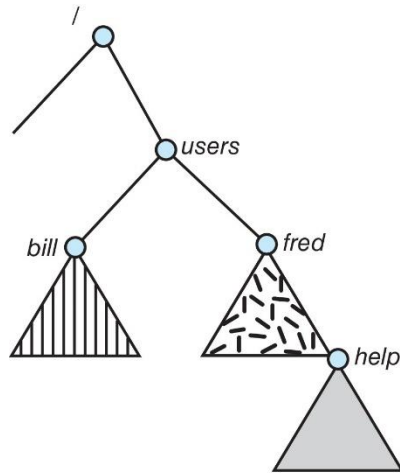
Mounting

- A File System must be mounted before it is available to processes
- Directory structure may be built out of multiple volumes
- Mounting - to make them available within the file system name space.
- Mount Point – Location within the file structure where the file system is to be attached.
 - OS examines the structure to determine the type of file system
- Mount Point is an empty directory.

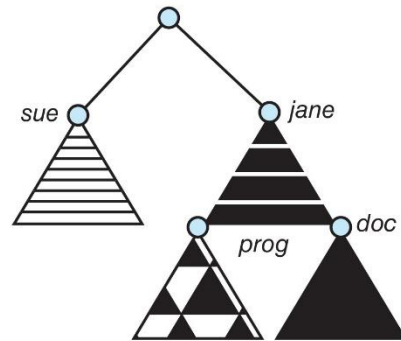
Mounting

- OS verifies that the device contains a valid file system
 - Reads the device directory
 - Confirms the format of the directory
- Makes entries in the empty directory for the Mount Point to point to the device directory
- Can traverse the directory structure going from file system to file system

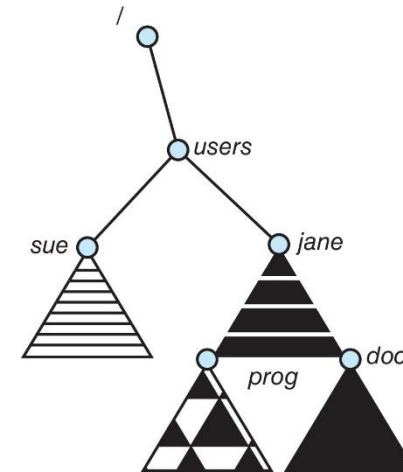
Mounting – Volume mounted at /users



(a)



(b)



Partitions and Mounting

- Partition can be a volume containing a file system (“cooked”) or **raw** – just a sequence of blocks with no file system
- Master Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
 - Or a boot management program for multi-os booting
- **Root partition** contains the OS, other partitions can hold other OSes, other file systems, or be raw
 - Mounted at boot time
 - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
 - Is all metadata correct?
 - If not, fix it, try again
 - If yes, add to mount table, allow access

File Sharing

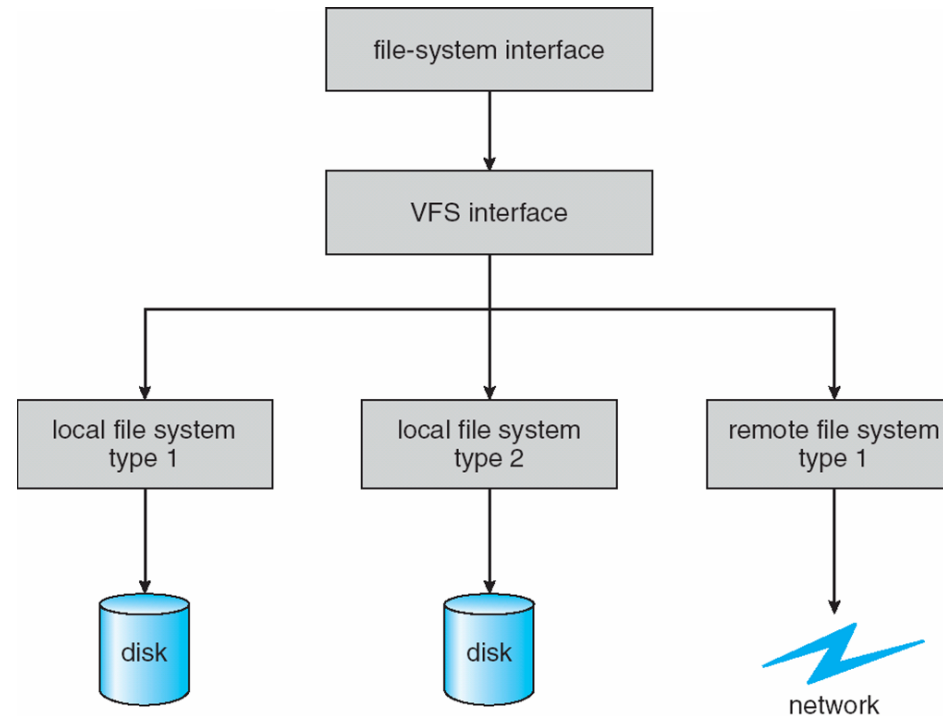
- Multiple Users need to share files when collaborating
 - Sharing
 - Naming
 - Protection
- System mediates file sharing
 - A can access files of B by default, or
 - B must grant explicit access to A
- Maintain Additional attributes
 - Owner/Group/Public attributes
 - Access Control List

Virtual File Systems

- **Virtual File Systems (VFS)** on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
 - Separates file-system generic operations from implementation details
 - Implementation can be one of many file systems types, or network file system
 - Implements **vnodes** which hold inodes or network file details
 - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems (Cont.)

- The API is to the VFS interface, rather than any specific type of file system



Virtual File System Implementation

- For example, Linux has four object types:
 - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
 - Every object has a pointer to a function table
 - Function table has addresses of routines to implement that function on that object
 - For example:
 - `int open(. . .)` —Open a file
 - `int close(. . .)` —Close an already-open file
 - `ssize_t read(. . .)` —Read from a file
 - `ssize_t write(. . .)` —Write to a file
 - `int mmap(. . .)` —Memory-map a file

Remote File Systems

- Systems connected through the Internet
- First approach
 - Use FTP protocol to manually move files
- Second – Distributed File System
 - Remote directories are visible from a local machine
- Third – World Wide Web
 - Use a browser
 - Wrapper for ftp

Client- Server Model

- Most commonly used paradigm
 - Server has the files
 - Client wants to access them
 - Many to many relationship
- Authentication
 - Keys/Certificates...
 - Unsecure authentication is most common
- Mounting
- Handling of requests

Distributed Information Systems

- Distributed Naming Services
 - Provide uniform access to information needed for remote computing
 - Example
 - DNS – Host name to network address translation
- Microsoft's Common Internet File System (CIFS)
 - Uses network information and user authentication information for network login that the server uses to permit access
 - Active Directory – Distributed naming structure to provide single name space for users-
 - Used by all clients and servers to authenticate users.
- LDAP – Lightweight Directory Access Protocol
 - Secure distributed naming mechanism

Failure Modes

- Local file system failures
 - Corruption of metadata or hardware problems
- Remote file systems
 - Additional failure modes
 - Primarily due to network based access
- Recovery from failures
 - Provisions have to be made

Consistency Semantics

- Define how multiple users are to access a shared file system simultaneously
 - When modifications done by one user are visible to the other user
- UNIX Semantics
 - Writes to an open file by a user are visible immediately to other users who have this file open
 - Users may share pointer to current location in a file. Thus advancing by one user affects all sharing users
 - Single file image – all accesses are interleaved

The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol) and Ethernet
 - Later version support TCP also

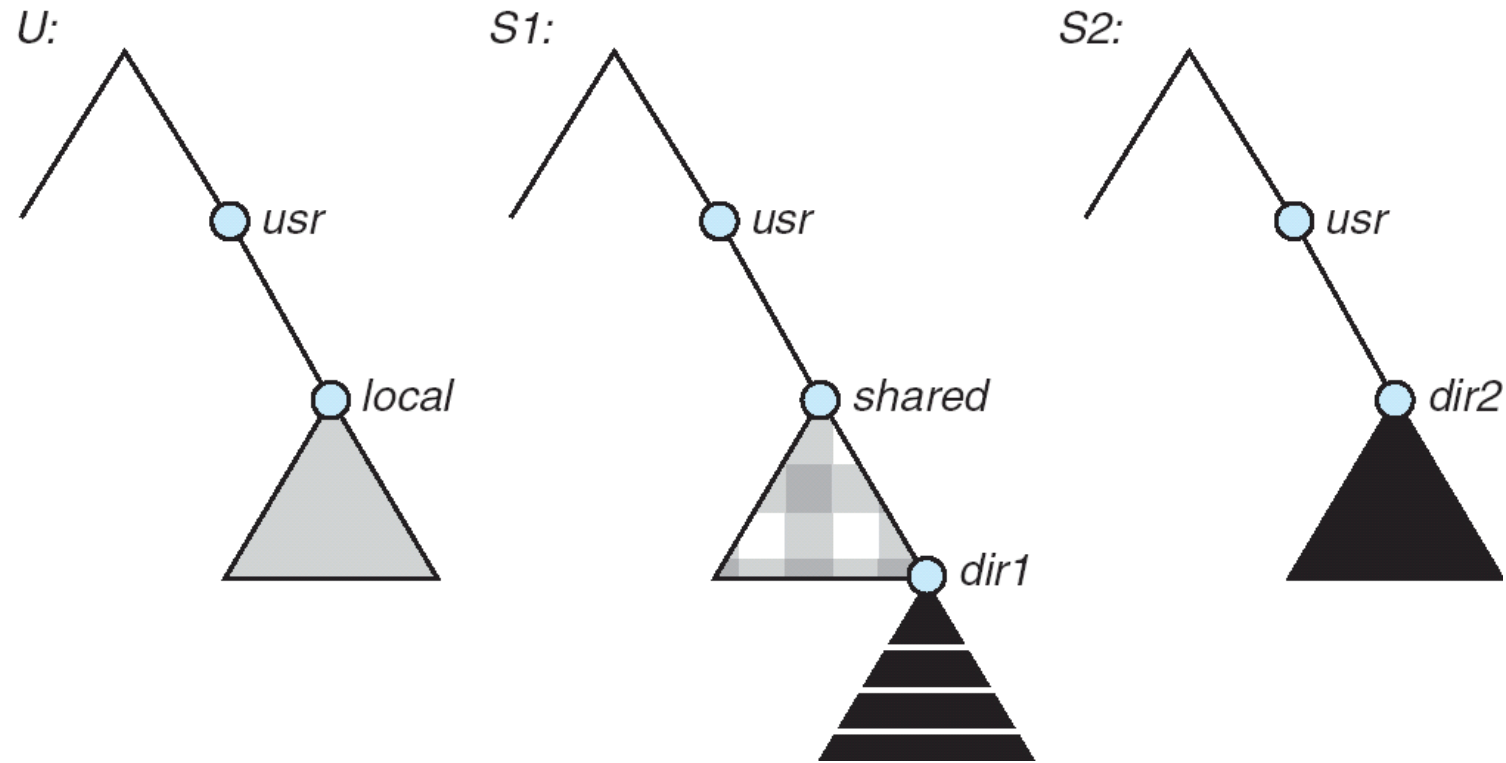
NFS (Cont.)

- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
 - A remote directory is mounted over a local file system directory
 - The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
 - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
 - Files in the remote directory can then be accessed in a transparent manner
 - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

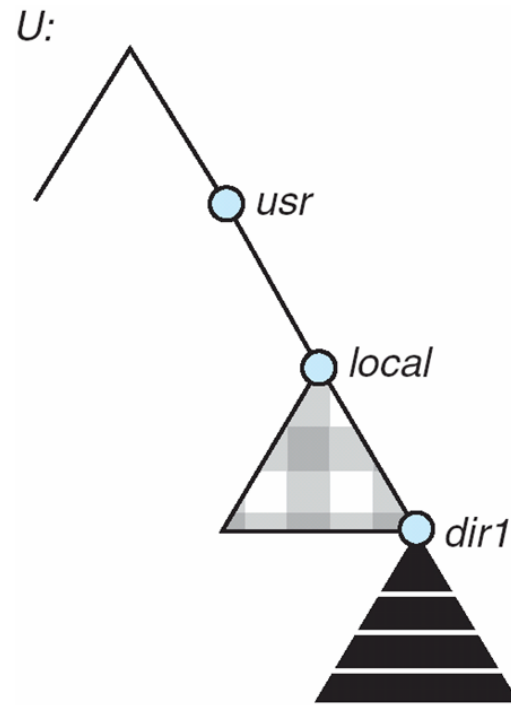
NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

Three Independent File Systems

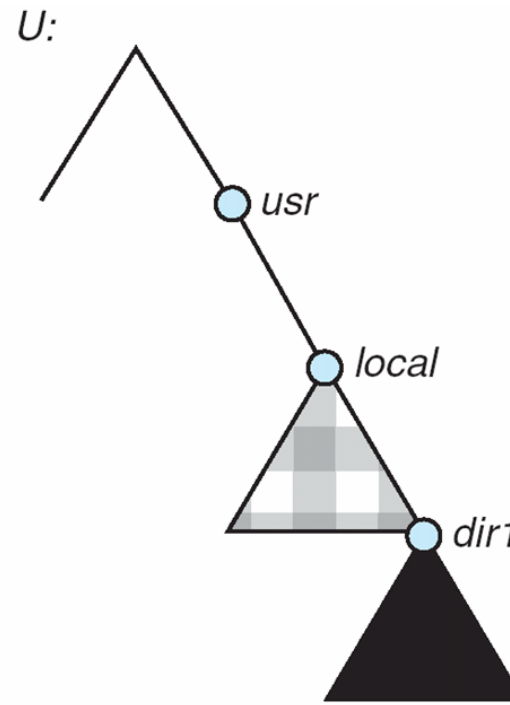


Mounting in NFS



(a)

Mounts



(b)

Cascading mounts

NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
 - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
 - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

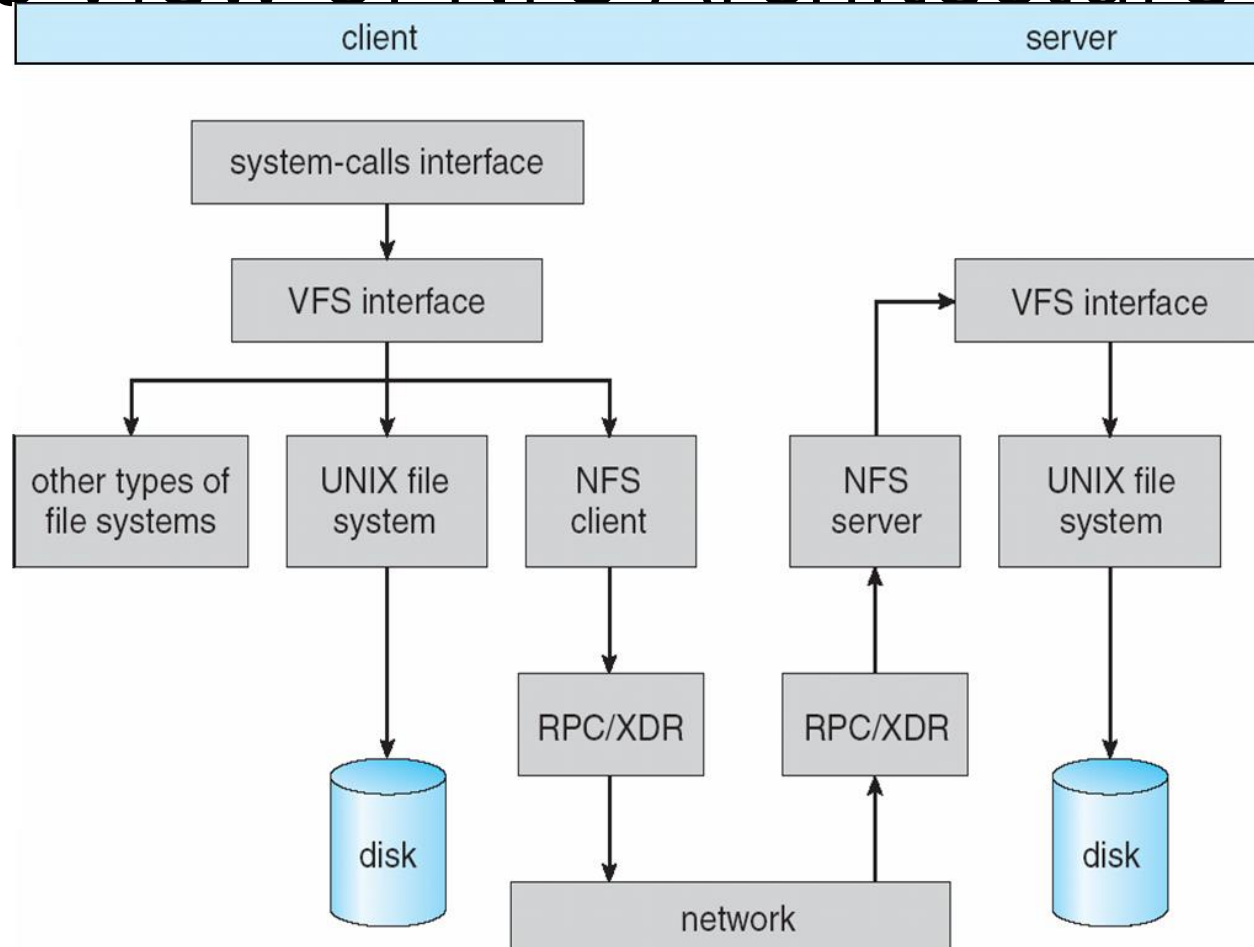
NFS Protocol

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
 - searching for a file within a directory
 - reading a set of directory entries
 - manipulating links and directories
 - accessing file attributes
 - reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments (NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms

Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open, read, write, and close** calls, and **file descriptors**)
- Virtual File System (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
 - The VFS activates file-system-specific operations to handle local requests according to their file-system types
 - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
 - Implements the NFS protocol

Schematic View of NFS Architecture



NFS Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
 - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk