

Chapter 15: File System Internals





Chapter 15: File System Internals

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





Objectives

□ a





File System

□ 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
- 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

- a
- b





Virtual File Systems

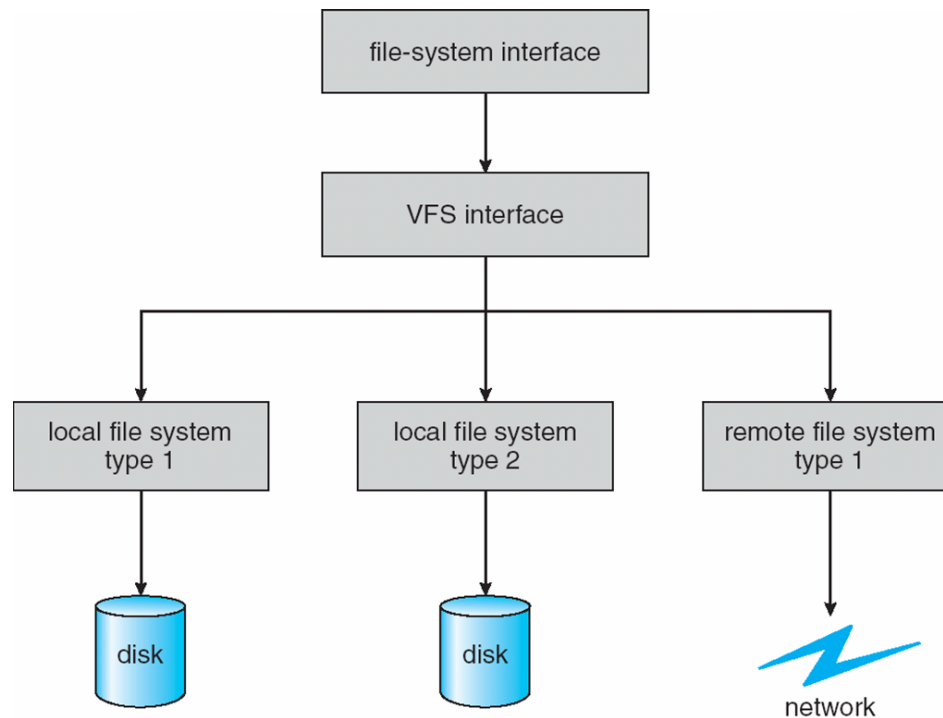
- ❑ **Virtual File Systems (VFS)** on Unix provide an object-oriented way of implementing file systems
- ❑ VFS allows the same system call interface (the API) to be used for different types of file systems
 - ❑ Separates file-system generic operations from implementation details
 - ❑ Implementation can 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





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)





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





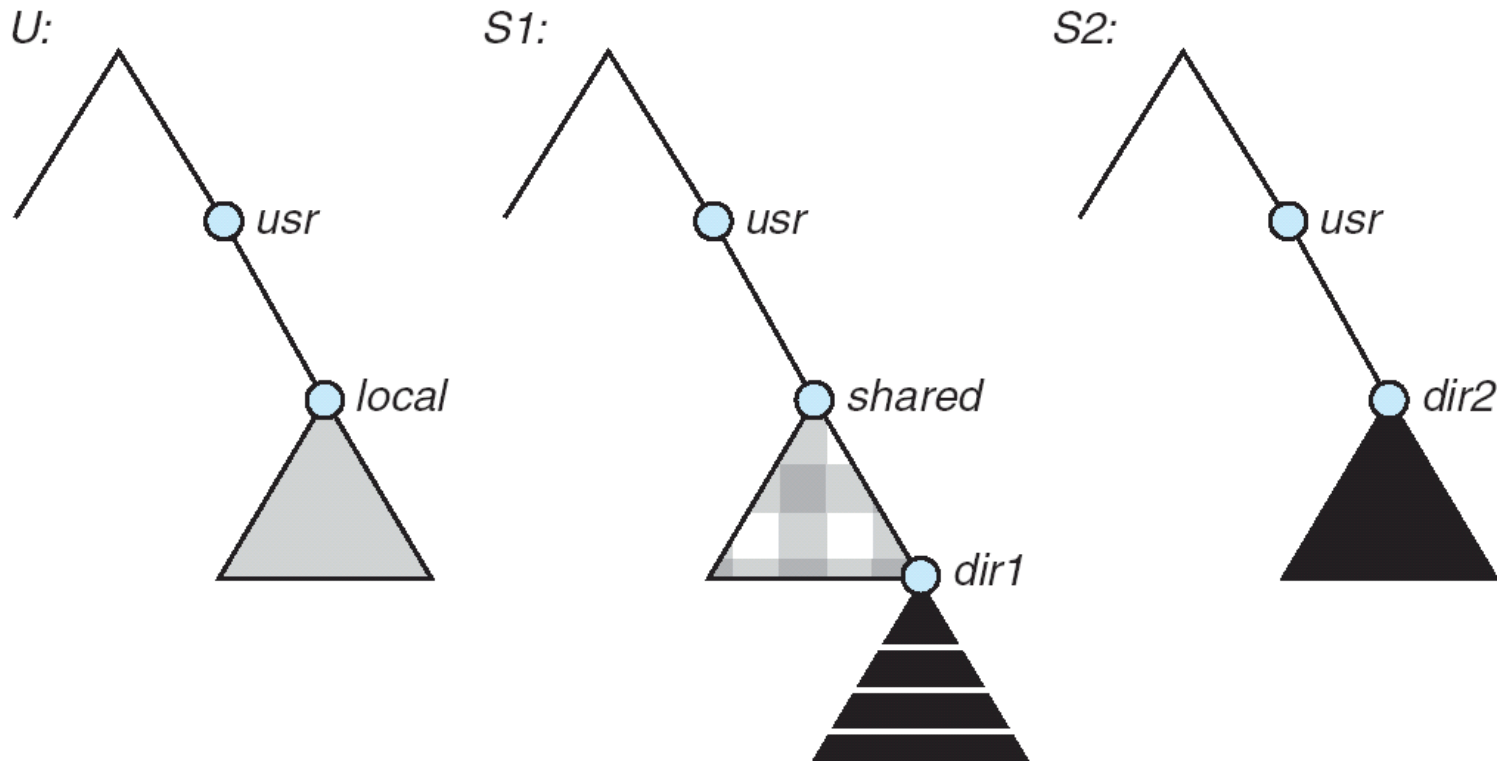
NFS (Cont.)

- 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



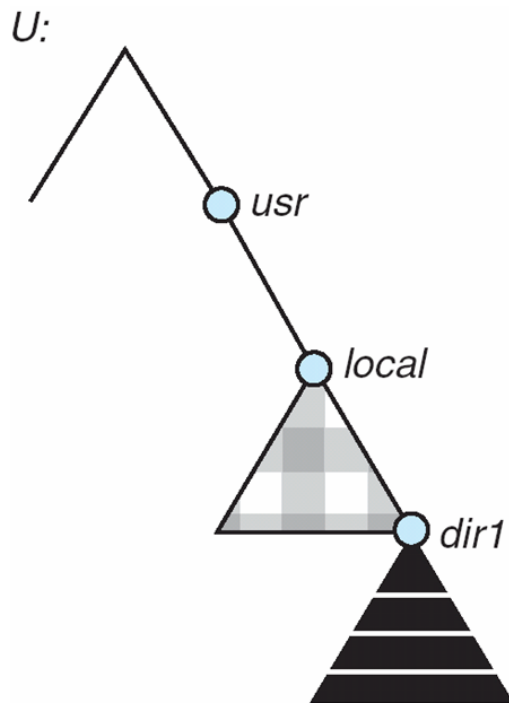


Three Independent File Systems



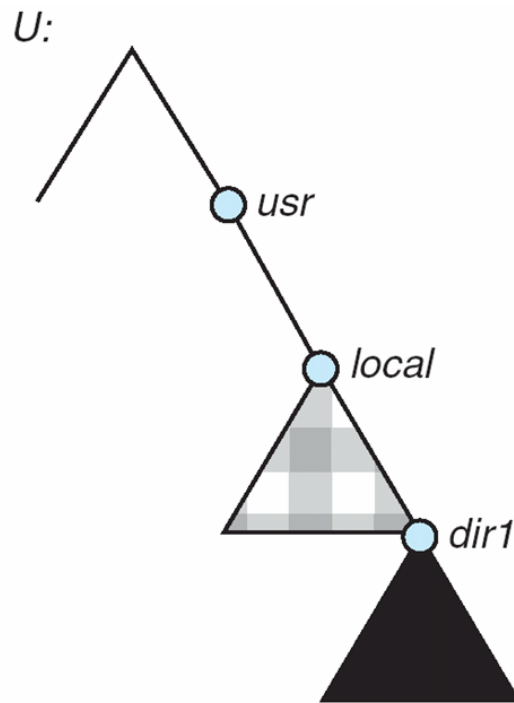


Mounting in NFS



(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





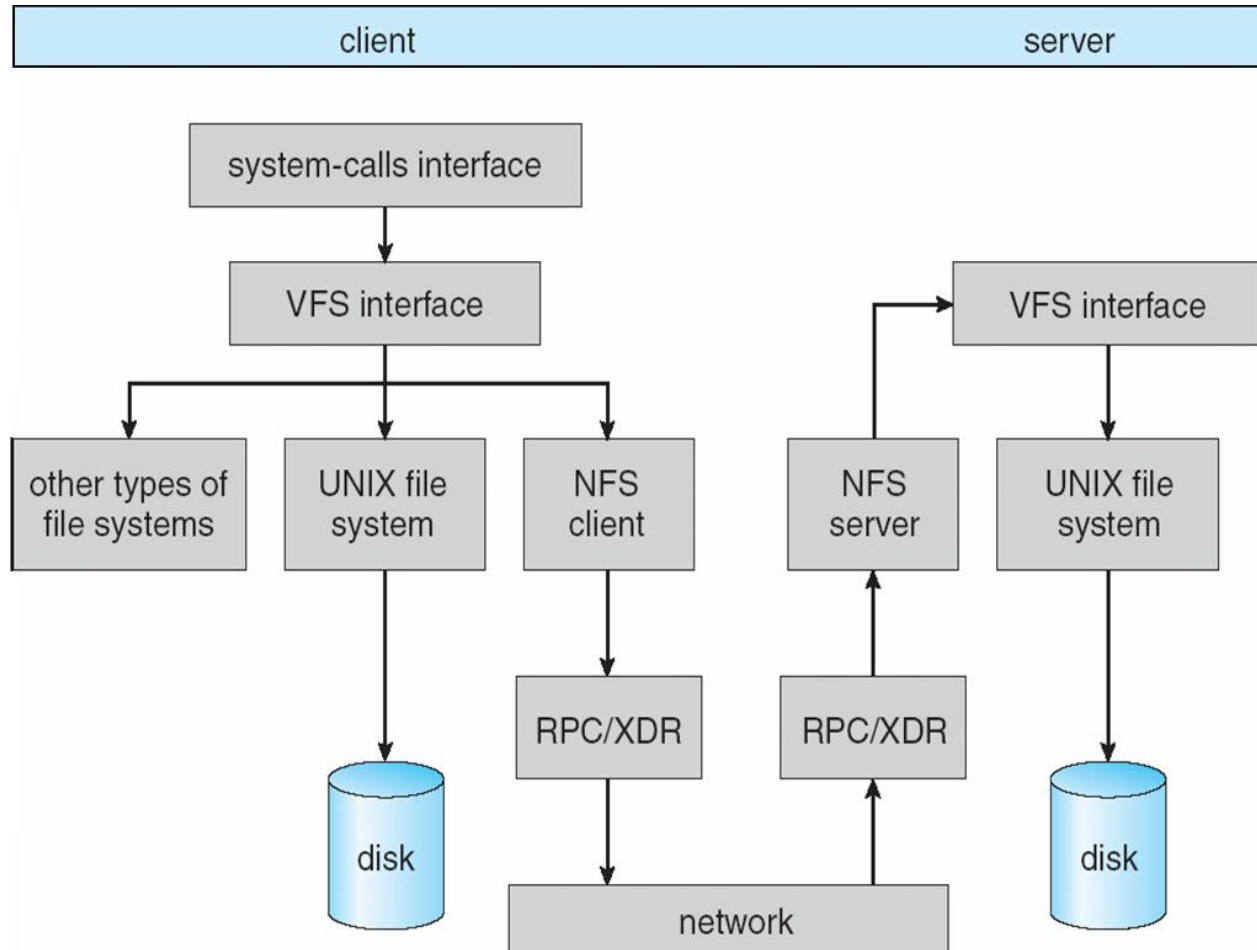
Three Major Layers of NFS Architecture

- 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



End of Chapter 15

