

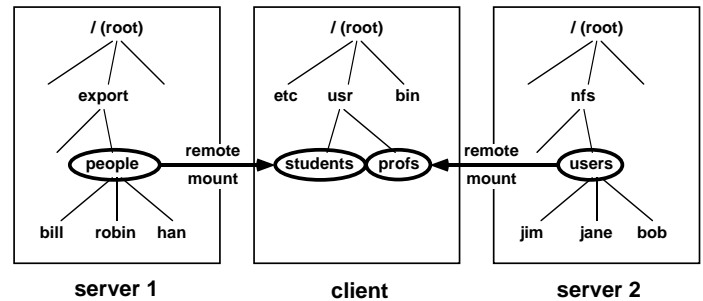
Sun's Network File System

- Designed by Sun Microsystems
 - First distributed file service designed as a project, introduced in 1985
 - To encourage its adoption as a standard
 - Definitions of the key interfaces were placed in the public domain in 1989
 - Source code for a reference implementation was made available to other computer vendors under license
 - Currently the *de facto* standard for LANs
- Provides transparent access to remote files on a LAN, for clients running on UNIX and other operating systems
 - A UNIX computer typically has a NFS client and server module in its OS kernel
 - Available for almost any UNIX and MACH
 - Client modules are available for Macintosh and PCs

1

Spring 2000, Lecture 23

Mounting Remote File Systems



- NFS supports mounting of remote file systems by client machines
 - Name space seen by each client may be different
 - Same file on server may have different path names on different clients
 - NFS does not enforce a single network-wide name space, but a uniform name space (and location transparency) can be established if desired

2

Spring 2000, Lecture 23

Mounting Remote File Systems (cont.)

- On each server
 - There is a file (usually */etc/exports*) containing the names of local file systems that are available for remote mounting
 - An access list is associated with each name, and indicates which hosts are permitted to mount that file system
- On each client
 - A modified version of the UNIX *mount* command mounts a remote file system
 - Based on RPC — specifies remote host name, pathname of a directory in the remote file system, and local name where it is to be mounted
 - Mount requests are usually performed when the system is initialized (booted)
 - Usually specified in */etc/fstab*
 - User may also be able to mount other remote file systems

3

Spring 2000, Lecture 23

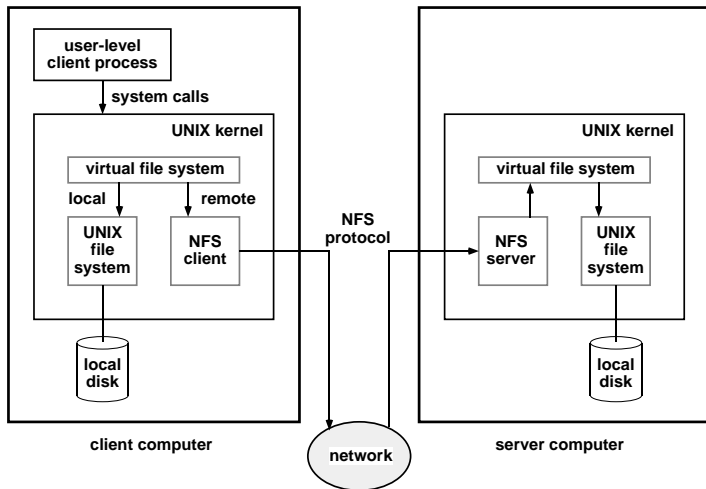
Mounting Remote File Systems (cont.)

- Remote file systems may be
 - *Hard mounted* — when a user-level process accesses a file, it is suspended until the request can be completed
 - If a server crashes, the user-level process will be suspended until recovers
 - *Soft mounted* — after a small number of retries, the NFS client returns a failure code to the user process
 - Most UNIX utilities don't check this code...
- Automounting
 - The *automounter* dynamically mounts a file system whenever an "empty" mount point is referenced by a client
 - Further accesses do not result in further requests to the automounter...
 - Unless there are no references to the remote file system for several minutes, in which case the automounter unmounts it

4

Spring 2000, Lecture 23

NFS Software Architecture



Virtual file system:

- Separates generic file-system operations from their implementation (can have different types of local file systems)
- Based on a file descriptor called a vnode that is unique networkwide (UNIX inodes are only unique on a single file system)

5

Spring 2000, Lecture 23

Distributed Naming Structures (Review)

- Mount remote directories onto local directories (possibly on demand)
 - Client-maintained mount information:
 - Used by UNIX and NFS — Sun's Network File System
 - Client maintains:
 - A set of local names for remote locations
 - A *mount table* (*/etc/fstab*) that specifies a:
 - » < remote machine name : pathname >
 - » and < local pathname >
 - At boot time, the local name is bound to the remote name
 - Afterwards, users refer to local pathname as if it were local, and the distributed OS takes care of the mapping
 - Location transparent and independent after the mount operation, but not before
 - Server-maintained mount information:
 - If files are moved to a different server, mount information need only be updated at servers

6

Spring 2000, Lecture 23

NFS Protocol

- NFS protocol provides a set of RPCs for remote file operations
 - Looking up a file within a directory
 - Manipulating links and directories
 - Creating, renaming, and removing files
 - Getting and setting file attributes
 - Reading and writing files
- NFS is stateless
 - Servers do not maintain information about their clients from one access to the next
 - There are no open-file tables on the server
 - There are no open and close operations
 - Each request must provide a unique file identifier, and an offset within the file
 - Easy to recover from a crash, but file operations must be idempotent

7

Spring 2000, Lecture 23

NFS Protocol (cont.)

- Because NFS is stateless, all modified data must be written to the server's disk before results are returned to the client
 - Server crash and recovery should be invisible to client — data should be intact
 - ✗ Lose benefits of caching
 - Solution — RAM disks with battery backup (un-interruptable power supply), written to disk periodically
- A single NFS write is guaranteed to be atomic, and not intermixed with other writes to the same file
 - However, NFS does not provide concurrency control
 - A write system call may be decomposed into several NFS writes, which may be interleaved
 - Since NFS is stateless, this is not considered to be an NFS problem

8

Spring 2000, Lecture 23

Caching in NFS

- Traditional UNIX
 - Caches file blocks, directories, and file attributes
 - Uses read-ahead (prefetching), and delayed-write (flushes every 30 seconds)
- NFS servers
 - Same as in UNIX, except server's write operations perform write-through
 - Otherwise, failure of server might result in undetected loss of data by clients
- NFS clients
 - Caches results of read, write, getattr, lookup, and readdir operations
 - Possible inconsistency problems
 - Writes by one client do not cause an immediate update of other clients' caches

9

Spring 2000, Lecture 23

Caching in NFS (cont.)

- NFS clients (cont.)
 - File reads
 - When a client caches one or more blocks from a file, it also caches a timestamp indicating the time when the file was last modified on the server
 - Whenever a file is opened, and the server is contacted to fetch a new block from the file, a validation check is performed
 - Client requests last modification time from server, and compares that time to its cached timestamp
 - If modification time is more recent, all cached blocks from that file are invalidated
 - Blocks are assumed to valid for next 3 seconds (30 seconds for directories)
 - File writes
 - When a cached page is modified, it is marked as dirty, and is flushed when the file is closed, or at the next periodic flush
 - Now two sources of inconsistency: delay after validation, delay until flush

10

Spring 2000, Lecture 23

Distributed Naming Structures (Review)

- Single name space for remote and local directories
 - Names of form: */.../machine/fs/pathname*
 - Used by:
 - CMU's Andrew, now in OSF's Distributed Computing Environment (DCE)
 - Berkeley's Sprite
 - File names are always the same, whether file is remote or local
 - As clients access a file, the server sends a copy to the client's workstation, and the workstation caches the file
 - In Andrew, local disks are used
 - In Sprite, large memories are used, and workstations are diskless
 - More details on these two next time...
 - Location independent, not location transparent

11

Spring 2000, Lecture 23

CMU's Andrew File System

- Designed by Carnegie Mellon University
 - Developed during mid-1980s as part of the Andrew distributed computing environment
 - Designed to support a WAN of more than 5000 workstations
 - Much of the core technology is now part of the Open Software Foundation (OSF) Distributed Computing Environment (DCE), available for most UNIX and some other operating systems
- Provides transparent access to remote files on a WAN, for clients running on UNIX and other operating systems
 - Access to all files is via the usual UNIX file primitives
 - Compatible with NFS — servers can mount NFS file systems

12

Spring 2000, Lecture 23

Caching in Andrew

- When a remote file is accessed, the server sends the entire file to the client
 - The entire file is then stored in a disk cache on the client computer
 - Cache is big enough to store several hundred files
- Implements session semantics
 - Files are cached when opened
 - Modified files are flushed to the server when they are closed
 - Writes may not be immediately visible to other processes
- When client caches a file, server records that fact — it has a *callback* on the file
 - When a client modifies and closes a file, other clients lose their callback, and are notified by server that their copy is invalid

How Can Andrew Perform Well?

- Most file accesses are to files that are infrequently updated, or are accessed by only a single user, so the cached copy will remain valid for a long time
- Local cache can be big — maybe 100 MB — which is probably sufficient for one user's working set of files
- Typical UNIX workloads:
 - Files are small, most are less than 10kB
 - Read operations are 6 times more common than write operations
 - Sequential access is common, while random access is rare
 - Most files are read and written by only one user; if a file shared, usually only one user modifies it
 - Files are referenced in bursts