



CS530: Advanced Operating Systems (Fall 2004)

Design and Implementation of the Sun Network Filesystem

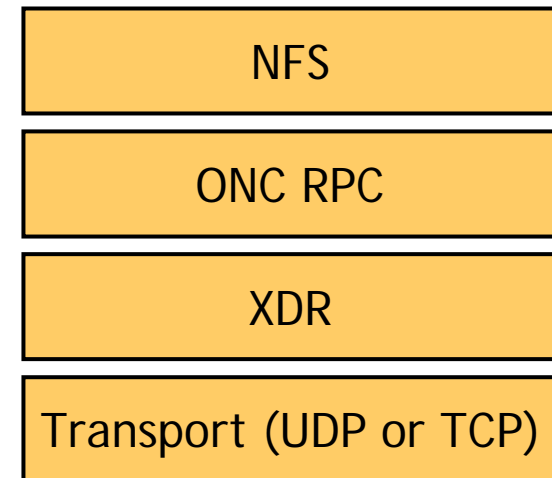
(R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon,
USENIX '85)

KAIST

Overview (1)

▪ Sun NFS (Network File System)

- A network protocol that makes files stored on a file server accessible to any computer on a network.
- An NFS client can also be a server, with remote and local mounted filesystems freely intermixed.
- An industry standard for file sharing on local networks since the 1980s.
 - RFC 1094 (NFS v2)
 - RFC 1813 (NFS v3)
 - RFC 3530 (NFS v4)
- Built on top of RPC and XDR.



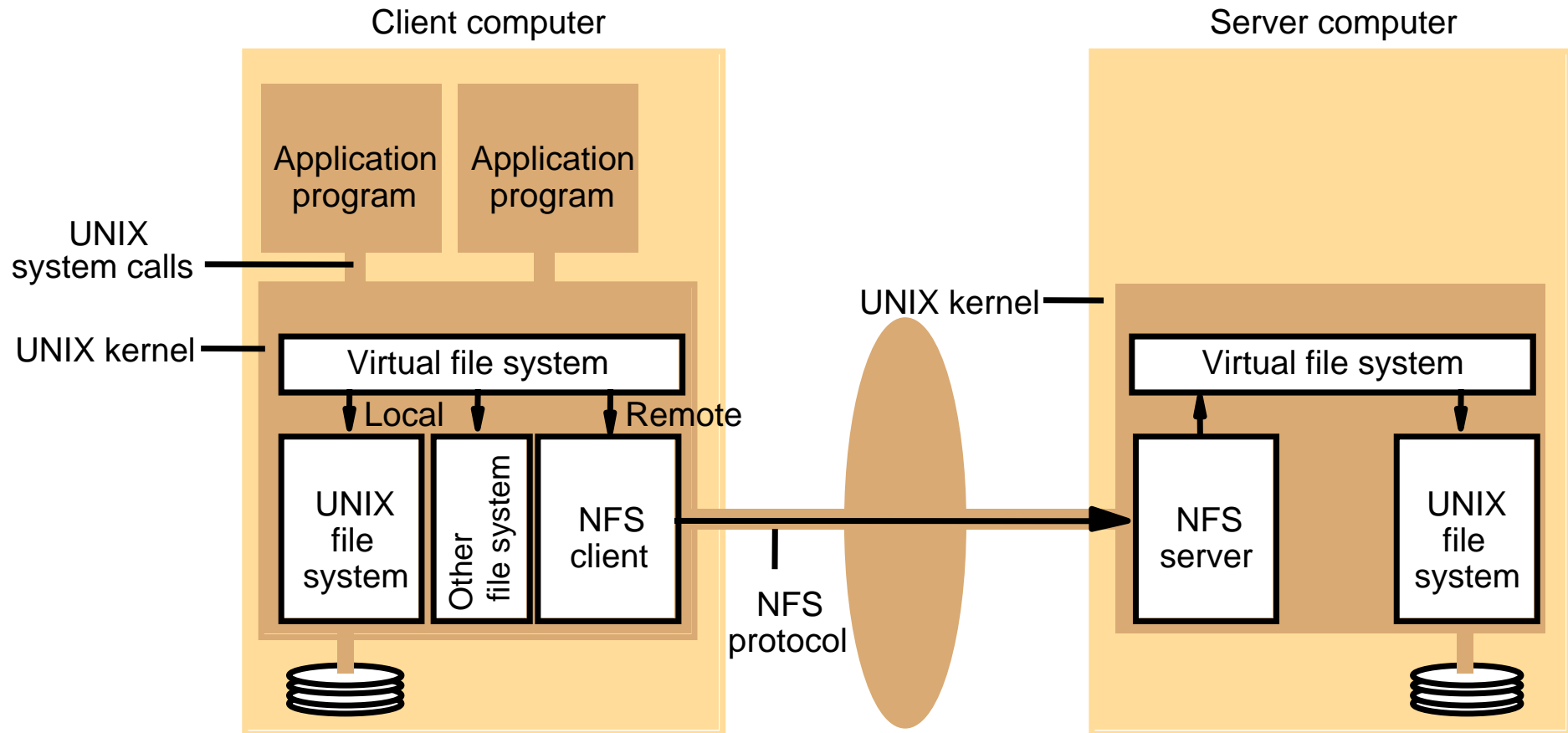
Overview (2)

■ Design Goals

- Machine and OS independence
 - Machine, OS, network architecture, and transport protocols
- Crash recovery
 - Clients should be able to recover easily from server crashes.
- Transparent access
 - Access remote files in exactly the same way as local files.
- UNIX semantics maintained on client
- Reasonable performance
 - As fast as the Sun Network Disk protocol (ND).
 - About 80% as fast as a local disk.

Operating System

NFS Architecture (1)



NFS Architecture (2)

▪ VFS (Virtual File System)

- VFS interface
 - Defines procedures that operate on the file system as a whole.
 - One VFS structure for each mounted filesystem.
- Vnode (Virtual Node) interface
 - Defines procedures that operate on a individual file.
 - One vnode per open file.
 - The vnode contains an indicator to show whether a file is local or remote.
 - If the file is local, the vnode contains a reference to the corresponding inode.
 - If the file is remote, it contains the file handle of the remote file.

NFS Architecture (3)

▪ Stateless Server

- The server does not keep track of any past requests.
 - The parameters contain all of the information necessary to complete the call.
- Avoid complex crash recovery and simplify protocol.
 - Server can crash and reboot; the client resends requests until a response is received.
 - When a client crashes no recovery is necessary for either the client or the server.
- Implies that I/O on server must be synchronous.
 - All operations that modify the filesystem must be committed to stable-storage before the RPC can be acknowledged.

NFS Architecture (4)

■ NFS Filehandle

- A reference to a file or directory that is independent of the filename.
- Opaque to the client
 - The server creates filehandles and only the server can interpret the data contained within them.
 - $\langle \text{filesystem id, inode number, the inode generation number} \rangle$
 - 32B fixed-length in NFS v2, up to 64B variable length in NFS v3.
- Clients expect filehandles to be persistent, i.e. to survive a server crash.
- It may become 'stale' at any time.
 - For example, if someone moves the file.

NFS Procedures (1)

- **Procedure 0: NULL** (Do nothing)
 - `null()` returns `()`
 - Do nothing procedure to ping the server and measure round trip time.
- **Procedure 1: GETATTR** (Get file attributes)
 - `getattr(fh)` returns `(attr)`
 - Retrieves file attributes. (similar to `stat()` system call.)
 - Most client NFS protocol implementations implement a time-bounded attribute caching scheme to reduce over-the-wire attribute checks.

NFS Procedures (2)

■ Procedure 2: SETATTR (Set file attributes)

- `setattr(fh, attr)` returns `(attr)`
 - Sets the mode, uid, gid, size, access time, and modify time of a file.
- The size field is used to request changes to the size of a file.
 - `size == 0`: the file is truncated.
 - `size < the current size`: data from new size to the end of the file is discarded.
 - `size > the current size`: logically zeroed data bytes are added to the end of the file.
- SETATTR is not guaranteed atomic.
 - A failed SETATTR may partially change a file's attributes.

NFS Procedures (3)

- **Procedure 4: LOOKUP** (Lookup file name)
 - `lookup(dirfh, name)` returns `(fh, attr)`
 - Returns a new filehandle and attributes for the named file in a directory.
 - Normally, the filename is a single pathname component.
 - Special filenames:
 - `“.”` for the current directory, `“..”` for the parent directory.
 - If the name represents a symbolic link, the server will return the filehandle for the symbolic link.
 - Not the file it points to.
 - A separate protocol (the MOUNT protocol) is used to get the root filehandle.

Operating System

NFS Procedures (4)

- **Procedure 5: READLINK** (Read from symbolic link)
 - `readlink(fh)` returns (string)
 - Returns the string which is associated with the symbolic link file.
- **Procedure 6: Read** (Read from file)
 - `read(fh, offset, count)` returns (attr, data)
 - Returns up to count bytes of data from a file starting offset bytes into the file.
 - If the server returns a “short read” to the client, the client will assume that the last byte of data is the EOF.

NFS Procedures (5)

- **Procedure 8: WRITE** (Write to file)
 - `write(fh, offset, count, data)` returns `(attr)`
 - Writes *count* bytes of data to a file beginning *offset* bytes from the beginning of the file.
 - The write operation is atomic.
 - Data from this WRITE will not be mixed with data from another client's WRITE.
 - If the write cannot be completed in its entirety, an error must be returned.
 - The act of writing data to a file will cause the mtime of the file to be updated.
 - The data must be written to stable storage.

NFS Procedures (6)

- **Procedure 9: CREATE** (Create a file)
 - `create(dirfh, name, attr)` returns `(newfh, attr)`
 - Creates a new file and returns its fhandle and attributes.
 - The CREATE does not support “exclusive create” semantics.
 - `O_EXCL` flag in `open()` or `creat()`: if the file already exists it is an error and the `open()` or `creat()` will fail.
 - Programs which rely on it for performing locking tasks will contain a race condition on NFS.
 - Exclusive create is supported by NFS v3.

NFS Procedures (7)

- **Procedure 10: REMOVE** (Remove file)
 - `remove(dirfh, name)` returns (status)
 - Removes a file from a directory.
 - Since a file may have multiple names, the REMOVE may have no effect on the file data.
 - The filehandle for the file may continue to be valid after the remove is complete.
- **Procedure 11: RENAME** (Rename file)
 - `rename(dirfh, name, tofh, toname)` returns (status)
 - Renames the file *name* in the directory *dirfh*, to *toname* in the directory *tofh*.
 - The RENAME operation must be atomic to the client.
 - It cannot be interrupted in the middle or leave a partial result on failure.

Operating System

NFS Procedures (8)

- **Procedure 12: LINK** (Create link to file)
 - `link(dirfh, name, tofh, toname)` returns (status)
 - Creates the file *toname* in the directory *tofh*, which is a link to the file *name* in the directory *dirfh*.
 - The link count for the file is increased by one.
- **Procedure 13: SYMLINK** (Create symbolic link)
 - `symlink(dirfh, name, string)` returns (status)
 - Creates a symbolic link *name* in the directory *dirfh* with value *string*.
 - The server does not interpret the string argument.
 - It is only stored in the newly created file.

NFS Procedures (9)

- **Procedure 14: MKDIR** (Create directory)
 - `mkdir(dirfh, name, attr)` returns `(fh, newattr)`
 - Creates a new directory *name* in the directory *dirfh* and returns the new fhandle and attributes.
- **Procedure 15: RMDIR** (Remove directory)
 - `rmdir (dirfh, name)` returns `(status)`
 - Removes the empty directory *name* from the parent directory *dirfh*.
- **Procedure 17: STATFS** (Get filesystem attributes)
 - `statfs(fh)` returns `(fsstats)`
 - Returns filesystem information such as block size, number of free blocks, etc.

NFS Procedures (10)

- **Procedure 16: READDIR** (Read from directory)
 - `readdir(dirfh, cookie, count)` returns (entries)
 - Returns up to *count* bytes of directory entries from the directory *dirfh*.
 - Each entry contains a file name, file id, and an opaque pointer to the next directory entry called a cookie.
 - The cookie is used in subsequent READDIR calls to start reading at a specific entry in the directory.
 - A cookie can become invalid if two READDIRs are separated by one or more operations that change the directory in some way.
 - A READDIR call with the cookie of zero returns entries starting with the first entry in the directory.

NFS MOUNT Protocol (1)

- **NFS MOUNT Protocol**

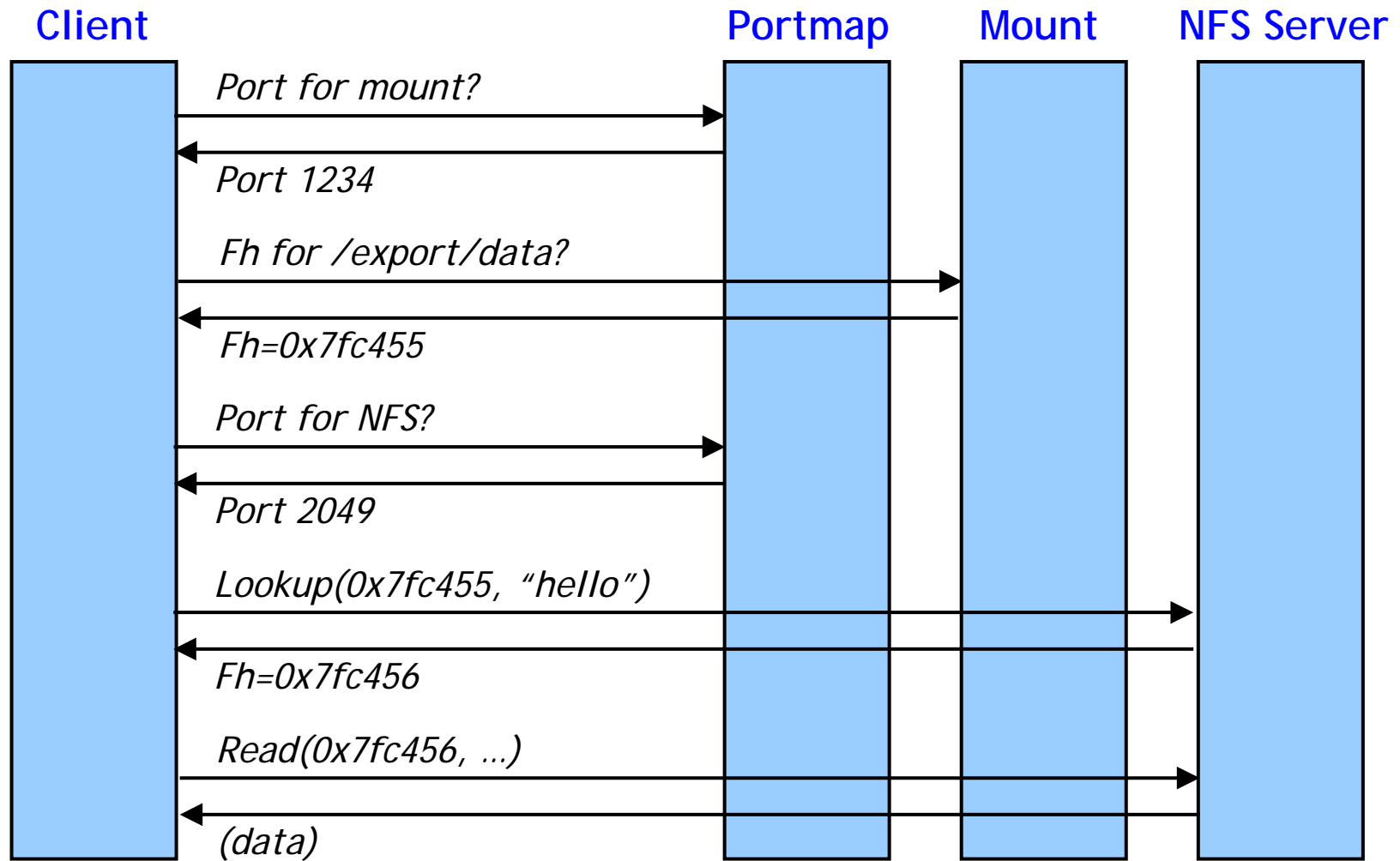
- Used to obtain an initial filehandle for the root of an exported filesystem.

- **Why have a separate protocol?**

- Easy to check access to exported filesystems.
 - The NFS server is implemented within the kernel.
 - A kernel process cannot easily access user-level services and files for checking access to exported filesystems.
 - » NIS, NIS+, LDAP, etc.
 - The NFS MOUNT protocol is implemented in a user-level daemon process.
- Allows for alternative protocols.

Operating System

NFS MOUNT Protocol (2)



Operating System

NFS Implementation (1)

■ File Attributes

POSIX	NFS v2	FAT	NTFS	Description
	type			File type
st_mode	mode	readonly	ACL	Permission bits
st_ino	fileid			File ID
st_dev	fsid			Filesystem ID
st_rdev	rdev			Device ID
st_nlink	nlink		names	Number of links
st_uid	uid		owner	Owner ID
st_gid	gid		group	Group ID
st_size	size	size	size	File size
st_atime	atime			Last access time
st_mtime	mtime	mtime	mtime	Last modification time
st_ctime	ctime			Metadata modification time
st_blksize	blocks			Number of file blocks
st_blocks	blocksize			File block size

Operating System

NFS Implementation (2)

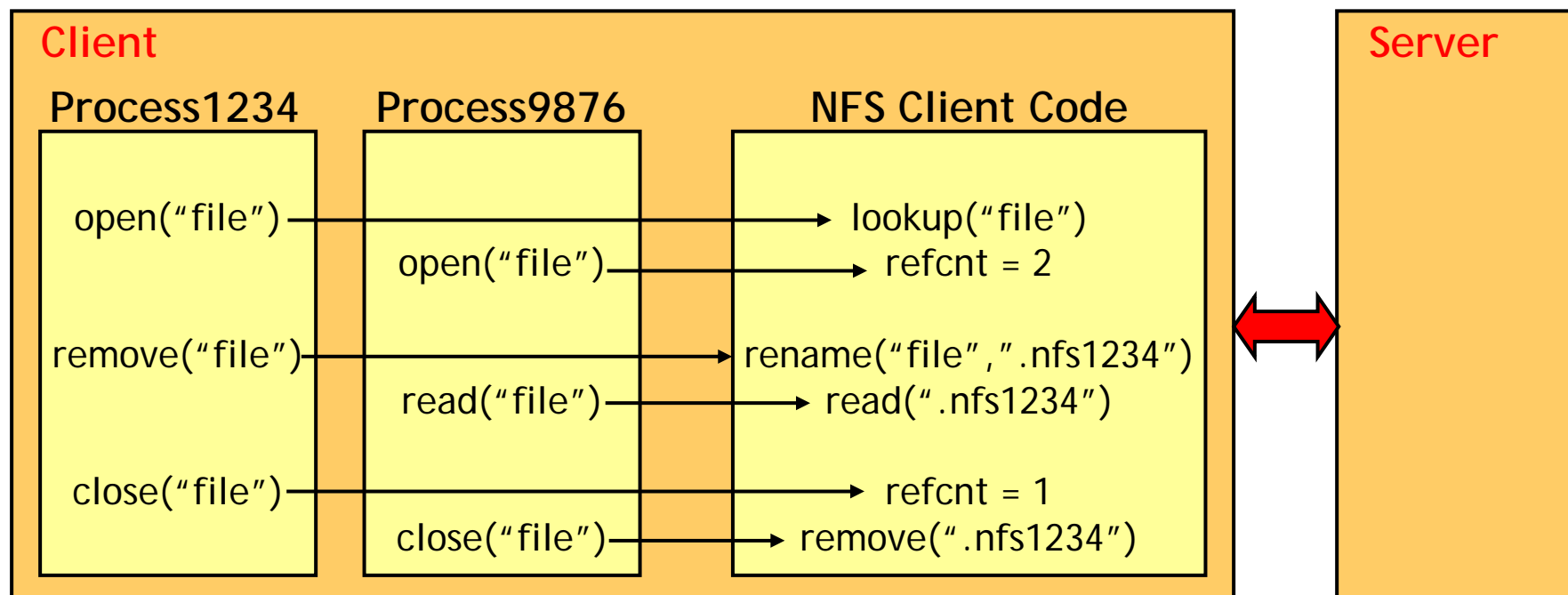
■ Last Close Problem

- UNIX allows removal of open files.
 - The other process must continue to have access to the file data until it closes the file.
 - The filesystem must maintain a count of all processes that currently have the unnamed file open and remove the file when the last process closes the file.
 - Many programs use this for temporary files. (csh, sendmail, ...)
- An NFS server has no knowledge of file opens.
 - Due to the statelessness of the NFS server.
- If the file is open on REMOVE, issues a RENAME request to change the name to a hidden name.
 - .nfsxxxx in Solaris where xxxx is the process's ID.

NFS Implementation (3)

▪ Last Close Problem (cont'd)

- The hidden files are deleted later by the server when it has determined that they are no longer in use.



Operating System

NFS Implementation (4)

■ Access Control and Authentication

- User's identity and access rights must be checked by the server on each request.
 - Every client request is accompanied by the uid and gid.
- Every server and client pair must have the same mapping from username to uid and from group name to gid.
 - This implies a flat uid, gid space over a while local network.
 - NIS(YP)/NIS+: another RPC-based service to provide a simple, replicated database lookup service.
- The server maps user root (uid 0) to user nobody (uid - 2) before checking access permission.

Operating System

NFS Implementation (5)

■ Checking File Access Permissions

- A UNIX process can open a file, then change the permissions.
 - The process is still granted access to the file, but the others are not allowed to touch it.
 - This is because the access permission is only checked when the file is opened.
- In NFS, the permission is checked on every NFS call.
 - The server's permission-checking algorithm has to allow the owner of a file access for READ or WRITE calls regardless of the permission setting.

Operating System

NFS Implementation (6)

■ Crossing of Server Mountpoints

- What happens if the LOOKUP request refers to a mountpoint on the server?
 - UNIX clients require each filesystem ID to have a corresponding mounted filesystem.
 - The server in the middle cannot reliably distinguish the remote NFS server filehandles from the filehandles for its own exported filesystem.
 - The risk of a namespace cycle exists.
- NFS servers prevent NFS client LOOKUPS from crossing server mountpoints.
 - The client must mount each of the filesystems individually so that the mountpoint crossing takes place on the client.

Operating System

NFS Implementation (7)

■ Server Caching

- NFS servers use the cache just as it is used for other file accesses.
- Data-modifying operations in the NFS protocol must be synchronous.
 - Requests which modify the filesystem must flush all modified data to disk before returning from the call.
- The requirement for synchronous operations can have a significant impact on the performance.
 - Some implementations offer asynchronous server operations as an option to improve performance.
 - Some implementations relax the requirement only for metadata update. (except the file access time – atime)
 - The NFS v3 introduces safe asynchronous writes on the server. (using the COMMIT request)

Operating System

NFS Implementation (8)

■ Client Caching

- The NFS client caches the results of previous operations.
 - File data (READ), Lookup results (LOOKUP), Directories (REaddir), Attributes, Symbolic links (READLINK), Filesystem information (STATFS)
- Clients are expected to make a best-effort attempt to keep their cached data in sync with the server.
- The most commonly implemented caching scheme uses a cache time and a modification time.
 - If the data is referenced within the cache time, then the cache data are used.
 - Otherwise, the client will contact the server and verify that the cached modification time has not changed.

Operating System

NFS Implementation (9)

▪ Client Caching (cont'd)

- The cache time is a compromise that trades off cache consistency against server and network loading.
 - In Solaris, the min/max cache time for file is 3sec/30sec.
 - For directory, the min/max cache time is 30sec/60sec.
- New files created on a client may not be visible elsewhere for 30 seconds.
- It is indeterminate whether writes to a file at one site are visible to other sites that have file open for reading.

Operating System

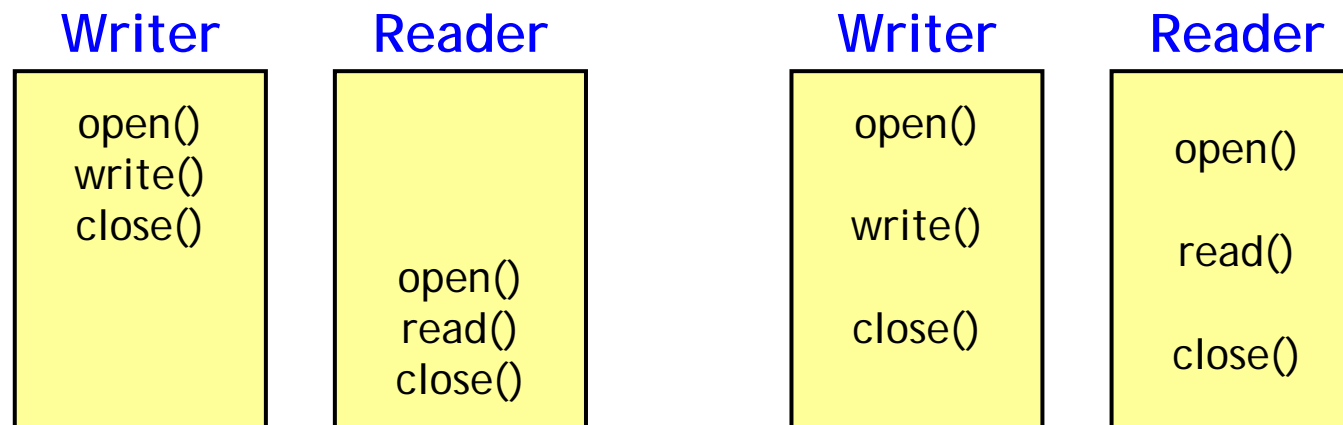
NFS Implementation (10)

- **Close-to-Open Consistency**
 - With earlier versions of NFS, updates made on one NFS client might not show up on another client for a few seconds.
 - NFS clients may implement close-to-open consistency.
 - If you update a file and then close it on one NFS client, any process that will open it on another client will see your updates.
 - How?
 - When a file is closed by an application, the client flushes any cached changes to the server.
 - When a file is opened, the client ignores any cache time remaining and makes an explicit GETATTR call to the server to check the file modification time.

NFS Implementation (11)

■ Close-to-Open Consistency (cont'd)

- Change did not require any modification to the NFS protocol.
- It cannot help where the reader's open does not follow the writer's close; the reader may not see the data written by the writer.



Operating System

NFS Implementation (12)

■ File Locking

- The NFS does not support remote file locking.
- NFS Lock Manager Protocol

■ Time Skew

- Time skew between two clients or a client and a server can cause time associated with a file to be inconsistent.
 - e.g. ranlib and ld
- This is a potential problem for any program that compares system time to file modification time.
- Need to use a time synchronization protocol.

Discussion (1)

■ Access Transparency

- Programs written to operate on local files are able to access remote files without modification.
- Very good.

■ Location Transparency

- Client programs should see a uniform file name space.
- NFS does not enforce a single network-wide file name space.
- However, a uniform name space can be established with appropriate configuration tables in each client.

Discussion (2)

▪ Mobility Transparency

- Neither client programs nor system tables in client nodes need to be changed when files are moved.
- In NFS, file systems may be moved between servers.
- However, the remote mount tables in each client must be updated separately to enable the clients to access the filesystem in its new location.

Discussion (3)

■ Performance Transparency

- Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.
- Both the client and server employ caching to achieve satisfactory performance.
- For clients, the maintenance of cache coherence is somewhat complex, because several clients may be using and updating the same file.

Discussion (4)

■ Scaling Transparency

- The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.
- Scalability of the NFS is limited.
 - Due to the lack of replication.
- When the performance limit is reached, additional servers can be installed.
 - Filesystems must be reallocated between them.
- The effectiveness is limited by the existence of “hot spot” files.

Discussion (5)

▪ Fault Tolerance

- The failure modes observed by clients when accessing remote files are similar to those for local file access.
 - NFS server is stateless.
 - Most file access operations are idempotent.
- When a server fails, the service is suspended until the server is restarted.
 - Once it has restarted, user-level client processes proceed from the point at which the service was interrupted, unaware of the failure.
- The failure of a client or a user-level process in a client has no effect on any server that it may be using.

Operating System

Discussion (6)

■ File Replication

- Read-only file stores can be replicated on several NFS servers.
- However, NFS does not support file replication with updates.

■ Concurrency

- NFS cannot control concurrent updates to files.

■ Hardware and OS Heterogeneity

- Very good.

Discussion (7)

■ Consistency

- Provides a close approximation to one-copy semantics and meets the needs of the vast majority of applications.
- The use of file sharing via NFS for communication or close coordination between processes on different machines cannot be recommended.
- Supports the close-to-open consistency in later versions.

■ Security

- The need for security in NFS only emerged with the connection of most intranets to the Internet.
- Kerberized NFS, Secure RPC, ...