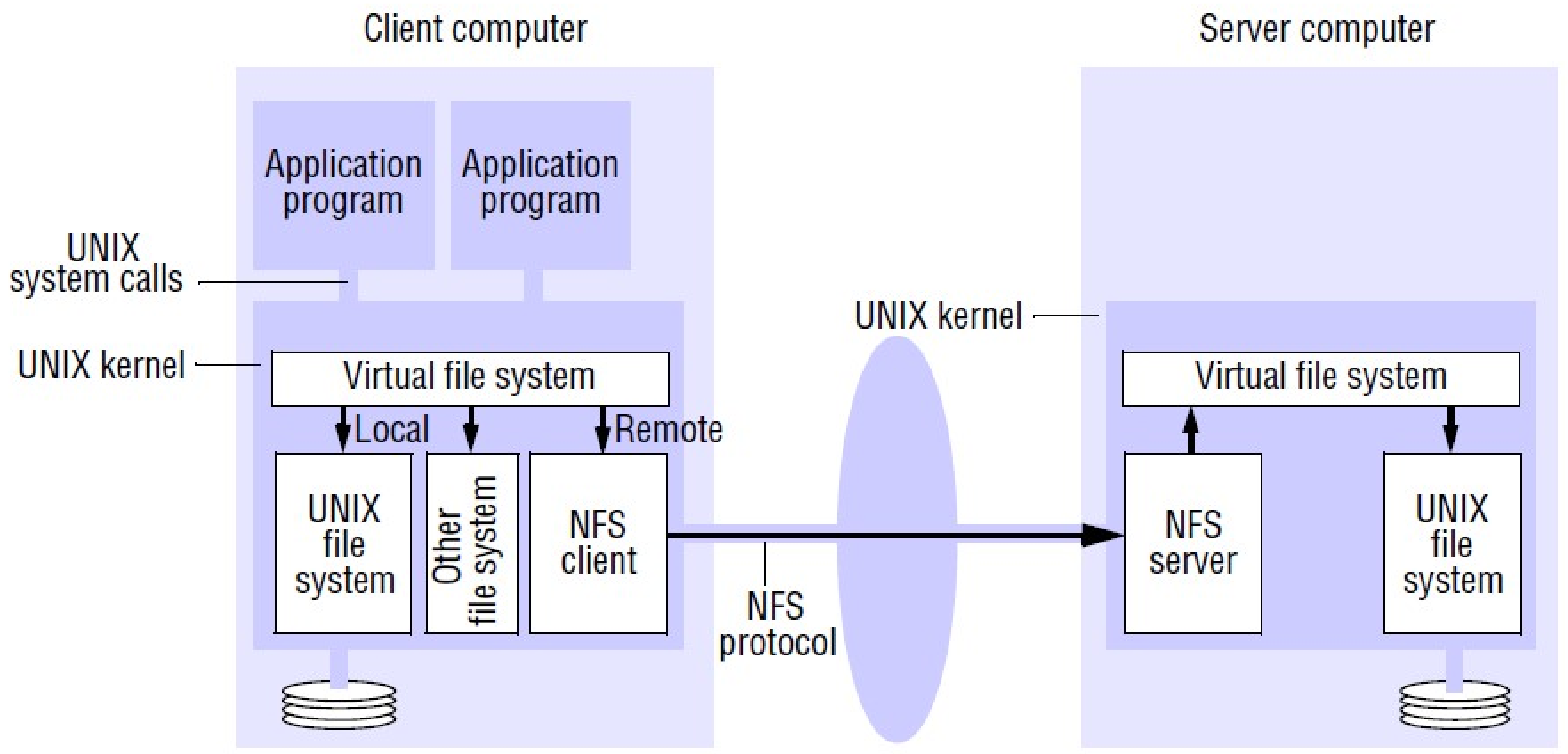
Case study on Network File System

Mrs. Vishakha Shelke

Computer Engineering Dept, UCoE

Figure 12.8 NFS architecture



Sun NFS

- All implementations of NFS support the NFS protocol – a set of remote procedure calls that provide the means for clients to perform operations on a remote file store.
- The NFS protocol is operating system–independent but was originally developed for use in networks of UNIX systems

Sun NFS

- The NFS server module resides in the kernel on each computer that acts as an NFS server.
- Requests referring to files in a remote file system are translated by the client module to NFS protocol operations and then passed to the NFS server module at the computer holding the relevant file system.
- The NFS client and server modules communicate using remote procedure calls.

Sun NFS

- Sun’s RPC system, studied in Module 2 ,was developed for use in NFS. It can be configured to use either UDP or TCP, and the NFS protocol is compatible with both.
- A port mapper service is included to enable clients to bind to services in a given host by name.
- The RPC interface to the NFS server is open: any process can send requests to an NFS server;
- If the requests are valid and they include valid user credentials, they will be acted upon.
- The submission of signed user credentials can be required as an optional security feature, as can the encryption of data for privacy and integrity.

Virtual file system

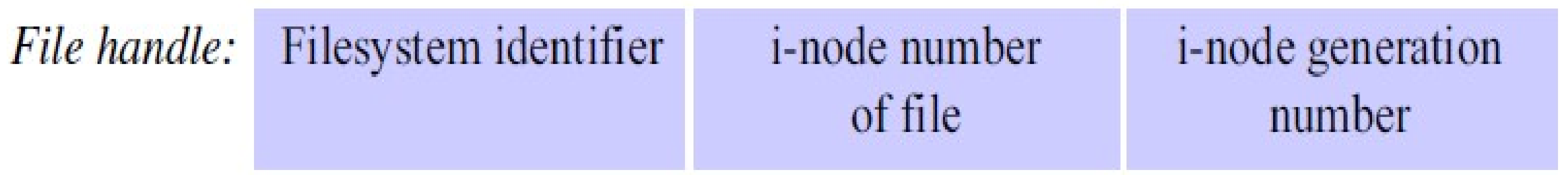
- **Figure 12.8 makes it clear that NFS provides access transparency:** user programs can issue file operations for local or remote files without distinction. Other distributed file systems may be present that support UNIX system calls, and if so, they could be integrated in the same way.

- The integration is achieved by a virtual file system (VFS) module, which has been added to the UNIX kernel to distinguish between local and remote files and to translate between the UNIX independent file identifiers used by NFS and the internal file identifiers normally used in UNIX and other file systems.
- In addition VFS keeps track of the file systems that are currently available both locally and remotely, and it passes each request to the appropriate local system module

File Identifier

- The file identifiers used in NFS are called *file handles*
- *A file handle is opaque to* clients and contains whatever information the server needs to distinguish an individual file.
- In UNIX implementations of NFS, the file handle is derived from the file's *i-node number by adding two extra fields as follows*

File Handle



➢The *file system identifier field is a unique number that is allocated to* each file system when it is created.

➢the i-node number of a UNIX file is a number that serves to identify and locate the file within the file system in which the file is stored

➢The *i-node generation number is needed because in the* conventional UNIX file system
 i-node numbers are reused after a file is removed.

➢In the VFS extensions to the UNIX file system, a generation number is stored with each file and is incremented each time the i-node number is reused

The virtual file system layer

- The virtual file system layer has one VFS structure for each mounted file system and one *v-node per open file.*
- *A VFS structure relates a remote file system to the local* directory on which it is mounted.
- The v-node contains an indicator to show whether a file is local or remote. If the file is local, the v-node contains a reference to the index of

the local file ,If the file is remote, it contains the file handle of the remote file.

Client integration

- **The NFS client module plays the role described for the client**
- module in our architectural model, supplying an interface suitable for use by conventional application programs.
- But unlike our model client module, it emulates the

  semantics of the standard UNIX file system primitives precisely and is integrated with the UNIX kernel.

  It is integrated with the kernel and not supplied as a library for loading into client processes so that:

Client Integration

- user programs can access files via UNIX system calls without recompilation or reloading;
- a single client module serves all of the user-level processes, with a shared cache of recently used blocks
- the encryption key used to authenticate user IDs passed to the server can be retained in the kernel, preventing impersonation by user-level clients.

Client Integration

- The NFS client module cooperates with the virtual file system in each client machine.
- It operates in a similar manner to the conventional UNIX file system, transferring blocks of files to and from the server and caching the blocks in the local memory whenever possible.
- It shares the same buffer cache that is used by the local input-output system

Access control and authentication

- Unlike the conventional UNIX file system, the NFS server is stateless and does not keep files open on behalf of its clients. So the server must check the user’s identity against the file’s access permission attributes to see whether the user is permitted to access the file in the manner requested.
- An NFS server provides a conventional RPC interface at a well-known port on each host

NFS server interface

- The NFS file access operations *read, write, getattr and setattr are almost identical* to the *Read, Write, GetAttributes and SetAttributes operations defined for flat file* service model
- 222 The *lookup operation and most of the other directory*
- operations defined in Figure 12.9 are similar to those in our directory service model
- (Figure 12.7).

NFS server interface

- The file and directory operations are integrated in a single service; the creation and insertion of file names in directories is performed by a single *create operation, which btakes* the text name of the new file and the file handle for the target directory as arguments.
- The other NFS operations on directories are *create, remove, rename, link, symlink, readlink, mkdir, rmdir, readdir and statfs.*
- *They resemble their UNIX* counterparts with the exception of *readdir, which provides a representation independent*

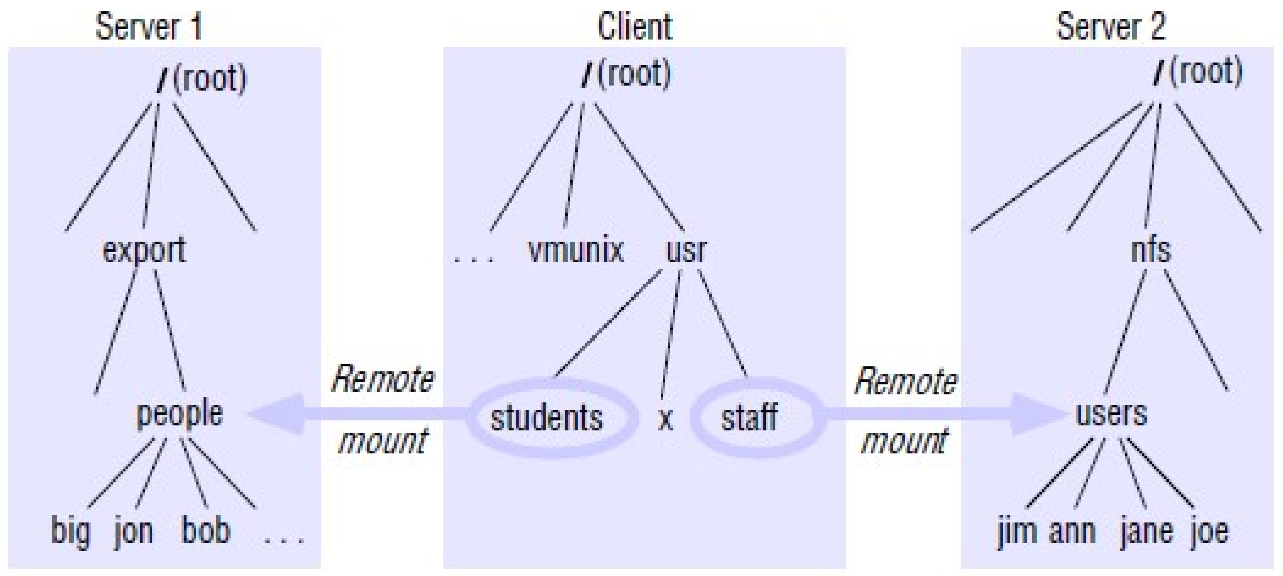
  method for reading the contents of directories, and *statfs, which gives the* status information on remote file systems.

Mount service

- The mounting of subtrees of remote file systems by clients is supported by a separate *mount service process that runs at user level on each NFS server computer.*
- On each server, there is a file with a well-known name *(/etc/exports) containing the* names of local file systems that are available for remote mounting.
- An access list is associated with each file system name indicating which hosts are permitted to mount the file system.
- Clients use a modified version of the UNIX *mount command to request mounting* of a remote file system, specifying the remote host's name, the pathname of a directory in the remote file system and the local name with which it is to be mounted.
- The remote directory may be any subtree of the required remote file system, enabling clients to mount any part of the remote file system.

Mount service

Figure 12.10 Local and remote filesystems accessible on an NFS client



Note: The file system mounted at */usr/students* in the client is actually the subtree located at */export/people* in Server 1; the filesystem mounted at */usr/staff* in the client is actually the subtree located at /nfs/users in Server 2.

Mount Service

- The modified *mount command communicates* with the mount service process on the remote host using a *mount protocol.*
- *This is an* RPC protocol and includes an operation that takes a directory pathname and returns the file handle of the specified directory if the client has access permission for the relevant filesystem.
- The location (IP address and port number) of the server and the file handle for the remote directory are passed on to the VFS layer and the NFS client.

Pathname translation

- In NFS, pathnames cannot be translated at a server, because the name may cross a'mount point' at the client – directories holding different parts of a multi-part name may reside in file systems at different servers. So pathnames are parsed, and their translation is performed in an iterative manner by the client.
- Each part of a name that refers to a remote-mounted directory is translated to a file handle using a separate *lookup request* to the remote server.
- The *lookup operation looks for a single part of a pathname in a given directory and* returns the corresponding file handle and file attributes.

Pathname translation

- The file handle returned in the previous step is used as a parameter in the next *lookup step*.
- *Since file handles are* opaque to NFS client code, the virtual file system is responsible for resolving file handles to a local or a remote directory and performing the necessary indirection when it references a local mount point.
- Caching of the results of each step in pathname translations alleviates the apparent inefficiency of this process, taking advantage of locality of reference to files and directories; users and programs typically access files in only one or a small number of directories

Automounter

- The automounter was added to the UNIX implementation of NFS in order to mount a remote directory dynamically whenever an ‘empty’ mount point is referenced by a client.
- The automounter maintains a table of mount points (pathnames) with a reference to one or more NFS servers listed against each. It behaves like a local NFS server at the client machine.
- When the NFS client module attempts to resolve a pathname that includes one of these mount points, it passes to the local automounter a *lookup() request* that locates the required file system in its table and sends a ‘probe’ request to each server listed.
- The file system on the first server to respond is then mounted at the client using the normal mount service.

Server caching

- NFS servers use the cache at the server machine just as it is used for other file accesses.
- Caching in both the client and the server computer are indispensable features of NFS implementations in order to achieve adequate performance.

Client caching

- The NFS client module caches the results of *read, write, getattr, lookup and readdir operations in order to reduce the number of requests transmitted to* servers.

Other design goals of NFS

- *Access transparency*
- *Location transparency*
- *Mobility transparency*
- *Scalability*
- *File replication*
- *Hardware and operating system heterogeneity*
- *Fault tolerance*
- *Consistency*
- *Security*
- *Efficiency*