Distributed Systems: Concepts and Design

Edition 5

By George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair

Chapter 12 Exercise Solutions

12.1 File systems are responsible for the organization, storage, retrieval, naming, sharing and protection of files. What are the requirements of distributed file systems?

12.1 Ans.

Many of the requirements and potential pitfalls in the design of distributed services were first observed in the early development of distributed file systems. Initially, they offered access transparency and location transparency; performance, scalability, concurrency control, fault tolerance and security requirements emerged and were met in subsequent phases of development. These are some of the requirements of distributed file systems.

12.2 Give one important goal of Sun Microsystems' Network File System (NFS).

12.2 Ans.

An important goal of NFS is to achieve a high level of support for hardware and operating system heterogeneity. The design is operating-system independent: client and server implementations exist for almost all operating systems and platforms, including all versions of Windows, Mac OS, Linux and every other version of UNIX.

12.3 Write a procedure PathLookup(Pathname, Dir) $\rightarrow$ UFID that implements Lookup for UNIX-like pathnames based on our model directory service.

12.3 Ans.

Left to the reader.

12.4 What is a file group? What are the properties of the file group identifier?

12.4 Ans.

A file group is a collection of files located on a given server. A server may hold several file groups, and groups can be moved between servers, but a file cannot change the group to which it belongs. File group identifiers must be unique throughout a distributed system.

12.5 To what extent does Sun NFS deviate from one-copy file update semantics? Construct a scenario in which two user-level processes sharing a file would operate correctly in a single UNIX host but would observe inconsistencies when running in different hosts.

12.5 Ans.

After a *write* operation, the corresponding data cached in clients other than the one performing the *write* is invalid (since it does not reflect the current contents of the file on the NFS server), but the other clients will not discover the discrepancy and may use the stale data for a period of up to 3 seconds (the time for which cached blocks are assumed to be fresh). For directories, the corresponding period is 30 seconds, but the consequences are less serious because the only operations on directories are to insert and delete file names.

Scenario: any programs that depend on the use of file data for synchronization would have problems. For example, program A checks and sets two locks in a set of lock bits stored at the beginning of a file, protecting records within the file. Then program A updates the two locked records. One second later program B reads the same locks from its cache, finds them unset, sets them and updates the same records. The resulting values of the two records are undefined.

12.6 Sun NFS aims to support heterogeneous distributed systems by the provision of an operating system-independent file service. What are the key decisions that the implementer of an NFS server for an operating system other than UNIX would have to take? What constraints should an underlying filing system obey to be suitable for the implementation of NFS servers?

12.6 Ans.

The Virtual file system interface provides an operating-system independent interface to UNIX and other file systems. The implementor of an NFS server for a non-Unix operating system must decide on a representation for file handles. The last 64 bits of a file handle must uniquely define a file within a file system. The Unix representation is defined (as shown on page ??) but it is not defined for other operating systems. If the operating system does not provide a means to identify files in less than 64 bits, then the server would have to generate identifiers in response to *lookup*, *create* and *mkdir* operations and maintain a table of identifiers against file names.

Any filing system that is used to support an NFS server must provide:

- efficient block-level access to files;
- file attributes must include write timestamps to maintain consistency of client caches;
- other attributes are desirable, such owner identity and access permission bits.

12.7 Which services must be included to enable clients to bind to services in a given host by name?

12.7 Ans.

Port mapper service

12.8 Give the uses of the following NFS server operations: (i) *read (fh, offset, count)* (ii) *write (fh, offset, count, data)* (iii) *lookup (dirfh, name)*.

12.8 Ans.

i. *read (fh, offset, count)*: Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.

ii. *write (fh, offset, count, data)*: Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.

iii. *lookup (dirfh, name)*: Returns file handle and attributes for the file *name* in the directory *dirfh*.

12.9 Explain why the RPC interface to early implementations of NFS is potentially insecure. The security loophole has been closed in NFS 3 by the use of encryption. How is the encryption key kept secret? Is the security of the key adequate?

12.9 Ans.

The user id for the client process was passed in the RPCs to the server in unencrypted form. Any program could simulate the NFS client module and transmit RPC calls to an NFS server with the user id of any user, thus gaining unauthorized access to their files. DES encryption is used in NFS version 3. The encryption key is established at *mount* time. The mount protocol is therefore a potential target for a security attack. Any workstation could simulate the mount protocol, and once a target filesystem has been mounted, could impersonate any user using the encryption agreed at mount time..

12.10 After the timeout of an RPC call to access a file on a hard-mounted file system the NFS client module does not return control to the user-level process that originated the call. Why?

12.10 Ans.

Many Unix programs (tools and applications) are not designed to detect and recover form error conditions returned by file operations. It was considered preferable to avoid error conditions wherever possible, even at the cost of suspending programs indefinitely.

12.11 How does server caching help to achieve the adequate performance of NFS implementations?

12.11 Ans.

NFS servers use the cache at the server machine just as it is used for other file accesses. The use of the server's cache to hold recently read disk blocks does not raise any consistency problems; but when a server performs write operations, extra measures are needed to ensure that clients can be confident that the results of write operations are persistent, even when server crashes occur.

12.12 How many lookup calls are needed to resolve a 5-part pathname (for example, /usr/users/jim/code/xyz.c) for a file that is stored on an NFS server? What is the reason for performing the translation step-by-step?

12.12 Ans.

Five *lookups*, one for each part of the name.

Here are several reasons why pathnames are looked up one part at a time, only the first of those listed is mentioned in the book (on p. 229):

i) pathnames may cross mount points;

ii) pathnames may contain symbolic links;

iii) pathnames may contain '..';

iv) the syntax of pathnames could be client-specific.

Case (i) requires reference to the client mount tables, and a change in the server to which subsequent *lookups* are dispatched. Case (ii) cannot be determined by the client. Case (iii) requires a check in the client against the 'pseudo-root' of the client process making the request to make sure that the path doesn't go above the pseudo-root. Case (iv) requires parsing of the name at the client (not in itself a bar to multi-part lookups, since the parsed name could be passed to the server, but the NFSD protocol doesn't provide for that).

12.13 Kerberos is an authentication system used to secure intranet servers against unauthorized access and imposter attacks. In what way is the 'Kerberization' of NFS carried out?

12.13 Ans.

When NFS is used in a 'Kerberized' environment it should accept requests only from clients whose identity can be shown to have been authenticated by Kerberos.

12.14 What are the design characteristics of AFS? Name the two software components, which exist as UNIX processes, with which ANS is implemented.

12.14 Ans.

Design characteristics: Whole-file serving and Whole-file caching.

Two software components: Vice and Venus.

12.15 Explain the uses of the following components of the Vice service interface:

(i) *SetLock(fid, mode)* (ii) *RemoveCallback(fid)*

12.15 Ans.

SetLock(fid, mode): Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.

RemoveCallback(fid): Informs the server that a Venus process has flushed a file from its cache.

12.16 How does AFS deal with the risk that callback messages may be lost?

12.16 Ans.

Callbacks are renewed by Venus before an *open* if a time T has elapsed since the file was cached, without communication from the server. T is of the order of a few minutes.

12.17 Which features of the AFS design make it more scalable than NFS? What are the limits on its scalability, assuming that servers can be added as required? Which recent developments offer greater scalbility?

12.17 Ans.

The load of RPC calls on AFS servers is much less than NFS servers for the same client workload. This is achieved by the elimination of all remote calls except those associated with *open* and *close* operations, and the use of the *callback* mechanism to maintain the consistency of client caches (compared to the use of *getattributes* calls by the clients in NFS). The scalability of AFS is limited by the performance of the single server that holds the most-frequently accessed file volume (e.g. the volume containing */etc/passwd*, */etc/hosts*, or some similar system file). Since read-write files cannot be replicated in AFS, there is no way to distribute the load of access to frequently-used files.

Designs such as xFS and Frangipani offer greater scalability by separating the management and metadata operations from the data handling, and they reduce network traffic by locating files based on usage patterns.