CMPSCI 677 Distributed & Operating Systems **Spring 2022**

Lecture 22: April 20

Lecturer: Prashant Shenoy *Scribe:*

22.1 File System Basics

22.1.1 File

A file is a container of data in text format, binary format etc. which is stored on a disk so that the user can re-visit it at a later point in time. In UNIX, a file is an uninterpreted sequence of bytes which implies that the file system is unaware of the contents/type of the file. Other operating systems like Windows and Mac knows the file types (This information can be useful to open a file in the right application).

22.1.2 File System

- File system abstracts and provides a logical view of data (a hierarchy of files and folders) and storage functions.
- It helps us to create, modify, organize and delete files and takes care of how to map them to the underlying storage device.
- It provides a user-friendly interface so that the user need not deal with the low-level interfaces exported by the disk.
- It allows us to share the files among other users by giving permissions and also allows us to protect the files.

22.1.3 UNIX File System Review

- In UNIX, the files structure can be viewed as a directed acyclic graph. Note that this looks like a tree structure but can contain soft links pointing from one directory to other which makes it a DAG. Each directory entry for each file contains the file name, inode number (metadata for the file), major device number and minor device number. All inodes are stored at a particular location on the disk called super block. To access the file, the file system needs to first get this metadata to know where the file is located in the actual disk (aka block locations of the file).
- An inode structure consists of the fields like mode, Owner ID, group id, Dir file, protection bits, last access time, size, reference count, address[0]...address[14] etc. The addresses stores the pointers to the data blocks. The first 12 are the direct blocks which stores the pointers to the data blocks (see figure 22.1), the 13th address stores the pointers to the location which in turn stores the pointers to the direct data blocks (one level of indirection). The 14th address follows two levels of indirection which stores the pointers to one level indirection blocks. So the hierarchy grows as the size of the file grows but we have an upper limit of the size of the file that can be stored on this file system because we only have a certain number of pointers in addresses (In this case from 0 till 14).

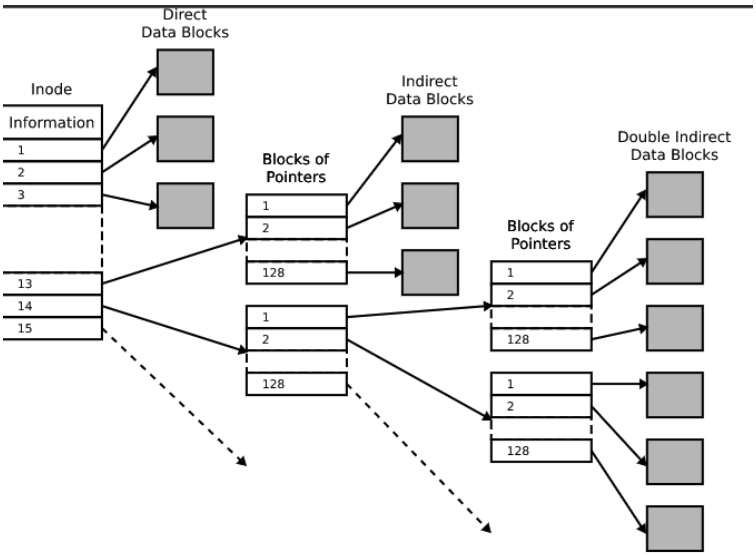

Figure 22.1: Inode structure

22.2 Distributed File Systems (DFS)

If files on a different machine can be accessed, it is a distributed file system. Another way to think about DFS is that the servers store different files on different servers, and all the servers collectively form your file system.

22.2.1 File server

A machine that stores all the files.

22.2.2 File service

The interface that the machine exposes for other machines to access the files on this machine. For example NFS uses RPCs to send read/write requests to a remote file system. There are two types of file services as shown in figure 22.2.

- Remote access model: The client requests are sent to the server and the server sends back the results after doing the work requested by the client.
- Upload/download model: When the client performs a request to the server, entire file is sent as a copy to the client, and subsequent access are made to the local copy. To maintain consistency, the client eventually sends back the changed file to the server.

Note: As the files are directly updated on the server, there is consistency in the remote access model, but each operation is an RPC call which makes it slow. In upload/download model, there is a period of time in which the file on the server is out of date. Having said that, upload/download model gives better performance as the operations are taking place on the local machine and very less calls are made to the remote machine.

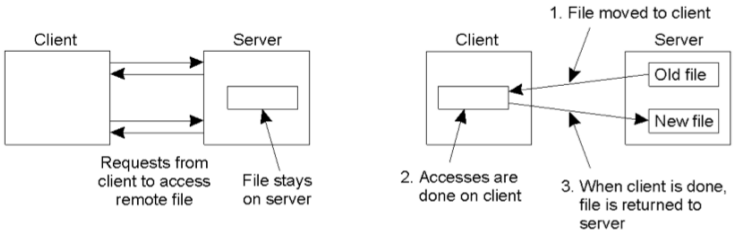

Figure 22.2: Remote access model(left), Upload/download model(right).

22.2.3 Server Type

There are two types of server and one would need to make the choice from one of them when building any distributed file system.

Stateless server: No information about clients is kept at the server.

Stateful server: Server maintains information about the client accesses. It is less tolerant to failures because the state is lost when a server crashes. There is slight performance benefit here due to the compact request messages (Clients do not need to send the information like permissions every time the request is being made). Consistency and idempotency are easier to achieve.

Note: An Idempotent server executes as if it has performed the request only once regardless of how many times same request was received.

22.2.4 Network File System (NFS)

NFS is a layer on top of an existing file system that allows to share the file system over a network. NFS is implemented using virtual file system layer supported by the underlying operating system. Virtual file system layer can be seen as a forwarding layer that looks at where is the file stored and invoke that file system(local file system for local files and possibly NFS for remote files).

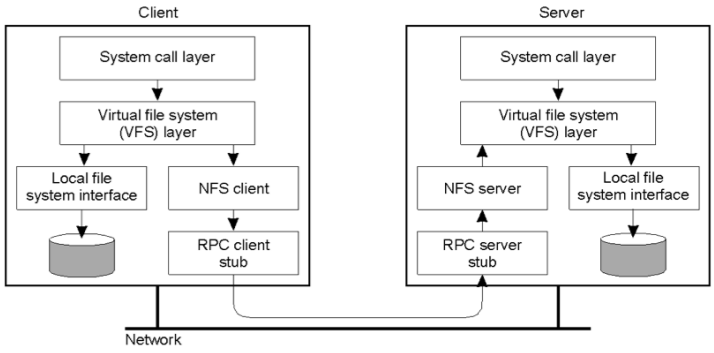

Figure 22.3: Network File System (NFS)

Note: Till version 3, NFS used stateless server protocol but from version 4, it uses stateful server protocol. So it now supports open call to a remote file.

In figure 22.4, we can observe that in version 3 of NFS, individual LOOKUP and READ RPC calls were

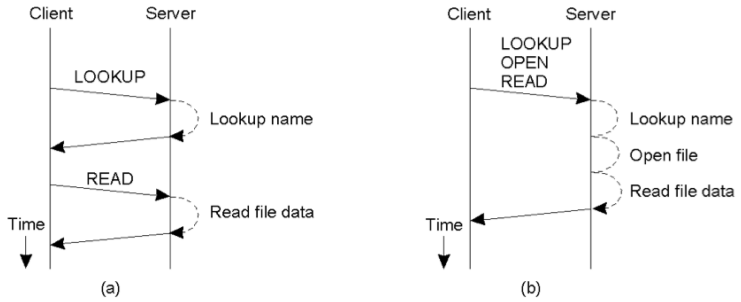

Figure 22.4: Difference in communication in NFS version 3 and version4.

needed whereas in version 4 of NFS, we can perform a batch RPC request.

22.2.5 Mount protocol

Mount protocol is a way how a NFS client gets access to a remote file system. Certain directories can be mapped from remote file system to the local file system in order to get access.

22.2.6 Crossing mount points

Crossing mount points is mounting nested directories from multiple servers.

22.2.7 Automounting

Automounting is also known as mounting on demand. The mappings get established but the mounting only happens when the user tries to access those directories. And if there is an idle time, it unmounts. This way we can reduce the amount of kernel resources used.

22.2.8 File attributes

There are specific attributes (like TYPE, SIZE, CHANGE, FSID) that a file system must to support to be compatible with NFS. There are other attributes (like OWNER of a file) which are not mandatory to be compatible with NFS but are recommended.

22.2.9 Semantics of file sharing

- In UNIX semantics, every operation on a file is instantly visible to all the other processes using the same file.
- In session semantics, no changes are visible to other processes until the file is closed.
- Immutable files cannot be mutated. A new version of the file needs to be created if we need any changes.
- In transaction semantics, all changes occur atomically.

Note: NFS follows semantics in between UNIX and session. It caches the file and periodically flushes the changes to the server. If one process writes to a file, the other process might have a different or outdated version of the file for a period of time. NFS uses local caches for performance reasons which leads to this weak consistency.

22.2.10 File locking in NFS

- Version 3 of NFS used stateless server protocol. One of the uses of having a state is file locking. Version 4 of NFS uses stateful server protocol, so applications can now use locks to ensure consistency. File locking can be done in different ways like locking the entire file, locking a specific range of bytes in a file etc.
- In share reservations file locking, we have the notion of a denial state where if an application has a write denial state to a file, it cannot write to the file but it can read the file.