

Lecture 20: April 22

*Lecturer: Prashant Shenoy**Scribe: Anirudha Desai(2017), Aggrey Muhebwa(2019)*

20.1 Announcements

The final exam is on Friday next week, 5/3/2019. It will be a take home exam and you can use any material on the internet, but you have to cite the sources that you'll use. The solutions can either be typeset or handwritten and then scanned. The exam is 24 hours and the solutions will be uploaded to gradescope.

20.2 Distributed File Systems

The class was primarily about stand-alone(UNIX) file systems.

There are many file systems. Linux can read many file systems. But windows on the other hand has its own proprietary file systems and cannot read linux file system. A FS manifests as files and directories.

A file is a named collection of logically related data. OS has no way to interpret the files. It is the application's job to interpret the file. A file system :

1. Provides a logical view of data and storage functions
2. User-friendly interface
3. Provides facility to create , modify , organize, and delete files.
4. Provides sharing among users in a controlled manner - Permissions can be granted on files to specific users.
5. Provides protection

A directory is an index of files contained in a folder.

20.2.1 UNIX File System review

A user file is a linear array of bytes. A file structure is a directed acyclic graph as shown in fig 20.1.

But the operating system does not interpret English. It only understands numbers. Every directory/file has an associated number in its metadata. This number is the *inode* number. The OS can understand this inode number. All inodes are stored at a special location on disk [super block]. Within a file structure, directories may not be shared, but files may be. Shared in this context means links. 2 files can be linked to the same inode number. There can be a *hard link* or a *soft link* (shortcuts to a file). But directories cannot have links because we can end up with a cycle when a child directory has a hard link back to its parent directory.

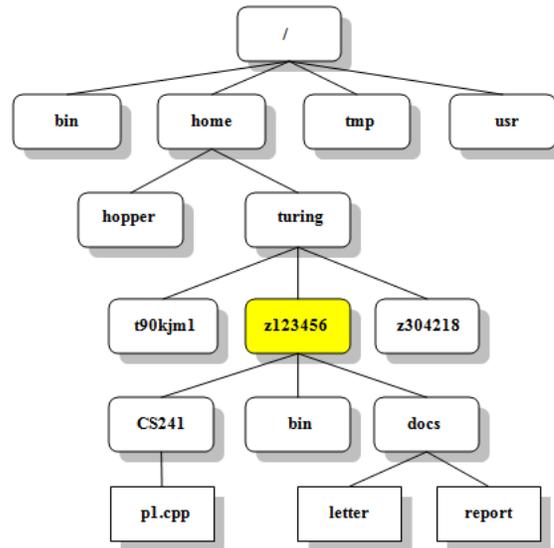


Figure 20.1: Typical file structure

Inodes store file attributes and a multi-level index that has a list of disk block locations for the file. As shown in figure 20.2, there are multiple levels of indirection used to point to different blocks of a file. In this multi-level index format, we can technically store $11 * 1024 * 1024 \approx 4gb$ file. It should be noted that this is one way of implementing a FS. There are many other ways.

20.2.2 Inode structure

The following fields comprise the inode number : mode, owne_id, dir_file, protection bits, last access time, last write time, last inode time, size, ref_cnt and Address[0]...Address[14] with mult-level index.

20.2.3 Distributed File Systems (DFS)

So far, the class was about single FS. A DFS is more complicated. A Distributed File System consists of two major parts;

1. File service which is a specification of what the file system offers, e.g client primitives, APIs, etc
2. File server which is a process that implements the file service. A single Machine can have several servers (UNIX, DOS) running on it

20.2.3.1 File Service

2 types : The slides could be referred for diagrammatic representation.

1. Remote access model : Work done at server. It is a stateless server. Server may be a bottleneck.

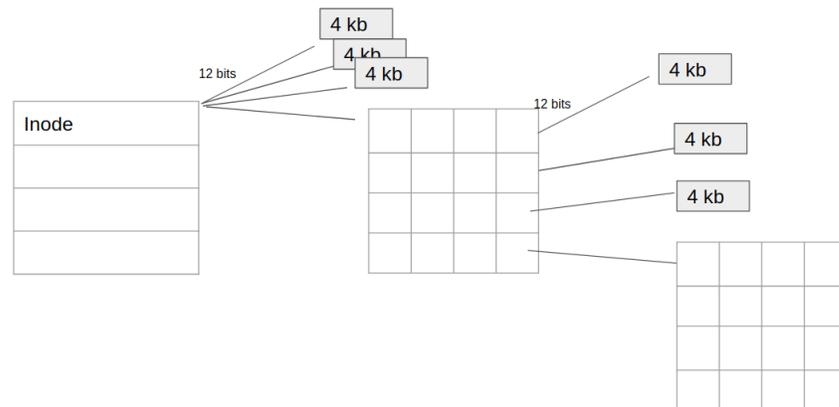


Figure 20.2: inode multi level index

The client needs to send RPC to server for every action. The file is stored on the server. There is excessive communication happening in this model. The server needs to maintain the state to know the status of communication with a client.

2. Upload/download model : Work done at client. It is a stateful server. Simple functionality. Moves files/blocks, need storage.

The server moves the files from local storage to client storage. The client performs the processes and pushes the file back to server. This makes the server stateless. This is very slow in the beginning in order to load the file.

Question: In the upload/Download model, what happens when there is concurrent access to the same files ?

Answer: Do not make the two concurrent process copy the original file as they will overwrite each others' changes. The solution is to have your clients send all their requests to the server and make changes from there.

20.2.3.2 System Structure: Server Type

Stateless server

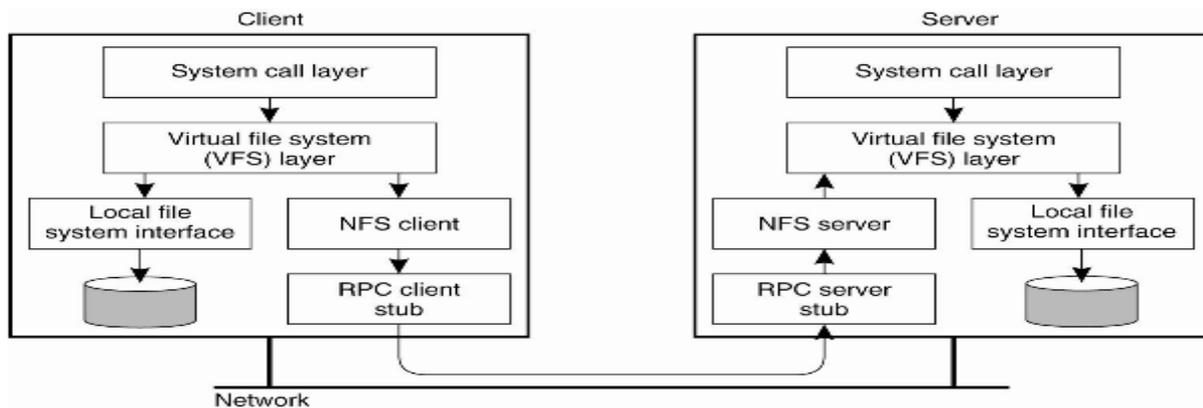
1. No information is kept at the server in between client requests
2. All information need to service a request must be provided by the client with each request
3. More tolerant to server crashes

Stateful server

1. Server maintains information about client accesses
2. Shorted request messages
3. Better performance
4. consistency is easy to achieve

20.2.4 NFS Architecture

The NFS architecture is shown in fig 20.3.



Most widely used distributed file system that uses a virtual file system layer to handle local and remote files.

Figure 20.3: NFS Architecture

Either the data is on local FS or the data is on another machine. Then we go through the NFS. The NFS, as can be seen from the figure, is essentially an add-on to the OS.

Question:What is the system call layer in the NFS architecture ?

Answer: This is an interface that the operating system is exporting to the application.

It should be noted that the NFs is just a layer that sits on top of the regular file system, but is not a file system in itself.

Question: Is the server stub going to aler the client when there are is a set of files available ?

Answer: NFS likes like any other file system. You send a specific request to the server and you get an equally specific response depending on whether the file exists or not.

20.2.4.1 NFS Operations

The basic NFS operations can be referred on the slides. The highlights here are that the v3 NFS was stateless where v4 is stateful. *Open* and *Close* were not present in v3 because it was stateless. These operations are present in v4.

The server exports the directory that will be mount on that client so as to be accessed locally, similar to how a network drive is set up in windows.

Question: If NFS V3 is stateless, what happens to the server if you mount 1000 volumes ?

Answer: We are obtaining client resources, not server resources so it has a little to no impact.

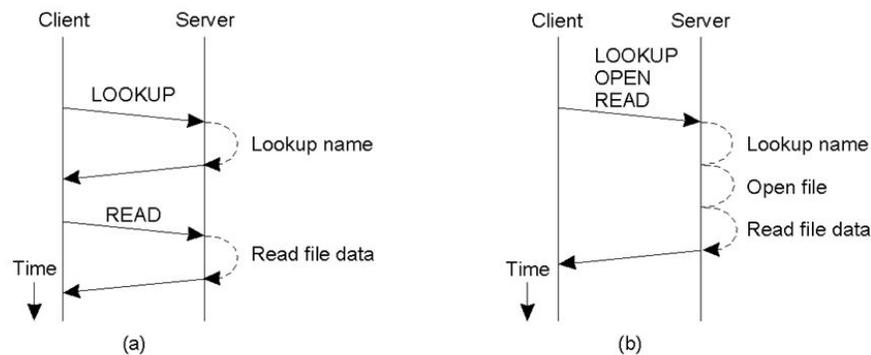
Question: How do you handle duplicates on the local system and the server ?

Answer: NFS won't handle duplicates for you. However, since they are mounted at different mount points, they will have different paths.

20.2.4.2 Communication

The communication between client and server between v3 and v4 of NFS is represented in fig 20.4

Communication



- a) Reading data from a file in NFS version 3.
- b) Reading data using a compound procedure in version 4.

3

Figure 20.4: communication in NFS

In v3, the client takes the open call and does a *lookup*. If the file exists, server returns the lookup name. The client then sends a *read* call.

In v4, a bunch of calls can be sent like lookup, open and read. The server performs all the actions and returns to client. The v4 is faster because of lesser *round trip time*.

20.2.4.3 Naming: Mount Protocol

Mounting tells the OS about where to put the external storage like USB etc. The OS takes care of the mapping. Users can access remote files using local names.

20.2.4.4 Automounting

In a multiuser setting, 2 different users can share data together. In v3, mounting mounts everything the NFS has access to. But, with automounting, the user can mount exactly what is needed. It is basically, mount on demand.

Attribute	Description
TYPE	The type of the file (regular, directory, symbolic link)
SIZE	The length of the file in bytes
CHANGE	Indicator for a client to see if and/or when the file has changed
FSID	Server-unique identifier of the file's file system

Figure 20.5: Mandatory attributes for NFS

20.2.4.5 File Attributes

The attributes in fig 20.5 is mandatory for every FS. With v4, there were additional attributes that are recommended which are listed in fig 20.6

Attribute	Description
ACL	an access control list associated with the file
FILEHANDLE	The server-provided file handle of this file
FILEID	A file-system unique identifier for this file
FS_LOCATIONS	Locations in the network where this file system may be found
OWNER	The character-string name of the file's owner
TIME_ACCESS	Time when the file data were last accessed
TIME_MODIFY	Time when the file data were last modified
TIME_CREATE	Time when the file was created

Figure 20.6: Recommended attributes for NFS

20.2.4.6 Semantics of File Sharing

If a file is shared among users concurrently, it is necessary to define the semantics of reading and writing. The semantics which enforces an absolute ordering on file reads and writes is known as **UNIX semantics**. This semantics is generally desirable as it avoids inconsistency issues across different users. However, for a distributed system, Unix semantics can only be achieved if there is one file server and clients do not cache files. A no cache policy, for a distributed file system, can cause serious performance issues. Hence weaker semantics are adopted. By allowing clients to update in local caches might create inconsistent versions of files across users. For example, if a client locally modifies a cached file and shortly thereafter another client reads the file from the server, the second client will get an obsolete file. To avoid these issues, a weaker semantics known as **Session semantics** is adopted. In this semantics, only when the file is closed the changes are made visible to other clients. A description of different file sharing semantics is shown in Figure 20.7.

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transaction	All changes occur atomically

Figure 20.7: Different File sharing semantics

Question: Does the NFS support horizontal partitioning ?

Answer: NFS just mounts the already partitioned volumes (which is usually done by the systems administrator)

20.2.4.7 File Locking in NFS

Operation	Description
Lock	Create a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the lease on a specified lock

Figure 20.8: NFSv4 operations related to file locking

NFS allows clients to use locking at a file system level. Locking can also be enabled for a particular section of a file. Typically applications for databases and emails use file system level locking. Locking was not part of NFS until version 3. NFS v4 supports locking as part of the protocol. A description of several file locking operations of NFS v4 is shown in Figure 20.8.

Question: Can you implement a lock by creating a lock file on the disk ?

Answer: Yes, you can, but you have to make sure that access to that file is atomic.

Question: How does client caching that supports open delegation handle failures ? **Answer:** This is not resilient to failure. If failure occurs, changes will be lost.

20.2.4.8 Client Caching

NFS does not enforce any caching policy to clients. Figure 20.9 depicts the client side implementation of caching in NFS. The client talks to memory cache and then to disk cache to access a file. If the file is not present in the caches, then the request is forwarded to the NFS server over RPC call. NFS by default implements

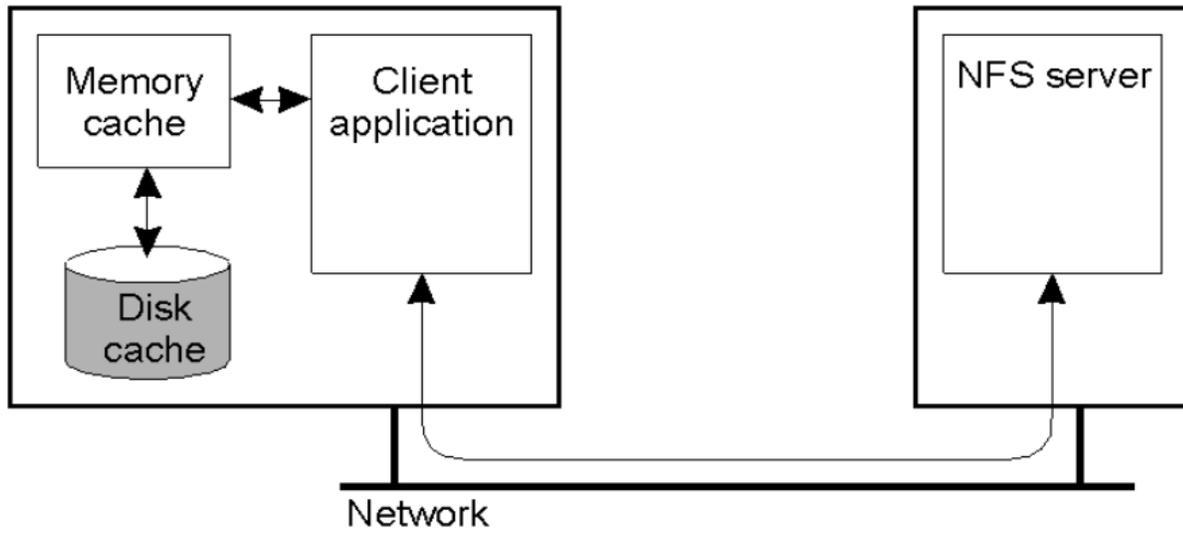


Figure 20.9: Client side caching in NFS

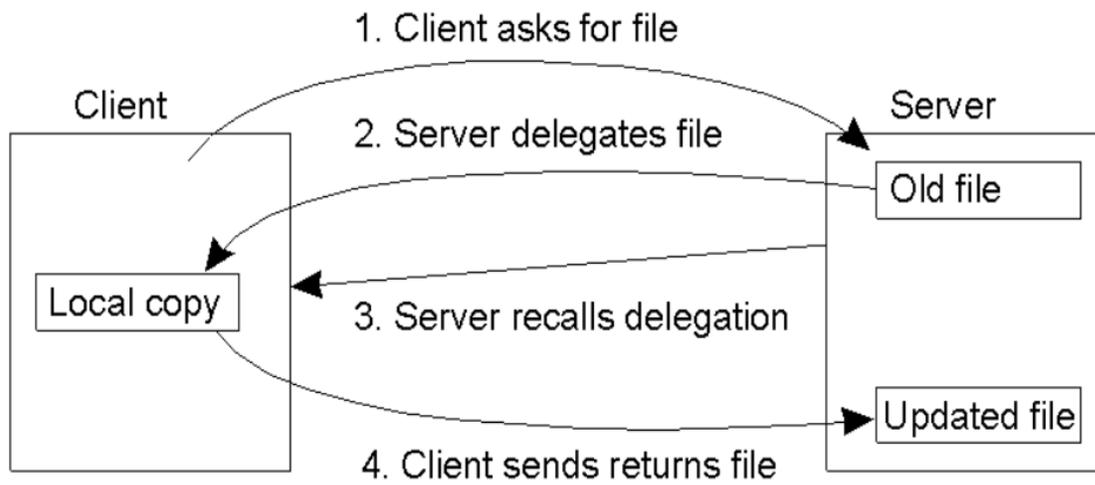


Figure 20.10: File delegation in NFS v4

remote access file service model. NFS v4 extends this architecture to accommodate upload/download file service model as well. If a single client is connected to the file server, the server delegates the master file to the client. All read/write operation occurs at the delegated copy. This is known as **File Delegation**. When the file is closed or when the server recalls the delegation, the delegated file is sent back to the server. The overall file delegation procedure is depicted in Figure 20.10.

20.2.4.9 RPC Failures

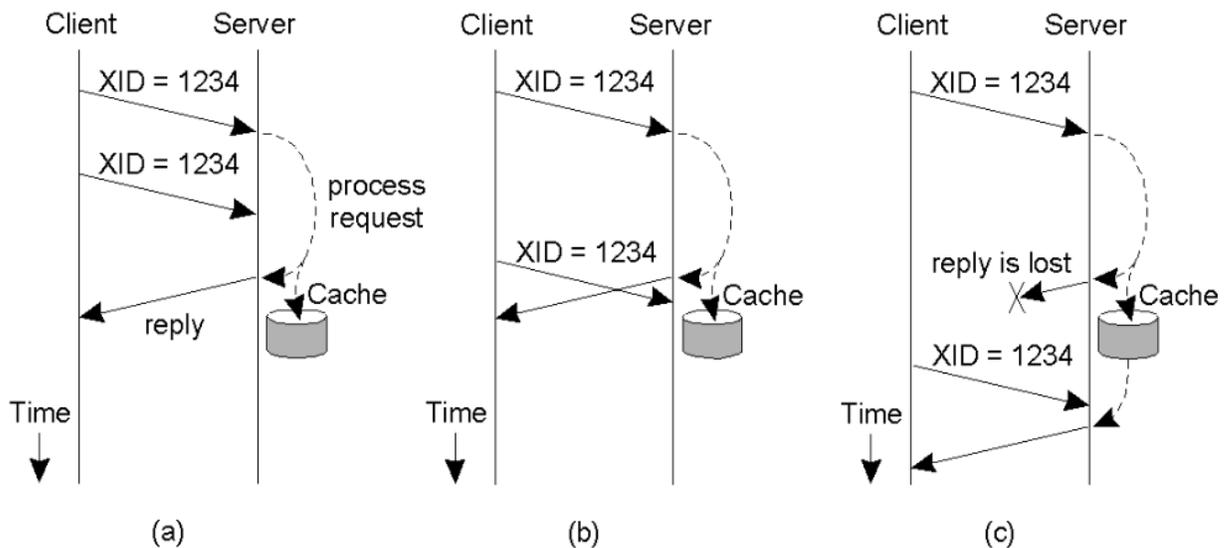


Figure 20.11: RPC retransmissions using a duplicate request cache

For NFS request over UDP, both the client and the server have to implement timeout and retransmission mechanism due to unreliability of UDP protocol. Generally RPC file service model is adopted. Retransmitting an RPC request requires the RPC operation to be **idempotent** i.e. any number of execution of the same RPC request should produce same result. Idem-potency is achieved through a duplicate-request cache. Each RPC is assigned a transaction Id. The cache stores the results for a transaction. When ever a RPC request comes, it is first checked with the cache. If a cache hit occurs, the result is returned. If not, the transaction is executed, the cache is updated and the result is returned back. Figure 20.11 depicts retransmissions of RPC requests using a duplicate request cache.

20.2.4.10 Handling re-transmissions

Use a duplicate request cache if; The request is still in progress, the reply has just been returned or the reply has been some time ago, but was lost

20.2.4.11 Security

Secure RPC is used in NFS v4. NFS V4 , all client server communications are encrypted.

20.2.4.12 Replica servers

NFS v4 supports replication of servers. This is implementation specific.