**CMPSCI 677 Operating Systems** **Spring 2014**

# Lecture 20: April 09

*Lecturer: Prashant Shenoy* *Scribe: Deepak Arora*

## 20.1 Stand-alone (UNIX) File System

File : Named collection of logically related data.

File System : Provides transparency with respect to actual storage, modification along with functions to manipulate local/remote files in a user friendly manner.

Files are stored as uninterpreted sequence of bytes. Data inside the file is not associated with the file type. Directory comprises of file (table) that stores names of other files, metadata(inode), devices associated with those files. Inodes store file attributes and a multi-level index(levels of pointers) that has a list of disk block locations for the file. To read the file we first need to locate the metadata. Refer slides for different components of Inode and directory.

## 20.2 Distributed File system

A Distributed file system is a system that allows a machine to access files on another machine.

There are two canonical accessing models as a part of the file service for distributed file systems:

**Remote access model**
All read/write requests to a file are processed on the server.

**Upload/download model**
When a client opens a file, a copy of the file is transferred (downloaded) to the client. All the subsequent read/write operations are done on the local machine(cached copy). Once the file is closed, the result will be sent back (uploaded) to the server.

Consistency is simpler to achieve in the remote access model than in the upload/download model, because in the former model all the requests are handled on the server, which ensures consistency is attained. As for the upload/download model, there might be multiple clients opening the same file simultaneously. Keeping multiple copies of the same file on different machines makes consistency more challenging.

Efficiency is better in case of Upload/Download model in case of longer retained/frequentlly accessed files since it avoids frequent calls to the server. For infrequent read/writes Remote Access model is more efficient.

Distributed file system server can be categorized into two types: stateless server and stateful server. A stateful server keeps information such as which client is currently accessing a file and which file is opened on the server side; while a stateless server keeps no information with that regard.

Stateless servers over stateful servers are,tolerant to server crashes, easily scalable, consistency is to be provided by the client, all information needs to be provided by the client for each request leading to decline in performance in case of frequent similar requests, is not idempotent since no track of request/clients in case of failure.

## 20.3 Network File System

Network File System (NFS) is a distributed file system protocol developed by Sun. In the case the file being accessed is local, the file accessing mechanism on an NFS machine sends a request to the local file system interface to access the local storage. On the other hand, if the file is sitting on the remote machine, a request will be sent to the NFS client, which in turn makes an RPC call to the NFS server. Once the NFS server received the RPC call it will access the local file system and send the results back to the client. As a matter of fact, RPC was invented by Sun for implementing NFS.

NFS is a layer sits on the local file system which makes file available to remote machines. VFS(Virtual file system) layer examines the system call and decides the file system requested by system call. Deciding if its a local file or a remote file it is forwarded to respective file system(NFS,Local,FAT).

Currently, the NFS versions still in use are version 3 and version 4. NFS version 3 follows a stateless server design, whereas version 4 is a stateful server design. NFS protocol defines several file accessing operations. Some of the operations are not supported by both version 3 and version 4. For example, there is no "open" operation in NFS version 3, which is stateless, meaning that servers do not need to keep track of file open states of individual clients.

In particular, opening a file in NFS version 3 is translated into a lookup operation to check if the specified file exists and the client has sufficient permission to access the file. Followed by a lookup, subsequent read requests will then be sent to the server in order to read data. For reading data in NFS version 4, the server only needs to keep track of which client is opening which file. As a result, clients can group three operations *lookup, open, read* together as one compound RPC request. This decreases overall RPC latency and leads to faster, performance optimized file accessing.

### 20.3.1 Mount protocol

Mount protocol establishes a mapping of a file system subtree on the server to local file hierarchy of the client.

Notice that NFS version 3 does not support transitive exports. For instance, say server $B$ exports a subtree $T_1$ to server $A$, and server $A$ exports a subtree $T_2$ that contains $T_1$ to some client. For security reason, the client is not allows to directly access $T_1$.

Table "FSTab" created at boot time maintains mapping of the hierarchy for instance network drives are mounted as a labelled volume on local machines. Mount can be performed on demand instead of at the boot time. In particular, when user tries to access an NFS directory, it will be mounted automatically(on demand) – this is called automounting.

Transitive sharing of volumes to machines other can be selected in NFS v4. If the volumes is to be accessed only with the machine it is exported to then its access should be limited accordingly. Previous versions of NFS did not provide for this flexibility, it simply restricted the access to the exported machine for security reasons.

### 20.3.2 File Attributes

NFS designed to be a general File System. It requires very little support from the underlying File System. NFS supports compatibility with underlying FS by using a few mandatory attributes(generalized to use even on a rudimentary FS like FAT) and allows better control over the files using the recommended attributes.

### 20.3.3 Consistency

There are several types of file sharing semantics. UNIX semantics means that each operation is performed immediately and is guaranteed to be visible to all processes accessing that file. UNIX semantics are expected When operations are done locally.

NFS provides weaker semantics(session) because of consistency reasons. When a process A writes to a file on the server, and a process B on another machine reads that file right after that, what process B reads is not guaranteed to include the change made by A because the modification might be cached locally at A and not yet updated with the server.

NFS protocol does not specify consistency requirements but leave it to the implementation. In the current implementation, clients synchronize changes with server every 30 seconds.

If we need to implement UNIX semantics instead of session semantics we have to use remote access model discussed before.

### 20.3.4 File locking

OS allows processes acquire locks when accessing files to prevent multiple processes from writing to a file at the same time. For instance, the inbox file in UNIX is protected with file lock. Because NFS version 3 is stateless, file locking is not supported. On the other hand, NFS version 4 is stateful and it allows client lock a file or even just a part of the file (a range of bytes in the file).

### 20.3.5 Client Caching and Delegation

Session semantics are required in NFS hence caching is allowed. Cached data is allowed to be kept for 30 seconds. If NFS is stateless, caching helps overcome the limitations.

Delegation is the act of transferring responsibility over an object (e.g.: a file), which is tightly related to the idea of transferring files in the download/upload access model. NFS by default only supports remote server mode in version 3. Delegation was introduced from version 4. When a client asks for a file, the server transfers a copy of the file to the client, called *master copy*, and all the subsequent changes are made locally and therefore no RPC requests are sent since then until the client closes the file.

If multiple processes from different machines want to access the same file, the server can recall delegation by asking the delegated client send the master copy back and the server then switch to remote access mode. NFS version 4 supports both remote access mode and upload/download mode.

As we can see, delegation impacts scalability. On the one hand, delegation aims at optimizing performance. On the other hand, being stateful hurts performance and scalability. In general, NFS version 4 is less scalable, not for its stateful design but because it supports a much richer set of features that complicate the entire system.

### 20.3.6 RPC failures

Original NFS runs on UDP, while more recent versions of NFS run on TCP and hence deal with packet losses at the TCP level (although it is still possible to set it to work over UDP).

To use UDP, NFS has to handle potential packet loss by itself. It is common to use a time-out to determine

if a packet is lost or not. Say if an NFS client does not receive the response from the server within a period of time, it will then re-issue the request. To guarantee correctness, operations have to be idempotent, that is, repeating an operation multiple times will result in the same outcome as being done exactly once.

To achieve this, all the requests are logged when processed and the results are cached. Once a request comes in, the server checks the log if the request has already been processed. If that is the case, the server simply returns the cached output. By doing so, exactly once semantic is provided.

### 20.3.7 Security

Server authenticates the clients to allow access. Requests from clients are not encrypted, secure RPC's on NFS v4 takes care of the encryption of the read/write/authentication requests. Handled by the RPC subsystem. NFS only suggests the type of encryption to be used.