Network File System - NFS

- NFS is a distributed file system (DFS) originally implemented by Sun Microsystems.
- NFS is intended for file sharing in a local network with a rather small number of computers.
- NFS is both a specification and an implementation of a DFS. Today three versions exist of the specification - version 2, version 3, and version 4.
- The text in the course book is based on v2 (and v3) of the specification. Implementation dependent parts are based on the Solaris implementation.
- Every machine may be both client and server.
- In order to make files on other machines available, a mount operation is needed.
- Information about a mounted file system is only stored on the clients, not on the server.

NFS Specification

- The NFS specification distinguishes between the services provided by the mount mechanism and the remote file access services.
- Two different protocols are specified - one protocol for mounting and one protocol for performing file operations - the NFS protocol.
- The protocols are specified as sets of RPC:s.
- NFS is required to work also if the client and the server has different processor architectures.
- The RPC routines use a coding called XDR (External Data Representation) to code data sent across the network in a way that is independent of the memory architecture (*big-endian* versus *little-endian*).
- Network data is sent using UDP in NFS v2. NFS v3 also allows the use of TCP.
- Because the XDR-interface also is implementation independent, all file system code above the XDR level is independent of the actual network.

The Mount Protocol

- The mount protocol is used to mount a directory from another machine in the local file system tree.
- A mount operation includes the name of the remote directory to be mounted and the name of the server machine.
- The server maintains an **export list (/etc/exports)** that specifies which filesystems may be exported and which machines that are permitted to mount them.
- The server returns a file handle to the client. This file handle serves as a key for further accesses to the mounted file system.
- For UNIX file systems the file handle consists of a file system identifier and an inode number for the mounted directory.
- Usually the clients use a static mounting configuration, specified in the file */etc/fstab*.

The NFS Protocol

The NFS protocol provides a set of RPC:s for remote operations.

The procedures support the following operations:

- Searching for a file in a directory
- Reading a set of directory entries
- Manipulating links and directories
- Accessing file attributes
- Reading and writing files

Open and close are not included because NSF use a *stateless* server.

NFS requests do include a sequence number that is stored on the server and used to detect duplicated request. This status information do not invalidate the stateless property because it is only considered as a hint and is not restored after a server crash.

File locking is not part of NFS but is provided as a separate service.

The NFS Protocol

- The RPC routines are *blocking*, that is they do not return until the requested service is completed. Together with a stateless server this have the effect that a write operation cannot be completed until the data is written to the server disk.
- This mode of operation can give very bad write performance.
- NFS v3 also specifies an asynchronous write mode. This mode trades better write performance against an increased risk for corrupted files.
- The NFS RPC procedure calls may contain 8 Kbyte of data and are supposed to be atomic.
- A problem is that the Ethernet data size limit of 1500 bytes forces the packets to be fragmented at the IP level. If UDP is used there is no error handling below the UDP level, thus if a single fragment is lost all 8 KB have to be retransmitted. This is the main reason why TCP is in fact a better choice than UDP.
- NFS v2 have no other access control than the file handle and an UID included in the datagrams (all sent in clear text). The use of UID:s means that the client and server must use common UID:s. The root UID is not accepted but translated to nouser.

NFS Architecture

The NFS architecture consist of three levels:

1. The UNIX file system interface with the system calls open, close, read and write.
2. The Virtual File System (VFS) level.
   (a) Directs the call to the correct file system implementation.
   (b) Based on a data structure called **vnode**.
3. The local file system or the NFS-interface.

Remote Operations

- In principle all RPC operations result in communication with the server. In reality caches are used to improve performance.
- Note that only the server is stateless. The clients are not and may use arbitrary caching strategies.
- There are two caches in the clients. One for file attributes (inode information) and one for data blocks.
- The data block cache normally uses a delayed-write strategy for writing.
- The effect of this is that UNIX semantics are not preserved nor do it follow any other easily defined consistency semantics. New files created by one machine may not be visible to other machines for 30 seconds.

Andrew File System - AFS

- Developed at Carnegie Mellon University with start 1983.
- A commercial version was developed by Transarc Corporation, which was subsequently acquired by IBM.
- In 2000 IBM terminated the commercial version of Transarc DFS, and made the source available as an open-source product, termed OpenAFS.

AFS

- Designed to make it possible for 5000 workstations to share a common filesystem.
- The file system tree have a local part and a shared part. The local part is the root file system. The shared file system is located under a certain directory in the local file system.
- Every client need to have a local disk memory to store the root file system and also used for caching.
- Dedicated file server machines collectively called *vice* are used.
- The file server machines cooperate to present the clients with a common location independent file system tree that looks alike at all clients.

AFS

- Viewed at a finer granularity, clients and servers are structured in clusters interconnected by a WAN.
- Each cluster consists of a number of workstations on a LAN and a **cluster server**. Each cluster server is connected to the WAN by a router.
- The reason for this decomposition is to make the system scalable. For best performance the clients should use the server on their own cluster most of the time.

AFS

A few other design issues:

Security

- Because vice are executed on dedicated servers and no client programs are executed on the servers, the security of the files can be guaranteed provided that a secure communication protocol is used.
- This requires an authentication process there both the client and the server identity is verified. The authentication is based on encryption and built into the RPC package.
- All data sent between the client and the server are encrypted.

Protection

- In addition to the traditional UNIX file access bits AFS have access control lists.

AFS

The shared name space

- The shared file system is made up of component units called volumes. A volume is rather small and typically contains the files for a single user.
- Internally vice identifies a file with a position independent file identifier called a **fid**. A **fid** is 96 bits long and consists of volume number, vnode number and uniquifier. The vnode number is unique within a volume. The uniquifier allows reuse of vnodes.
- The volume number itself do not tell where the volume is stored.
- The current location of a volume is found in the **volume-location database** replicated at every server.
- This organization makes it possible to move a volume to another server while she system is running without disturbing the clients.

AFS

Caching and consistency semantics

- AFS uses disk caches at the clients and caches in chunks of 64 kBytes.
- Session semantics is used. The operating system on each client intercepts file-system calls and forwards them to a user level process, called *Venus*. *Venus* caches files when they are opened and writes them back to the server when they are closed.
- Read and write operations use the local cache.
- To keep the caches consistent a server initiated strategy called **callback** is used. The server keeps status information about which clients have a file cached. If the file is modified the server notifies all clients that have the file cached.
- Thus if a client reopens a cached file it can use the local cache and need not contact the server unless it has been informed that the cache is invalid.
- Venus also caches contents of directories and symbolic links for pathname translation. All lookups are done locally in the cached directories.
- The only exception to this caching policy is modification to directories that are made directly at the server.

NFS V4

- Closer to AFS than to earliar versions of NFS.
- Stateful server.
- File locking integrated into protocol.
- Mount protocol removed.
- NFS protocol now supports *open* and *close* operations.
- A COMPOUND operation added.
  - ➝ NFS procedures are now groupable.
- Uses TCP as transport protocol.
  - ➝ Uses only one well defined port (2049).
- Strong security based on GSS_API (Generic Security Service API).
  - ➝ Kerberos V5.
  - ➝ LIPKEY (A public key protocol).

Memory Mapped Files

- The most common way to handle files use operations like *read* and *write*.
- An alternative way is to map the file into the virtual address space and use normal memory access operations.
- This method requires two extra system calls:

MAP(file,address)	Map the file, *file*, to the address *address* in the process's address space.
UNMAP(address)	Remove the previously mapped file from the process's

Copying of file abc to file xyz with this method involves the following steps:

1. Map the file *abc* into a memory segment
2. Create the file *xyz* and set it's size with *ftruncate*
3. map the file *xyz* into a memory segment
4. Copy memory area *abc* to memory area *xyz*
5. unmap memory segments *abc* and *xyz*

Memory Mapped Files

Memory mapping is implemented as part of the virtual memory.

- When a file is mapped into the virtual memory, a page table is created for the new segment and all pages are marked as invalid. The mapped file is set as paging memory for the segment.
- When the segment is referenced a page fault interrupt is generated and the faulting page is brought in by the virtual memory.

Advantages

- When a normal read operation is used data is read from disk memory to a system buffer. From the system buffer data is copied to a buffer in the process's address space. With memory mapping this last copy operation is eliminated as the data is mapped into the process's address space by the virtual memory.
- The most common use of mmap is to set up segments for shared libraries.

Memory Mapped Files

Problems

- When writing the system cannot know the exact length of a file unless it is assigned by a separate system call like *ftruncate*.
- If memory mapping and conventional I/O is used concurrently at the same file, inconsistent versions of the file may be seen. This problem have been solved in most systems by using the same buffering mechanism for the file system and the virtual memory.
- The address space in a 32-bit architecture is not always big enough to map a complete file.