Network File System

Giovanni Agosta

Piattaforme Software per la Rete – Modulo 2

Outline

1. NFS Concepts
2. NFS and Mount Protocols
   - NFS Implementation
   - The Mount Protocol
3. NFS in Linux

Remote File System Access

Generalities

Possible solutions

File transfer access to remote data at the granularity of entire files

File access access to remote data at the granularity of blocks

Remote File Access

- Run a server on the machine where the files reside to:
  - Respond to access requests
  - Check authorization credentials
- Several mechanisms have been defined for this purpose
  - We focus on Sun Microsystems Network File System

Remote File System Access

Issues

Required Functionalities

- Read
- Write
- Create
- Destroy
- List/Navigate Directories
- Authenticate requests
- Honor protection levels

Heterogeneous Computers

Different systems may have different ways of:

- Denoting directory paths
- Defining file names
- Storing file information
- Defining file operation semantics

Sun Network File System Design

Key design choices

Stateless Server

- Scalable solution
- No need to keep track of server crashes/reboots
- No way to keep track of current positions (in file or directory)

NFS and UNIX File Semantics

- NFS designed to accomodate heterogeneous file systems
- But based on UNIX file system semantics

Sun Network File System Design

Key design choices

NFS File Types

```
enum ftype {
NFNON = 0, /* Not a file */
NFREG = 1, /* Data file */
NFDIR = 2, /* Directory */
NFBLK = 3, /* Block device */
NFCHR = 4, /* Character device */
NFLNK = 5, /* Symbolic link */
}
```

NFS Mode

- From UNIX
- File types
- uid, gid, swap
- Permissions

Sun Network File System Design
Client and Server

Server

- Runs on a file server machine to provide access to local files from other machines
- Receives requests that do not need path interpretation

Client

- Runs on a machine that wants to acces remote files
- Manages all the path decoding, interacting with the server for each component
- The open primitive decides whether to use local or NFS implementation depending on the path name
- On UNIX, this is achieved by use of the mount mechanism
- NFS version 4 integrates mount routines

Sun Network File System Design

File Handles

Client Side

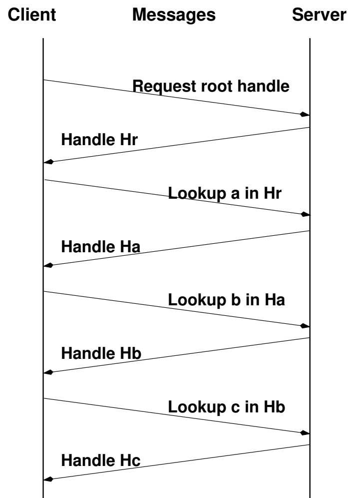
- A 32-byte string
- Received from the server at file opening
- Opaque data

Server Side

- Fabricated from the server in any convenient way
- Server must be able to decode handles (*stateless*!)
- Can encode information for quick decoding
- Can have time-limits (for security purposes)

Sun Network File System Design

NSF File Path Decoding



Path Decoding

- Only appens in the open call
- Requires server-client interaction
- Each path component is navigated independently
- Each directory is associated to one handle
- No state is needed, as per specification

Sun Network File System Design

File & Directory Positioning

Stateless File Positioning

- File position maintained by the client
- lseek implementation completely local
- Position information sent at every read/write request

Stateless Directory Operations

- Listing for directories with many entries cannot be transmitted in a single message
- Use a position identifier (magic cookie) in a way similar to file positioning
- Unsafe with respect to concurrent directory modification

Sun Network File System Design

Handling Multiple Roots

Problem

- What if you want multiple directory hierarchies within the same NFS server?
- Initially, impossible: NFSPROC_ROOT function accessed a single root
- Additional mechanisms needed to replace NFSPROC_ROOT

Solution: Mount Protocol

- Provide a list of directory hierarchies
- Accept full path names to directory hierarchy roots
- Authenticate client requests and check permission to access a directory hierarchy
- Provide file handle for each directory hierarchy root

NFS Implementation

Overview

- Use RPC mechanism to implement NFS
- Each remote file operation corresponds to a remote procedure call
- Change of perspective:
  - From RPC as a way to divide a program into components
  - To RPC as a way to define a protocol not tied to a specific program

NFS Implementation

Protocol Implementation with RPC

When using RPC for a program

- Start with existing procedures and data
- Needs are clearly defined by existing interactions

When using RPC for a protocol

- Need to guarantee interoperability for programs that adhere to the protocol (precision)
- Need to to allow a wide variety of implementations (generality)
- Cannot be designed without significant effort and competence

NFS Implementation

Protocol Implementation with RPC

What must be done

- Provide declarations for constants, types and data structures used as procedure parameters
- Provide declarations for remote procedures
- Provide definitions of the semantics

Servers in RPC-specified protocols

- A server is a single remote program
- Only the client can initiate operations
- Each operation message maps to a remote procedure call
- Each return message maps to a remote procedure return

NFS Constant, Type and Data Declarations

Constants

Basic constants

```
const MAXDATA    = 8192; /* Max bytes in msg */
const MAXPATHLEN = 1024; /* Max chars in path*/
const MAXNAMLEN  =  255; /* Max chars in name*/
const COOKIESIZE =    4; /* Cookie size      */
const FHSIZE     =    4; /* File handle size */
```

NFS Constant, Type and Data Declarations

Constants

Error Reporting

```
enum stat {
  NFS_OK         = 0,   /* Success              */
  NFSERR_PERM    = 1,   /* Ownership error      */
  NFSERR_NOENT   = 2,   /* File does not exist  */
  NFSERR_IO      = 5,   /* I/O error            */
  NFSERR_NXIO    = 6,   /* Address does not exist*/
  NFSERR_ACCES   = 13,  /* Permission denied    */
  NFSERR_EXIST   = 17,  /* File alreay exists   */
  NFSERR_NODEV   = 19,  /* Device does not exist*/
  ...
} ;
```

NFS Constant, Type and Data Declarations

Types

File name and handle types

```
/* File name */
typedef string filename<MAXNAMLEN>;

/* File handle */
typedef opaque fhandle[FHSIZE];
```

Dates and times

```
struct timeval {
 unsigned int seconds;  /* seconds past epoch*/
 unsigned int useconds; /* microseconds      */
}
```

NFS Constant, Type and Data Declarations I

Directory Operation Data Structures

Arguments

```
struct diropargs {
 fhandle  dir;   /* handle for dir */
 filename name; /* name of file in dir */
}
```

NFS Constant, Type and Data Declarations II

Directory Operation Data Structures

Return values

```
union diropres switch (stat status) {
 case NFS_OK : /* success */
  struct {
   fhandle file;
   fattr   attributes; /* file status */
  } diropok;
 default :      /* failure  */
  void;         /* empty  */
}
```

NFS Constant, Type and Data Declarations

File Attribute Data Structure

```
struct fattr {
  ftype          type;
  unsigned int   mode;
  unsigned int   nlink;
  unsigned int   uid;
  unsigned int   gid;
  unsigned int   size;
  unsigned int   blocksize;
  unsigned int   rdev;
  ...
```

```
  ...
  unsigned int   rdev;
  unsigned int   blocks;
  unsigned int   fsid;
  unsigned int   fileid;
  timeval        atime;
  timeval        mtime;
  timeval        ctime;
};
```

NFS Constant, Type and Data Declarations

Read and Write Data Structures

Write Arguments

```
struct writeargs {
 fhandle   file;    /* handle for file */
 unsigned  offset;  /* position      */
 nfsdata   data     /* data to write */
}
```

Read Arguments

```
struct readargs {
 fhandle   file;    /* handle for file */
 unsigned  offset;  /* position      */
 unsigned  count;   /* bytes to read */
}
```

NFS Constant, Type and Data Declarations I

Remote file service routines

```
program NFS_PROGRAM {
 version NFS_VERSION {
  void        NFSPROC_NULL(void)              = 0;
  attrstat    NFSPROC_GETATTR(fhandle)        = 1;
  attrstat    NFSPROC_SETATTR(sattrargs)      = 2;
  void        NFSPROC_ROOT(void)              = 3;
  diropres    NFSPROC_LOOKUP(diropargs)       = 4;
  readlinkres NFSPROC_READLINK(fhandle)       = 5;
  readres     NFSPROC_READ(readargs)          = 6;
  void        NFSPROC_WRITECACHE(void)        = 7;
  attrstat    NFSPROC_WRITE(writeargs)        = 8;
...
```

NFS Constant, Type and Data Declarations II

Remote file service routines

```
    diropres     NFSPROC_CREATE(createargs)=  9;
    stat         NFSPROC_REMOVE(diropargs) = 10;
    stat         NFSPROC_RENAME(renameargs)= 11;
    stat         NFSPROC_LINK(linkargs)    = 12;
    stat         NFSPROC_SYMLINK(symlinkargs)= 13;
    diropres     NFSPROC_MKDIR(createargs)  = 14;
    stat         NFSPROC_RMDIR(diropargs)   = 15;
    readdirres   NFSPROC_READDIR(readdirargs)= 16;
    statfsres    NFSPROC_STATFS(fhandle)    = 17;
  } = 2;
} = 100003;
```

NFS Constant, Type and Data Declarations

Remote file service routines

Semantics

NFSPROC_NULL No effect, used to test server response

NFSPROC_ROOT Obsolete, single root not used anymore

NFSPROC_WRITECACHE Not used in current protocol

NFS Constant, Type and Data Declarations I

Remote file service routines: Write

Write to File

```
struct writeargs {
 fhandle file;
 unsigned beginoffset; /* obsolete */
 unsigned offset;
 unsigned totalcount; /* obsolete */
 opaque data<NFS_MAXDATA>;
};

attrstat
NFSPROC_WRITE(writeargs) = 8;
```

NFS Constant, Type and Data Declarations II

Remote file service routines: Write

Semantics

- Writes data beginning offset bytes from the beginning of file
- The first byte of the file is at offset zero
- If status is NFS_OK, then attributes contains file attributes after the write
- Atomic operation

The Mount Protocol

Overview

Use

- Provide server path name
- Validate user identity
- Check access permissions

Notes

- Contrary to NFS, it is a stateful protocol
- State used to maintain *mount lists*
- State information not critical
- Integrated in NFS as of version 4

The Mount Protocol

Declarations: Constants and Types

Constants

```
const MNTNAMLEN  =  255;
const MNTPATHLEN = 1024;
const FHSIZE     =   32;
```

Types

```
typedef opaque fhandle[FHSIZE];
typedef string dirpath<MNTPATHLEN>;
```

The Mount Protocol

Declarations: Data Structures

Return values

```
union fhstatus switch (unsigned status) {
 case 0:  /* success */
  fhandle directory;
 default: /* failure */
  void;
};

struct *groups {
name grname;   /* protection group */
groups grnext; /* pointer to next  */
};
```

The Mount Protocol

Declarations: Data Structures

Return values

```
/* list of available hierarchies */
struct *exportlist {
 dirpath filesys; /* pathname */
 groups groups;    /* allowed groups  */
 exportlist next; /* pointer to next */
};
/* list of mounted filesystems */
struct *mountlist {
name        hostname;  /* remote host */
dirpath     directory; /* path name  */
mountlist nextentry; /* pointer to next */
};
```

The Mount Protocol

Protocol routines

```
program MOUNTPROG {
 version MOUNTVERS {
  void        MOUNTPROC_NULL(void)      = 0;
  fhstatus    MOUNTPROC_MNT(dirpath)    = 1;
  mountlist   MOUNTPROC_DUMP(void)      = 2;
  void        MOUNTPROC_UMNT(dirpath)   = 3;
  void        MOUNTPROC_UMNTALL(void)   = 4;
  exportlist  MOUNTPROC_EXPORT(void)    = 5;
 } = 1;
} = 100005;
```

NFS Support in Linux

Overview

- Supported versions: v2, v3 and v4
- Use v3 unless there is a specific reason
- Export specified
- Mount specified in /etc/fstab
- Format: server:path /mountpoint fstype options 0 0
- Filesystem types: nfs and nfs4

NFS Support in Linux

Mount options

Option	Description
soft/hard	Retry indefinitely after timeout (*hard*, default) or fail after *retrans* retries
timeo=n	Timeout before retry (in tenths of seconds)
retrans=n	Number of retransmissions if *soft* (default 3)
rsize=n	Maximum read size (max 1MB)
wsize=n	Maximum write size (max 1MB)
ac/noac	Cache file attributes

NFS Support in Linux

Mount options

- **bg/fg** Retry indefinitely the mount operation in child (*bg*) or terminate with error (*fg*, default)
- **sec=mode** RPC security flavor
- **sharecache/nosharecache** Share data cache when mounting same export multiple times
- **proto=transport** UDP or TCP, v4 defaults to TCP
- **port=n** if 0, rely on portmapper

NFS Support in Linux

Example - Server

/etc/exports

```
/path/remote clienthostname(rw)
```

rpcinfo -p

```
program vers proto   port
100000     2   tcp    111  portmapper
100000     2   udp    111  portmapper
100024     1   udp  47727  status
100024     1   tcp  42454  status
100003     3   udp   2049  nfs
100003     3   tcp   2049  nfs
100005     1   udp   2049  mountd
100005     3   udp   2049  mountd
```

NFS Support in Linux

Example - Server

/etc/fstab

```
localhost:/path/remote /path/local nfs defaults 0 0
```

mount

```
mount -a -t nfs
```