Internet Systems Programming

NFS: Protocols, Programming, and Implementation

Erez Zadok
ezk@cs.columbia.edu

October 25, 1999
The BIG Picture

NFS Server

//etc/exports
//etc/rmtab

nfsd

RPC

XDR

ufsd_read()

nfsd_read()

biod

network

RPC

XDR

NFS_READ

NFS Client

NFS Client (kernel)

/etc/fstab
/etc/mtab

process

read()

nfs_read()
NFS Overview

- using RPC: Remote Procedure Calls
  - which use XDR: eXternal Data Representation
- stateless server
  - crash recovery
- client side caching (data and attributes)
  - request retransmission
- file handles: 32 bytes opaque to client
  - server encodes: fsid, inum, igen, possibly more
XDR: eXternal Data Representation

- de/serializes data into network-order bytes
- repeated calls encode/decode more "XDR" bytes

```c
bool_t xdr_long(XDR *xdrs, long *lp);

struct foo {
    int i;
    char *buf;
};

bool_t xdr_foo(XDR *xdrs, struct foo *foop) {
    if (!xdr_int(xdrs, &foop->i))
        return FALSE;
    if (!xdr_wrapstring(xdrs, &foop->buf))
        return FALSE;
    return TRUE;
}
```
RPC: Remote Procedure Call

- server does:
  \texttt{registerrpc(prognum, versum, procnum, s\_inproc, in, s\_outproc, out);}  
  \texttt{svc\_run()}

- client issues:
  \texttt{callrpc(char \*host, rpcprog\_t prognum, rpcvers\_t versnum,}
  \texttt{ rpcproc\_t procnum, xdrproc\_t inproc, char \*in,}
  \texttt{ xdrproc\_t outproc, char \*out);}

- which contacts server’s portmapper, then RPC server w/ \textit{procnum}.

when client request comes
  - find procnum
  - call \texttt{s\_inproc} to decode client args
  - call \texttt{s\_outproc} to encode output to client
  - return => client returns (or times out)

- \texttt{rpcgen} produces headers and \texttt{.c} stubs from \texttt{.x} files
Additional NFS Components

- on server:
  - mountd:
    - listen for mount requests
    - authenticate requests
    - return root fhandles

- on client:
  - biod: dirty page clustering, simulate async writes

- on both:
  - lockd: coordinates local/remote record locks
    - flock() uses lockd; lockf() only local locks; fcntl() can use both
  - statd: synchronizes lock information
    - client reboot: tell server to release locks
    - server reboot: tell all clients to reclaim locks
  - portmapper: the mother of all RPC servers
Example: mounting a remote server

- get fhandle (via MOUNTPROC_MNT rpc to mountd)
- fill in struct nfs_args
  - struct nfs_args na
- call mount(2) syscall
  - mount("/mnt", flags, "nfs", &na, sizeof(na))
Contents of struct nfs_args

NA->addr {sockaddr_in} (len=16) =
   "02000801803b14640000000000000000"
NA->addr.sin_family = "2"
NA->addr.sin_port = "264"
NA->addr.sin_addr = "803b1464"
NA->hostname = "opus"
NA->namlen = 255
NA->filehandle =
   "008000f4000000002000a000000000000026e065b6c000a00000000000026e065b6c"
NA->version = 3
NA->flags = 0x0
NA->rsize = 4096
NA->wsize = 4096
NA->bsize = 0
NA->timeo = 7
NA->retrans = 3
NA->acregmin = 3
NA->acregmax = 60
NA->acdirmin = 30
NA->acdirmax = 60
## NFS V.2

- Built on top of UDP
- 17 calls

<table>
<thead>
<tr>
<th>Function</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS_NULL</td>
<td>0</td>
</tr>
<tr>
<td>NFS_GETATTR</td>
<td>1</td>
</tr>
<tr>
<td>NFS_SETATTR</td>
<td>2</td>
</tr>
<tr>
<td>NFS_ROOT</td>
<td>3</td>
</tr>
<tr>
<td>NFS_LOOKUP</td>
<td>4</td>
</tr>
<tr>
<td>NFS_READLINK</td>
<td>5</td>
</tr>
<tr>
<td>NFS_READ</td>
<td>6</td>
</tr>
<tr>
<td>NFS_WRITECACHE</td>
<td>7</td>
</tr>
<tr>
<td>NFS_WRITE</td>
<td>8</td>
</tr>
<tr>
<td>NFS_CREATE</td>
<td>9</td>
</tr>
<tr>
<td>NFS_REMOVAL</td>
<td>10</td>
</tr>
<tr>
<td>NFS_RENAME</td>
<td>11</td>
</tr>
<tr>
<td>NFS_LINK</td>
<td>12</td>
</tr>
<tr>
<td>NFS_SYMLINK</td>
<td>13</td>
</tr>
<tr>
<td>NFS_MKDIR</td>
<td>14</td>
</tr>
<tr>
<td>NFS_RMDIR</td>
<td>15</td>
</tr>
<tr>
<td>NFS_REaddir</td>
<td>16</td>
</tr>
<tr>
<td>NFS_STATFS</td>
<td>17</td>
</tr>
</tbody>
</table>

*(why no lseek?)*
Ex: NFS_READ Call

struct readargs {
    fhandle file;
    unsigned offset;
    unsigned count;
    unsigned totalcount;
};

union readres switch (stat status) {
    case NFS_OK:
        fattr attributes;
        nfsdata data;
    default:
        void;
};
NFS V.3

- TCP and UDP
- 64 byte file handles
- files > 2GB
- ACLs supported
- Kerberos authentication type
- All ops return old/new attributes
  - saves on most popular call, getattr (update client caches faster)
NFS V.3 Protocol

- Removed: ROOT and WRITECACHE
- Added:
  - **READDIRPLUS**: 17
    - also returns file handles
    - saves on NFS_LOOKUPs
  - **FSSTAT**: 18
  - **FSINFO**: 19
  - **PATHCONF**: 20
  - **COMMIT**: 21
    - Saves cached data to disk
NFS V.4

- IETF design, not Sun
- Integrated file locking and mount protocol
- Stronger security w/ negotiation
  - Public file handles
  - Works with firewalls & proxies
- Compound operations
- Internationalization
- Better suited for Internet (i.e., WAN)
- Migration and replication
- Extensible protocol
User Level NFS-Based File Servers

- Context switches
- extra communication
- Amd dead/hung?
- CFS: cryptographic file server
Resources

- RFC 1094/1813
  - Usenix papers [Sandberg 84] and [Pawlowski 94]
- NFS V.2/3/4 specs and drafts
- sources to CFS
  - http://www.cryptography.org/
- Amd
- Email: ezk@cs.columbia.edu