Sun’s NFS - Network File System

TCP/IP class

Jim Binkley

outline

- intro
  - rpc
  - xdr
- NFS architecture
- NFS protocol
- some administrative aspects

intro - NFS

- NFS - Network File System
- requirement on UNIX systems, supported elsewhere as well (pcs)
- goal is for files on remote server to appear as if they are mounted locally on client
- hence clients can share
- RFCs for NFS exist but have been deemed historical

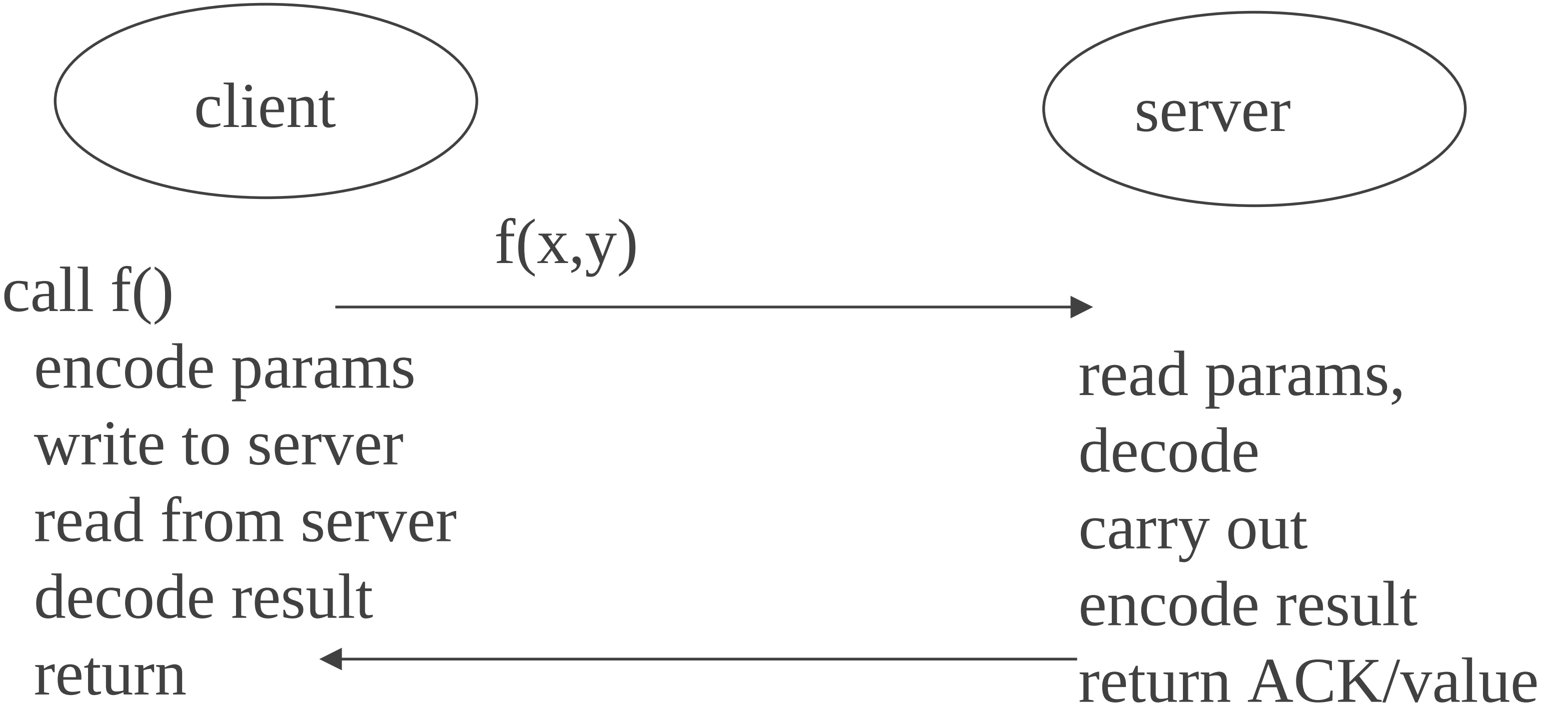
intro - NFS

- built on top of Sun RPC mechanism, “**Remote Procedure Call**”
- RPC gives us client/server focus
- RPC gives a functional interface with parameters that client’s may call
- looks like local function call but is remote (using TCP or UDP) as transports
- note that NFS uses UDP mostly

intro - NFS

- byte order problems dealt with by XDR, a data abstraction language
- **XDR - external data representation**, functions and structures may be declared and compiled down to “stub” code for clients and servers
- programmer must provide functionality, but mindless work of dealing with network byte order is taken care of
- basic rpc paradigm
  - client request f(x,y) sent to server, server carries out and returns ACK or value/s

Remote Procedure Call



client

server

f(x,y)

call f()
  encode params
  write to server
  read from server
  decode result
  return

read params,
decode
carry out
encode result
return ACK/value

XDR - external data representation

- very much like C, way to declare structures and functions, feed to compiler
- rpcgen defs.x -> C code
- couple with rpc library can handle “marshalling” (encoding/decoding) of
- data structures, function parameters, return values

writing a structure across the Net

-
  ```
  struct s {
      char s;
      int x;
      char buf[100;
  } s;
  ```
- not only do we have little-endian, big-endian problem but we have
- compiler offset problem too
- what is offset of int x above? 2/4/8?

two mechanisms to deal with it

- ASCII headers (http/ftp/tar/nntp/smtp)
- TLV (RPC, SNMP, IP/TCP options)
  (Type = INT, len = 4, value)
  (Type = DOUBLE, len = 8, value)
  (Type = BYTES, len = n, bytes...)

intro - how it works from user POV

- mounting
  - client mounts remote file system which must be exported a priori by server

    ```
    # mount foo.com:/usr/src  /remote/src
    (mount remote_dns:path  local_path)
    ```
- after that, you just use it

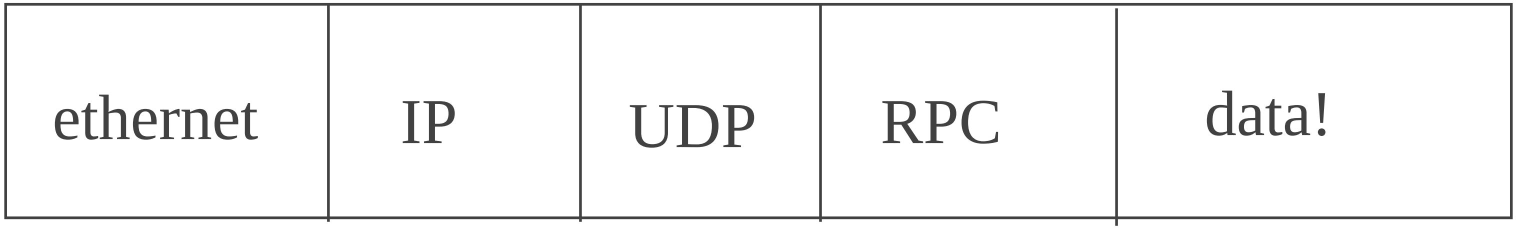
  ```
  % cd /remote/src
  % ls
  ```
- should be able to mount any directory

NFS architecture

- NFS is built on top of RPC/XDR/UDP
- “stateless” compared to TCP
- UDP also felt to be faster as efficiency is important since NFS is compared to local disk speeds (unfair, but so it goes)
- servers presumed local if not on same link
- over WAN, SLIP, might need NFS over TCP (exists but rare)

NFS architecture

- so servers and clients shouldn’t be too far apart
  - NFS adds to congestion...
- encapsulation like so :

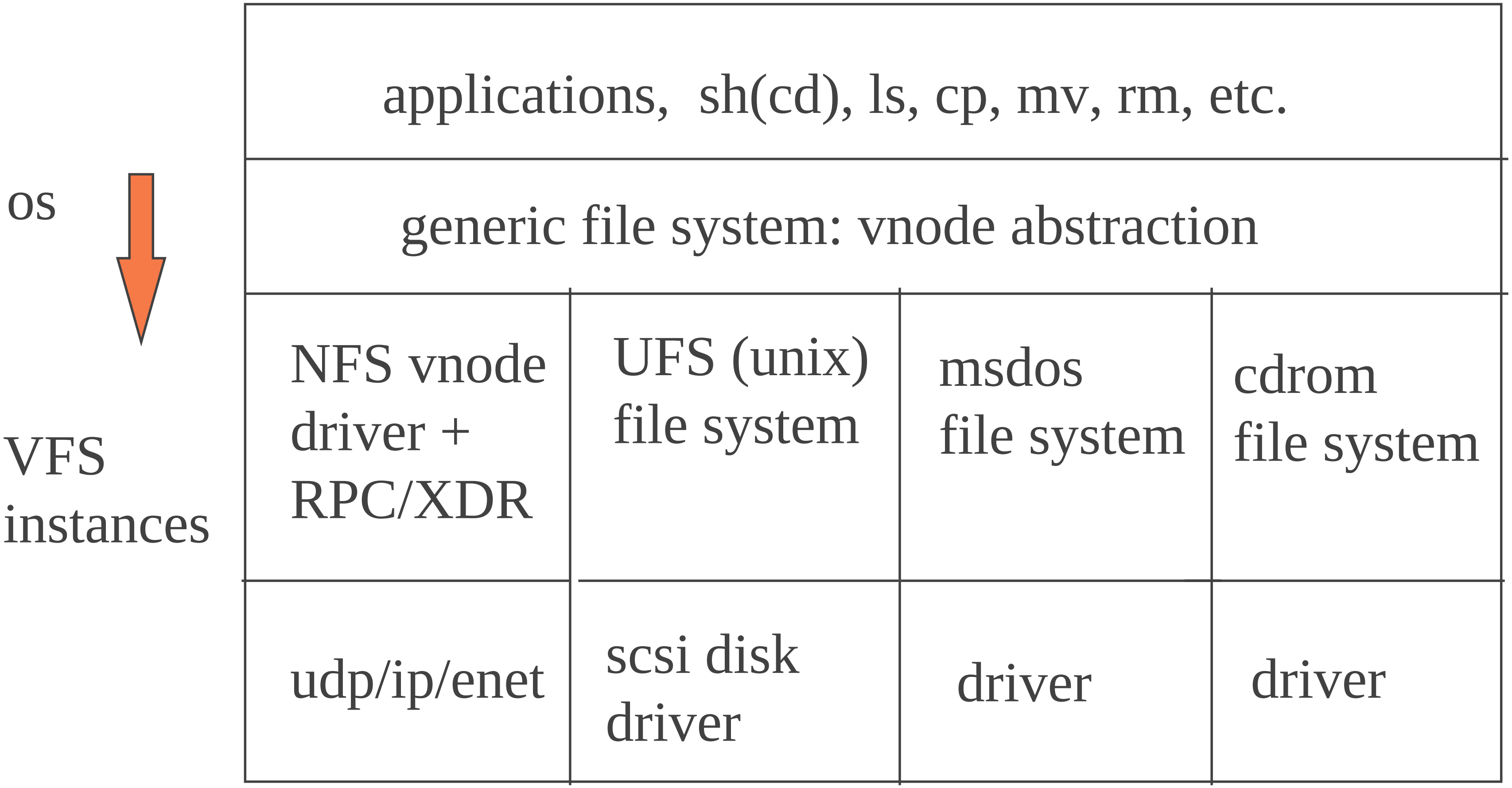


ethernet | IP | UDP | RPC | data!

NFS architecture

- assume SunOS, how did NFS change traditional UNIX?
- introduced notion of “virtual file system”
- + 2-3 protocols needed (using RPC/XDR)
  - mount protocol
  - NFS protocol (read/write data)
  - locking protocol (neglect)

client - Virtual File Systems



os

VFS instances

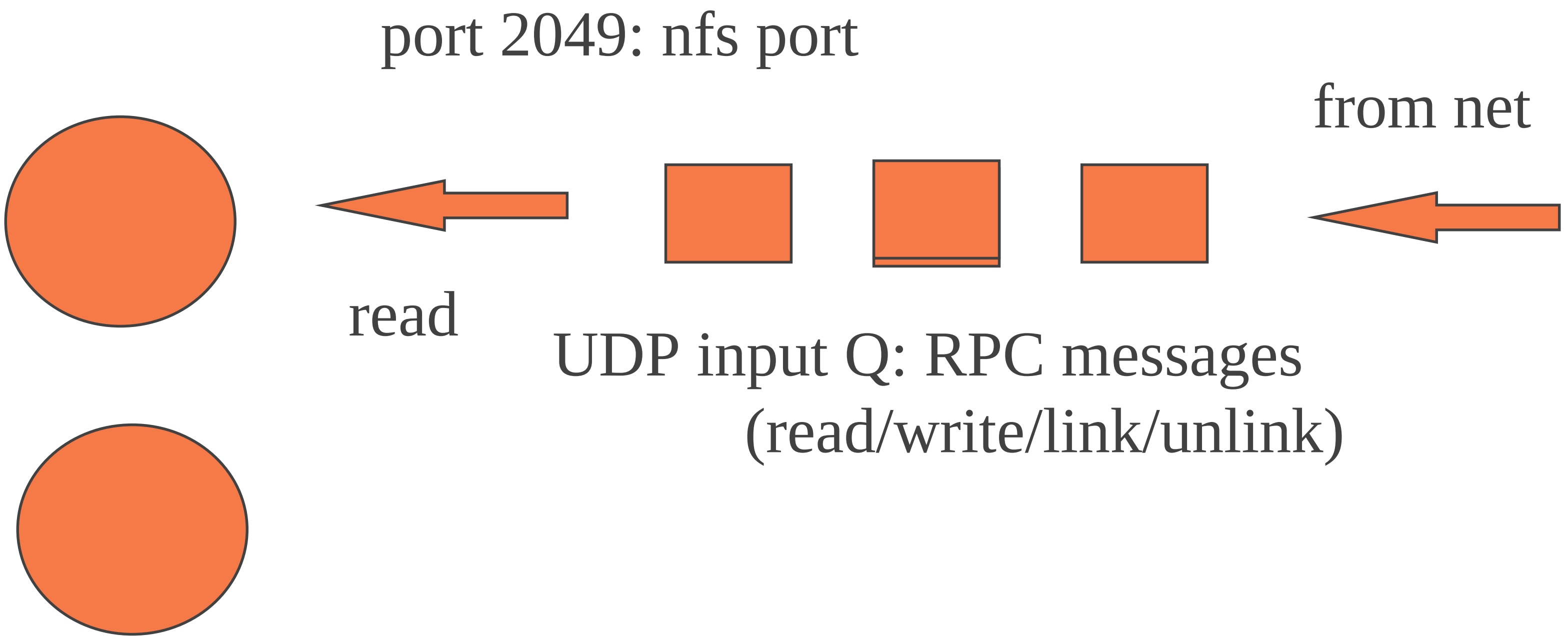
<table>
<tr><td colspan="4">applications, sh(cd), ls, cp, mv, rm, etc.</td></tr>
<tr><td colspan="4">generic file system: vnode abstraction</td></tr>
<tr><td>NFS vnode driver + RPC/XDR</td><td>UFS (unix) file system</td><td>msdos file system</td><td>cdrom file system</td></tr>
<tr><td>udp/ip/enet</td><td>scsi disk driver</td><td>driver</td><td>driver</td></tr>
</table>

Jim Binkley

server

- can think of it as thread that reads/decode RPC messages (read/write, etc.)
- takes RPC message and e.g., on UNIX translates them into UNIX i/o calls, open(2), close(2), read(2), write(2), etc
- reality - server is stateless as possible, no concept of “open”
- server is called at boot as nfsd, typically 4/8/10/12 threads,
- each makes a system call and executes in the operating system for reasons of efficiency

server-side message dispatch



port 2049: nfs port

from net

read

UDP input Q: RPC messages
(read/write/link/unlink)

nfs daemons: UDP reads are atomic, one 1 UDP port shared between 4,8,16 processes

client parts

- o.s. support (virtual file system)
- biod - bio “cache” daemons for typical UNIX style read-ahead, write-behind.
  - app reads 1 byte, o.s. reads 8k
- statd and lockd for locking

server parts

- nfsds, in quadruplets (4,8)
- mountd, weak authentication for remote mount
- portmapper, RPC uses “port mapping”
  - name service really, maps program numbers to (transport, port) pairs (both tcp/udp supported)
  - remote mount must contact portmapper to get port for mountd
  - portmapper is at well-known port 111
- /etc/exports, possibly export daemon

/etc/exports

- entries something like:

  ```
  /usr/bart   -access=heyman:dude,root=dude
  /usr/bob   -access=venus:flytrap
  ```

- if you change it, how do you notify mountd
  SunOs: /usr/etc/exportfs -a
  BSD: kill -HUP mountd.pid
- typically non-permitted root accesses are done as user “nobody”

plus utilities

- showmount - query remote or local mount daemon to see what is exported/what is mounted
- nfsstat/nfswatch - stats
- rpcinfo - look at portmapper setup, what is “mapped” in terms of programs
- spray - test capacity of nfs server, see if nfs packets are dropped (look at netstat -s)

protocol goals

- why UDP? and not TCP?
  - can support more clients if sockets not tied up in o.s.
- major goals: efficiency and statelessness
  - want to be able to reboot server after crash and have clients not have to remount/login
  - RPC calls are as idempotent as possible, i.e., call 2 should not depend on state of preceding call 1 (no open/read/close)
- interoperability

to achieve the goals

- use RPC protocol on top of UDP, request/response
  - » con: early versions were ping/pong protocols
- stateless handles are passed back to client from some RPC calls (surrogate of open) but don’t mean anything to client (mean something to server)
- UDP is fast too. For whatever reason, NFS has to compete with local disk access
  - » con: UDP checksums may not be done

mount protocol

- client mount command will contact server mount daemon for mount permission (and to get handle for remote volume)
- /usr/include/rpcsvc/mount.x on SunOs
- XDR for mount command:
  fhstatus
  MOUNTPROC_MNT(dirpath) = 1;

mount protocol commands (ops)

- MOUNTPROC_MNT - mount a dir
- MOUNTPROC_NULL - rpc ping
- MOUNTPROC_DUMP - list of mounts
- MOUNTPROC_UMNT - umount one
- MOUNTPROC_UMNTALL
- MOUNTPROC_EXPORT - tell exports
- showmount calls DUMP/EXPORT

NFS protocol commands (not all)

- NFSPROC_NULL - ping
- NFSPROC_GETATTR - stat(2)
- NFSPROC_SETATTR - chmod/chown(2)
- NFSPROC_LOOKUP(dirpargs) - “open”
- NFSPROC_READLINK - symlink contents
- NFSPROC_READ - read(2)
- NFSPROC_WRITE - write(2)

and more...

- NFSPROC_CREATE - create file
- NFSPROC_REMOVE - remove file
- NFSPROC_RENAME - mv file
- NFSPROC_LINK - create hard link
- NFSPROC_SYMLINK - BSD symlink creation
- NFSPROC_MKDIR - create directory
- NFSPROC_RMDIR - exterminate directory
- NFSPROC_READDIR - readdir(3)

a few data structures

```
struct fattr {
        ftype type;   /* file type */
        unsigned mode;   /* protection mode bits */
        unsigned nlink;    /* number of hard links */
        unsigned uid;       /* owner uid */
        unsigned gid;       /* owner gid */
        unsigned size;      /* size in bytes */
        unsigned blocksize;  /* preferred block size */
        unsigned rdev;     /* special device */
        etc...
        nfstime atime, mtime, ctime;  /* timestamps */
}
```

data structures...

```
struct sattr {  /* settable attributes */
        unsigned mode;
        unsigned uid, gid;
        unsigned size;
        nfstime atime;
        nfstime mtime;
}
struct readargs {
        nfs_fh file;     /* opaque 32-bit file handle */
        unsigned offset;   /* seek offset into file */
        unsigned count;   /* how much i/o */
        unsigned totalcount; /* total read count from this offset */
}
```

actual XDR for a few calls

```
diropres
NFSPROC_LOOKUP (diropargs) = 4;

readres
NFSPROC_READ (readargs) = 6;

diropres
NFSPROC_CREATE(createargs) = 9;
```

basic operation

- % cat file; i.e., open/read/close will be translated into some set of:
  - NFSPROC_LOOKUP() - client calls this for each link in pathname, gets directory or final link vnode (handle) back
  - NFSPROC_READ to read the file. The offset and handle is in every READ call
- note: no opens and closes

statelessness

- server keeps no state about client transactions
- clients know they did a mount - can do mount multiple times (crash/reboot), server doesn’t really care
- client doesn’t need mount if server crashes
- each request must completely describe operation.
- read is idempotent
- remove is not...

statelessness - con

- server must write block to disk immediately - no typical UNIX style write cache
- slows writes down
- some vendors can offer NVRAM to buffer blocks for better performance

retransmission/reliability

- at RPC level, NFS will retry
- client system call will by default “hang” until server reboots if no ack; calls are synchronous
- udp checksum/ethernet checksum is only csum. assumption is that client/server are fairly local
- BSD nfs/udp now has elements of TCP, slow start, etc.
- BSD nfs available over TCP. need a client that can do that.

file handles

- returned by lookup/create/mount
- used by read/write/readdir
- create on server, passed to client as “magic cookie”
- per server encoding of info server needs to find file; e.g.,
- UNIX: (device, inode number, nonce)
- non-UNIX server would use different semantics
- client cannot understand cookie, just use it

file handles

- lookup must be done label by label on client
- “namei” process on client UNIX system
  /a/b/c -> /, a, b, c is end
- consider if we mount
  mount server1:/usr/local   /usr/local
  mount server2:/usr/local/bin.mips usr/local/bin
- then client must lookup
  /usr/local/bin/ls and cross from server1 to server2

file handles can lose “freshness”

- file handle may become “stale” if client1 is using it (cat file)
- and client2 or server process removes file

mount options

- number of options given that affect basic operation, typically passed in at mount time
- rw/ro - readwrite or readonly volume
- bg - if mount fails, keep trying in background
- retrans/timeo - number of times to retransmit, with a given timeout per resend. timeout in 10’s of a second
- read/write basic buffer size.   default typically 8k (result is ip fragmentation)

hard/soft option

- hard/soft/”spongy”
- hard - client RPC call must hang in client kernel until completion. Process CANNOT interrupt call (say with signal)
- hard - emulates a missed disk interrupt and a dead disk; we hang until the server reboots
- soft - system call (read/write) is interruptible (emulates flakey local disk!)

hard/soft cont.

- if you are doing an ls, soft is ok
- if you are doing a cp, soft may not be ok
- you believe that all apps check read/write for errors and take corrective action?
- users get frustrated with having shells hung though because /usr/local/bin is on a crashed NFS server
- sun’s advice: hard for read/write, soft for readonly, many sites don’t pay attention

spongy? what the heck...

- on BSDI and OSF/1 systems, try and combile best of hard/soft
- hard except that stat/lookup/fsstat/readlink/readdir ops can return an error,
- so write NO, read YES, can possibly minimize NFS problems

other topics - automounter

- automounter
  - helps support large net installations
  - auto mounts file systems when needed and unmounts when not used for a while
  - “mostly transparent” to users, you have to know the name... you can’t cd there and do an ls
  - client-side “fake” server, intercepts request and mounts remote server
  - can support redundant file systems as well
  - “amd” better than sun’s product

other topics - security

- /etc/exports allows export of dir /something to system X
- as usual, only export what you need to export, don’t export everything
- security here is ip address security, subject to ip address spoofing
- secure RPC/kerberos other possibilities