SMB and NFS compared

SambaXP 2024
Somewhere in the internet

Volker Lendecke

SerNet / Samba Team

2024-04-18

Access paths to file systems

- Posix, NFS and SMB all give access to directories and files
- All three worlds serve different requirements with different historic backgrounds.
- Posix access goes through a local syscall interface
  - When the "server" (i.e. the kernel) dies, all clients are gone
  - When the "client" (i.e. a process) dies, the kernel immediately knows
  - Client $\leftrightarrow$ Server latency exists, but is extremely low
- NFS and SMB should not trip over the Fallacies of Distributed Computing (see wikipedia)
  - Everything between client and server is slow, either side and everything in between can fail or even lie.

Interoperability

- Different access paths to the same file system must coordinate
  - Local file systems provide Posix (or rather typically Linux) semantics
  - Linux NFS servers (knfsd/Ganesha/others) map these semantics for NFS protocol requirements
  - Same for Linux SMB servers such as ksmbd and smbd
- Other protocols like S3 also live in the same space
- Exposing protocol semantics in isolation is problem solved pretty well in both the FOSS as well as in the proprietary worlds
- Cross-Protocol semantics to my knowledge have never been addressed, at least not in “my bubble”, the FOSS world around Samba

How hard can it be?

- Why is it so hard?
  - Posix has its subtleties (for example how to properly fsync), but basics semantics are well-known to Linux developers
  - Both SMB and NFS are complex protocols with decades of history
  - Implementing either protocol is too much even for a single developer, so understanding and implementing both takes teams separate from each other.
- Why has this never been solved?
  - From a Samba perspective, nobody cared enough
  - Maybe the "NFS locking does not work" legend is either true or sufficiently sticky for users to not rely on locking at all

Areas of difference

- Security
  - NFS is usually machine-based, SMB sessions are per user
  - ACLs
- File name semantics
  - Case sensitive vs insensitive, special file names/characters
- File and directory metadata
  - Time stamps, xattrs, alternate data streams
- Request replay
- Locking
  - Share modes/reservations, byte range locks
- Client caching
  - Leases vs delegations

Security

- SMB had password protection of shares since the CORE protocol in the 1980s, users could not be distinguished initially
- With LANMAN 1.0 user login was added to the protocol, since then all SMB traffic is per user (machines can also be "users")
- The scope of a security context is the transport connection
- NFS relies on the underlying ONC-RPC for security
- Only with NFSv4.0 RPC security via GSSAPI is a requirement
- Scope of a security context in NFS is the individual request
  - NFS allows different RPC security settings per directory/file
- NFS protects locking state (open/share/delegation/brlock) separately from any other authentication on the transport
- A lot of NFS deployments run without any security

ACLs

- SMB has ACLs defined by the Windows security model and NTFS
  - Principals are Security Identifies
  - SIDs don't have a type such as user/group/machine etc.
  - 13 bits granting or denying specific types of access
- NFS deals with permission bits (RWXRWXRWX)
  - NFSv4 adds the 13 bits from the Windows doc plus 2 more (WRITE_RETENTION, WRITE_RETENTION_HOLD)
  - NFSv4.1 adds ACL inheritance flags
  - Deny ACEs and system acls supported
  - ACCESS request can only query 6 of the 15 bits
  - ACE principals are full UTF-8 strings
  - user@domain recommended, numeric string possible, no real mandatory standard

File and directory metadata

- Not much significant difference
- SMB has infolevels, NFS can query individual attributes
- Both have the typical time stamps, file size, etc
- Named extended attributes in both NFS and SMB
- SMB uses the : character for named streams
- NFS the OPENATTR - Open Named Attribute Directory
  - You read that right, NFS has alternate data streams!

SMB Request replay

- SMB runs over reliable transport
- Before SMB2 multichannel, there was one TCP connection per client and server machines: Replay of requests not an issue
- SMB2 Multichannel widens the transport to multiple connections
  - More performance, prerequisite for SMB over RDMA
  - "Plan B" for network disconnects
- Multichannel enables resending requests over a different connection
- Channel Sequence Number incremented on disconnects to indicate, Client indicates replay with a bit in the SMB2 header
- CreateFile has a CreateGUID to identify requests re-sent
- Locking calls detect replay with lock sequence numbers

NFS request replay

- UDP used to be a valid transport for NFS
- The ONC RPC Duplicate Request Cache is based on an opaque 32-bit XID (request ID)
  - Correct identification of clients is problematic
  - No mechanism to correctly throw away cache entries
- NFSv4.1 introduces proper DRC handling
  - CREATE_SESSION allocates an array of request slots on the server, holding sequence numbers.
  - Client chooses a slot number per RPC, sends its view of sequence number
  - Server validates sequence number increments, throws away cache entries when client sends sequence number incremented by one

Share Modes / Share Reservations

- SMB from the beginning was a stateful protocol
- Files have to be opened before use, locking was always possible
- For single-tasking MS-DOS compat reasons, per-open locks (share modes) protected client applications from each other
- NFS before v4 was designed as stateless
- Locking was done in external protocols, recovery from failures is still an area of concern
- NFSv4 identifies clients and servers and adds state to the protocol
- Recovery from client failure via leases, meaning regular client pings
- NFSv4 introduces share reservations to accomodate Win32 clients
- FILE_SHARE_READ and _WRITE are available, _DELETE is not
  - FILE_SHARE_DELETE is a lock on the name: Must be done at the directory layer

Byte Range Locks

- NFS models Posix, SMB models NTFS
- Overlapping locks handled differently from SMB
- Advisory and mandatory possible
- NFS READ can ask to override any mandatory locks

Locking state management

- In SMB, all locking state is tied to a file handle
  - Share modes and byte range logs are dropped when the file handle is closed
  - One operation to potentially wipe all locking state
- NFS has a separate name space for "open owner" and "lock owner" entities
- No clear file handle similar to a Posix FD exists
- Clients can implement their own "open/lock/delegation owner" name spaces independent of particular client processes or users
- Still unclear how exactly map those two concepts in a central server-side locking infrastructure

Client caching

- SMB1 allowed clients to cache via oplocks
  - Permission to handle requests locally per file handle
  - Oplocks can be broken, but not in-place upgraded
- SMB2 introduced leases
  - Separate key space that can be broken and upgraded while the files are kept open
- SMB2 allows to cache directory contents with directory leases
- NFSv4 has a similar concept with file delegations
  - Write delegations allow almost everything being cached on the client
- NFSv4.1 adds directory delegations, also allowing file change notify

Thanks for your attention

You have to implement an NFS server to understand the RFC

vl@samba.org / vl@sernet.de

https://www.sernet.de/

https://www.samba.org/