SMB and NFS compared

SDC 2024
Santa Clara

Volker Lendecke

SerNet / Samba Team

2024-09-16

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

- NFS serves Posix, SMB is for Windows, S3 is – well, S3.
- Different access paths to the same file system must coordinate
- Exposing protocol semantics in isolation is a 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 or how to deal with deleted files), but basic semantics are well-known to Linux developers
  - Both SMB and NFS are complex protocols with decades of history
  - Implementing either protocol is too much for a single developer, so understanding and implementing more than one takes teams separate from each other.
- Why has this never been solved?
  - From a Samba perspective, nobody cared enough
    - Windows users expect faithful semantics, the CEO's laptop must work
  - NFS users live with quirks forever, so semantic coherence seems not to be seen as a business opportunity

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 1980s
- 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
- NFSv4.0 introduced GSSAPI as 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
- NFSv3 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 (why??)
  - ACE principals are full UTF-8 strings
  - user@domain recommended, numeric string possible, no real mandatory standard

chown / chmod

- Posix stat information has permissions, owner and owning group as independent entities
- NFSv3 and v4 implement exactly this model
  - "owner" and "owner_group" are separate attributes
  - "mode" represents the permission bits
  - ACL implementation adds OWNER@, GROUP@ and EVERYONE@ entries
  - RFC8881 (NFSv4.1) section 6.4 handling both mode bits and acls pretty open
  - chmod MUST change the ACL, setfacl MUST change the mode
  - chown makes OWNER@ ACE to someone else
- SMB ACLs don't have OWNER@ and GROUP@
  - What to do with chown and chmod
  - Introduce OWNER@/GROUP@/EVERYONE@ special SIDs for interop?

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
  - Durable & higher handles make locking state survive
  - One operation to potentially wipe all locking state
- NFS has a separate names 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

How to cooperate

- Practial areas of concern: Idmapping and ACLs
  - These topics are ancient and don't go away
- **Identities:** Windows has AD and SIDs, Unix has 32-bit IDs
  - SIDs don't have a type (user/group/whatever) and can be used everywhere, they are unique worldwide
  - Unix IDs are just numbers, but their use (chown/chgrp) specifies the type, they overlap everywhere
- **ACLs:** Everybody has to implement their own
  - Relatively recent example: Amazon S3 with ACLs and then bucket policies
  - Microsoft extended ACLs with conditions (Can you put bucket policies there?)
- Who needs to talk to each other?
  - Kernel: ksmbd and knfsd people
  - User space: Samba and Ganesha
  - Testers/documenters: Define the expected behaviour

Thanks for your attention

You have to implement an NFS server to understand the RFC

Join the BoF Tue 8pm Cypress

vl@samba.org / vl@sernet.de

https://www.sernet.de/

https://www.samba.org/