Getting the Most Out of the Linux CIFS Client

Jeffrey Layton
Principal Software Engineer, Red Hat

SambaXP, April 10, 2013

Agenda

- Overview of Linux CIFS
- SMB Version Selection
- Authentication
- Cache Coherency
- Locking Coherency
- Read / Write Performance
- Access Control
- Identity Mapping

Overview of Linux CIFS

- The in-kernel filesystem that “speaks” SMB
- Samba-affiliated project, but it's not part of samba.
- Two main pieces:
  - kernel filesystem (aka cifs.ko)
    - Most of the code
    - Distributed as part of the Linux kernel
  - cifs-utils
    - Small package of user space utilities
    - Mount helpers and kernel callout programs

Mounting a CIFS Filesystem

- Typical mount syntax:

```
# mount //server.example.com/share \
          /mnt/cifs -o <options>
```

- cifs.ko supports a lot of options, most of which are documented in mount.cifs(8).
- Strive to allow for sensible defaults, but it's not always possible

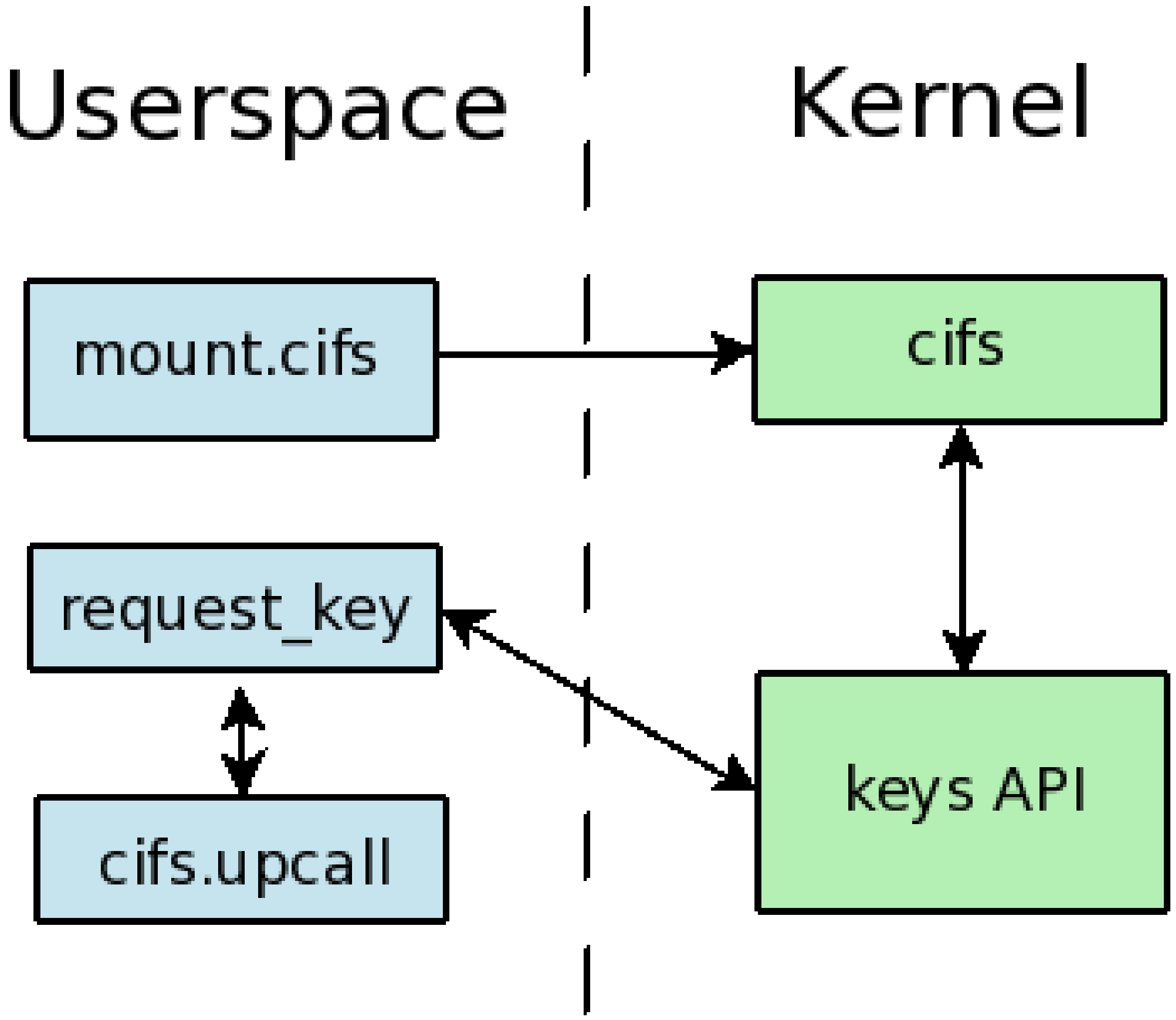
SMB Version Selection

- cifs.ko “speaks” SMB1 by default
- As of kernel v3.5, cifs.ko supports SMBv2.1 as an experimental feature
- Also nominal support for 2.002 and 3.0, but the implementations are not quite as robust
- Use the “vers=” mount option to select
- Not much advantage currently to using versions higher than 1.0. (2.x needs support for multicredit requests and compounds)

Authentication

- sec=ntlm – (default pre-v3.7)
- sec=ntlmssp – (default post v3.7)
- sec=ntlmv2 (not well supported by servers)
- sec=none: Anonymous auth
- sec=krb5: Kerberos (requires upcall)
- Append “i” (for “integrity”) to mandate signed sessions

Kerberos mount upcall process



Kerberos Mount Credentials

- Ticket used to handle the SMBs generated by mount() syscall
- Governed by "`-o cruid`" option. (In **really** old kernels, by the `uid=` option too, but that was a bug)
- These creds are always associated with the root user
- In single-user mount, all SMBs use these credentials regardless of who is accessing mount
- In multiuser mount, only root uses these creds
- Also possible to use keytab, use -o username= to tell it what principal name to use

Cache Coherency

- **cache=** option
  - **loose**: uses NFSv3-like semantics. Only invalidate caches when it appears that something has changed.
  - **none**: pagecache is not used. All reads and writes are done through to the server. Much like an implicit O_DIRECT. (aka **forcedirectio** on pre-v3.5 kernels)
  - **strict**: stricter adherence to the protocol. Only use the cache when an oplock is held. (aka **strictcache** on pre-v3.5 kernels)
- Default in v3.7+ is **strict**, but in older kernels it was **loose**.

FSCache

- Allows you to cache data on local disk for (sometimes) faster access...and less load on server.
- Fundamentally at odds with the standard CIFS cache coherency model after a reboot
- Can also act as “swap for pagecache” -- alternate way to get to data instead of going to server
- Only persistent across client reboots with **cache=loose**

Byte Range Locking Coherency

- Windows and POSIX locks don't mix well
- Windows locks are mandatory. Read and write requests are blocked when there is a lock held.
- POSIX locks are advisory. Only block other lock requests.
- POSIX byte range locks can split and merge.
- Windows locks “stack”
- Windows/Linux apps locking same files are very problematic
- Client caches locks when it has an oplock
- Best to avoid applications that do complex locking

Write Performance

- v3.0+ support async writes for cached requests
  - Cached I/O is page-aligned
- V3.4+ for uncached requests
- Max wsize is also larger in v3.0+
- wsize= option is a starting point for negotiation
- SMB2 capped at 64k (no multicredit support)

	non-POSIX	POSIX
Default	64k	1M
Maximum	128k	16M

Read Performance

- v3.2+ support async and larger reads into cache
  - Page aligned I/Os due to doing through pagecache
- v3.5+ has support for uncached reads
- On 32-bit arch, you want v3.7+ (removed artificial serialization to handle mapping highmem pages)
- SMB2 capped at 64k (no multicredit support).

	non-POSIX	POSIX
Default	60k	1M
Maximum	128k	16M

Socket options

- Can direct the TCP port to use with port= option
  - By default, client tries to use port 445 first and then falls back to 139.
  - Avoid port 139 if possible (kernel treats that port as implicitly needing NetBIOS encapsulation), which adds some overhead.
- Undocumented sockopt= option is now deprecated in v3.8.
  - TCP_NODELAY was only one implemented, and never proved to have any effect in most cases.
  - Kernel now corks/uncorks the socket when sending.

POSIX Extensions

- Allows the “tunneling” of POSIX semantics over CIFS protocol:
  - Larger read/write sizes on SMB1
  - Ownership/Permissions info in stat() calls
  - POSIX opens/creates
  - POSIX locks
- Enabled by default if server supports them
- Can be manually disabled with “-o nounix”

Ownership and Permissions

- Horribly confusing with cifs.ko
- 2 **separate** elements
  - Display: concerns result of stat() type calls
  - Enforcement: permissions checking
- Basic principles:
  - Client displays permissions according to what it knows, and will **sometimes** enforce those permissions vs. local users
  - Permissions are **always** enforced by server, according to the credentials used

Ownership and Permissions Display

- With POSIX extensions:
  - ownership/mode displayed as they appear on server
  - Might not make sense if UID/GIDs are mapped differently!

- Without POSIX extensions:
  - UID/GID set to default owner on the mount.
  - Mode set to file_mode/dir_mode mount options.

cifsacl

- When updating inode info:
  - Opens file
  - Reads ACL and maps it to mode bits
  - Maps SIDs to UID/GID (possible upcall)
  - Closes file
- Should also use `cifs.idmap` program to translate SID <-> UID/GID.
- Mapping ACL to/from mode bits is lossy translation
- Horrible for performance (a lot of extra round trips to server, breaks oplocks, etc.)

Ownership and Permissions Enforcement

- Historically, cifs.ko mounts used single set of creds
- All accesses used the mount credentials regardless of who was accessing the mount
- Client tries to enforce permissions according to local inode permissions, which are typically default ownership/permissions of the mount.
- Server will **always** enforce permissions based on mount credentials
- Result is a union of mount creds enforced by server and the local permissions enforced by client.

Multiuser Mounts

- If multiple users are accessing the share, consider multiuser mount (`-o multiuser`)
- Allows users to access the mount with their own credentials:
  - When non-root user accesses mount, build a new authenticated session to server based on UID
  - Torn down after period of inactivity
- Currently, all authentication type must be same for all users (mount with krb5, need krb5 auth for all users)

Multiuser Mounts (cont'd)

- With sec=krb5, just need a valid credcache (or keytab)
- With password-based auth methods, use the cifscreds program to stuff creds in keyring (v3.3+)
- Client-side permissions enforcement is disabled
- Without POSIX extensions or cifsacl, ownership is displayed as current accessing user

Ownership and Permissions Mount Options

- These set the “default” ownership/permissions for mount:
  - `uid=`
  - `gid=`
  - `file_mode=`
  - `dir_mode=`
- Force client to disregard ownership presented by server:
  - `forceuid`
  - `forcegid`

Ownership and Permissions Mount Options (cont'd)

- Disable client-side permissions checking entirely:
  - `noperm`
  - `multiuser`

Backup Intent

- MS permissions model allows “backup programs” to access filesystem with extra permissions.
- Program must open files with extra flag (CREATE_OPEN_BACKUP_INTENT).
- Lookups/readdir need CIFS_SEARCH_BACKUP_SEARCH flag.
- Set these on a per-cred basis via:
  - `backupuid=`
  - `backupgid=`

ACL setting and querying

- Can query/set ACLs using these commands:
    - `getcifsacl`
    - `setcifsacl`
- These use a special xattr to read/write raw ACL info
- In the future, may eventually be superseded by RichACL xattrs
- Mounting with -o cifsacl can change ACLs via mode-bit translation too (not recommended!)

Identity Mapping

- The CIFS wire protocols deal with **security identifiers** (SIDs)
- POSIX uses UIDs and GIDs
- cifs-utils has tools that map between them:
  - `getcifsacl`
  - `setcifsacl`
  - `cifs.idmap`
- Uses winbind to do the mapping
- New plugin interface in cifs-utils v5.9 (see cifsidmap.h for interface docs)
- sssd plugin is in the works

Home Directory Access

- Common usage pattern is to set up access to home directories via cifs.ko, typically using krb5 auth
  - Set up the credentials you want to use for root in keytab:

```
kadmin: addprinc -randkey nobody
Principal "nobody@EXAMPLE.COM" created.
```

  - Add that to keytab:

```
kadmin: ktadd nobody@EXAMPLE.COM
```

  - Set up mount in /etc/fstab:

```
//server.example.com/home /home cifs
sec=krb5,username=nobody,multiuser
```

Home Directory Access (notes)

- Generally want to “squash” root access to some non-privileged user on server.
- User can be **very** limited, but does need access to fetch attributes of the root of the share and any intermediate directories down to the root of mount
- Accesses by other users are done according to the credentials in their credcaches.
- Unauthenticated users get back EACCES on most syscalls

Questions?