Jeremy Allison
Samba Team

Future extensions to CIFS (CIFS to the desktop)

(or the death of NFS :-)

Novell

Why do we need extensions to CIFS ?

Novell

The purpose of CIFS extensions

- CIFS is the dominant desktop file sharing protocol.
- Most IT departments don't want more than one file sharing protocol to troubleshoot.
  - Most IT departments don't want to add new client code to Windows.
- In order to enter the desktop world, new desktops must live within a CIFS-only world.
  - NFS, even NFSv4, will not gain any traction on the desktop.

The purpose of CIFS extensions

- Linux and MacOS X desktops are the only viable competitors to Microsoft clients.
- Extending CIFS can provide value-add differentiators for CIFS server vendors.
- Creating a “standard” set of extensions can prevent fragmentation in the CIFS vendor marketplace.
  - Vendors can compete on quality and performance, rather than non-interoperable variants

The history of CIFS extensions

Novell

The history of CIFS extensions

- Early attempts to extend CIFS were in the OS/2 and early UNIX authentication documents.
  - Part of the OpenGroup specifications.
  - Mechanism specified for using UNIX password hashes.
- Next Thursby added new TRANS2 calls to cope with MacOS 9 resource forks and desktop database.
  - Reserved space between 0x300 and 0x399 in the TRANS2 space.
  - Only specification available seems to be an old Samba contribution (GPL).

The history of CIFS extensions

- First serious non-Microsoft changes were from the original (non-insane) SCO, then HP with the UNIX extensions document (1997-2000).
  - Created from discussions on a mailing list about what would be required for UNIX to UNIX CIFS.
- A milestone was an agreement from Microsoft to carve out an extension space for CIFS !
  - After the initial CIFS UNIX capability bit was used by Microsoft for “extended security”.

The history of CIFS extensions

- SNIA document included documentation of HP UNIX extensions, but this document is not usable.
  - Conditions preclude use of the SNIA document for any commercial purpose (explicitly stated).
  - Check out the original (Microsoft Word!) document on the Web instead.
  - The Samba server adopted the UNIX extensions in the 2.2.x series, but not seriously maintained until CIFS Linux client was adopted into the 2.6 kernel.

Current CIFS extensions

Novell

What are the current CIFS extensions ?

- The original intent was to create a dialect of CIFS that allows full UNIX to UNIX semantics.
- This meant allowing a diskless UNIX client workstation to remote-boot from a CIFS server.
- Client detects the presence of UNIX extensions in a bit (0x800000) in a NT negprot reply.
  - Client is then free to use a new set of TRANS2 calls, between 0x200-0x2FF.

What are the current CIFS extensions ?

- Most obvious changes were the addition of a UNIX_FILE_BASIC struct containing the UNIX-specific data not found in a CIFS directory entry.
  - TRANS2_SET/GET_FILE_INFO calls use this to set and query UNIX info.
- In addition, TRANS2 info levels to support UNIX symlink and hard links were specified.
  - NT_RENAME call can also create hard links, used for the NT POSIX subsystem.

What are the current CIFS extensions ?

- Some problems with this original spec, no on-the-wire mappings were specified for such things as UNIX permissions.
  - No block size was specified for the “number of blocks” returns in the UNIX_BASIC_INFO.
  - Somewhat HPUX-on-the-wire specific.
- After some review an “extension version” request was added, which returns a capabilities set for future expansion.

Difficulties in interpretation

- Symlinks present a particular problem for CIFS extensions.
- Allowing arbitrary target paths on a “create symlink” may allow Windows clients to break out of a share-specific area of the filesystem.
  - Due to server resolution of symlinks on Windows client lookup.
  - NFS clients don't suffer from this as symlink look-ups are client side only.
  - Vendor specific changes (Microsoft SFU product uses Extended Attributes to store symlinks).

Current issues - POSIX compliance

- Unix Extensions can't support full POSIX compliance due to differences in byte range locking semantics.
  - Do we want to implement POSIX locking ?
  - Compatible subset required.
- Renaming of open files also not supported by CIFS due to deny mode semantics.
- POSIX ACLs were needed. Capability bit already defined.
  - Simple GET/SET calls were sufficient, ignore modify race conditions.

Current issues - case sensitivity

- CIFS already has a “case insensitive” flag bit available in the standard protocol header.
  - Ideal situation would be UNIX clients turn this bit off.
  - Problem is earlier Microsoft clients (pre-Windows NT) don't bother to set this bit.
  - Samba auto-detects client type to determine if this bit should be obeyed.
- Windows file servers inconsistently obey this bit (Windows 2000 does, Windows 2003 needs a registry change).

Current issues – user and group identity.

- UNIX extensions currently can return a uid or gid that only has meaning on the server.
  - Similar issue to NFS, user and group databases are expected to be consistant over clients/servers.
- CIFS has traditionally specified user and group lookup functions.
  - CIFS takes a kitchen sink approach to solving file sharing issues. Such extra functionality could be added into the UNIX version of CIFS.

Current issues - character mapping.

- Several characters valid in UNIX filenames are invalid in CIFS filenames.
- These are :

  : < > | ? \ *

- CIFSFS and Windows (services for UNIX) map these into the “user defined” UNICODE space, by prefixing them with 0xFF.
- Samba server doesn't currently support this (until I get back from this conference).

Current state of the UNIX extensions

- POSIX ACLs GET/SET calls recently (Samba 3.0.x) added.
  - Developed in conbination with Steve French's Linux CIFSFS Client code.
  - Test code to getfacl included in smbclient.
  - No set code in smbclient as I didn't want to write the parse code for POSIX ACL semantics.
- Uses UIDs/GIDs on the wire, don't mix up SIDs with POSIX style calls.

Current state of the UNIX extensions

- Makes CIFS UNIX file sharing closer to SVR3 RFS than NFS.
  - Although NFSv4 is re-inventing many of the same techniques.
- Similar to NFS in that device files are not remoted, some operations are still client-side look up.
  - Symlink handling
  - Device file accesses.
- Single or multiple TCP socket connections, variety of ways to multiplex user connections.

Available Client implementations - Linux.

- Steve French is maintaining the Linux CIFSFS client for the 2.6 kernel.
- Closest match to Samba server code as they are developed together.
  - Sometimes client code comes first, sometimes server.
  - Use identical header file definitions.
- Tested with SPECFS filesystem test tools.
  - Currently passes POSIX filesystem tests with the exception of file locking.
  - Steve is currently working on performance.

Available Client implementations - MacOS X.

- Conrad Minshal at Apple is developing the MacOS X CIFS client.
- Tested mainly against Windows servers (Apple focused mainly on the desktop market).
- MacOS X is chosing Windows style ACLs rather than POSIX.
  - No current plans to add POSIX ACL extensions.
- Source code published in Darwin.
- Apple reluctant to participate in community development.

Available Client implementations - HPUX

- Eric Raeburn is developing the SHARITY NFS to CIFS gateway client on HPUX.
- Proprietary code, although works closely with Samba.
- Active participants in UNIX extensions process.
- Not currently using ACL extensions or EA's.
  - No EA's in NFS
  - No standard POSIX ACL protocol in (HPUX) NFS.
- HPUX only solution.

Where do we go from here ?

Future CIFS extensions

- Add POSIX file locking.
  - Should conflict with CIFS locks but keep locked ranges separate.
  - Allow lock range split/merge to obey POSIX specs.
  - File access should ignore POSIX locks (advisory only).
  - Lock owner call (getlock) is needed.
- UNIX (Linux) style EA's (extended attributes) not supported yet, only case insensitive DOS style EA's.
  - No EA standard in UNIX. Current implementations are Linux, SGI Irix, FreeBSD.

Future CIFS extensions

- Add POSIX open()/mkdir() calls.
  - Return current attributes.
  - Saves round trip to look up after open.
- Add POSIX rename() and unlink() calls, allowing POSIX semantics.
  - rename() should allow rename of open files.
  - unlink() should allow deletion of open files.
  - Should we take share mode into account here ?
- Try not to make this too Linux specific.
- At what point is this not CIFS anymore ?
  - UNIX errno returns ?

Future CIFS extensions

- Create a new UNIX NSS interface named pipe (\\UNIX_NSS ?).
  - This will allow clients to completely forward uid/gid to name translation to a file server, allowing a consistant name space.
  - Allows one machine authentication (probably krb5) to control access to all name services.
  - On the wire specification probably based on a NSS call linearization.
  - Allow multiple uid/gid -> name, name-> uid/gid lookups for efficiency.

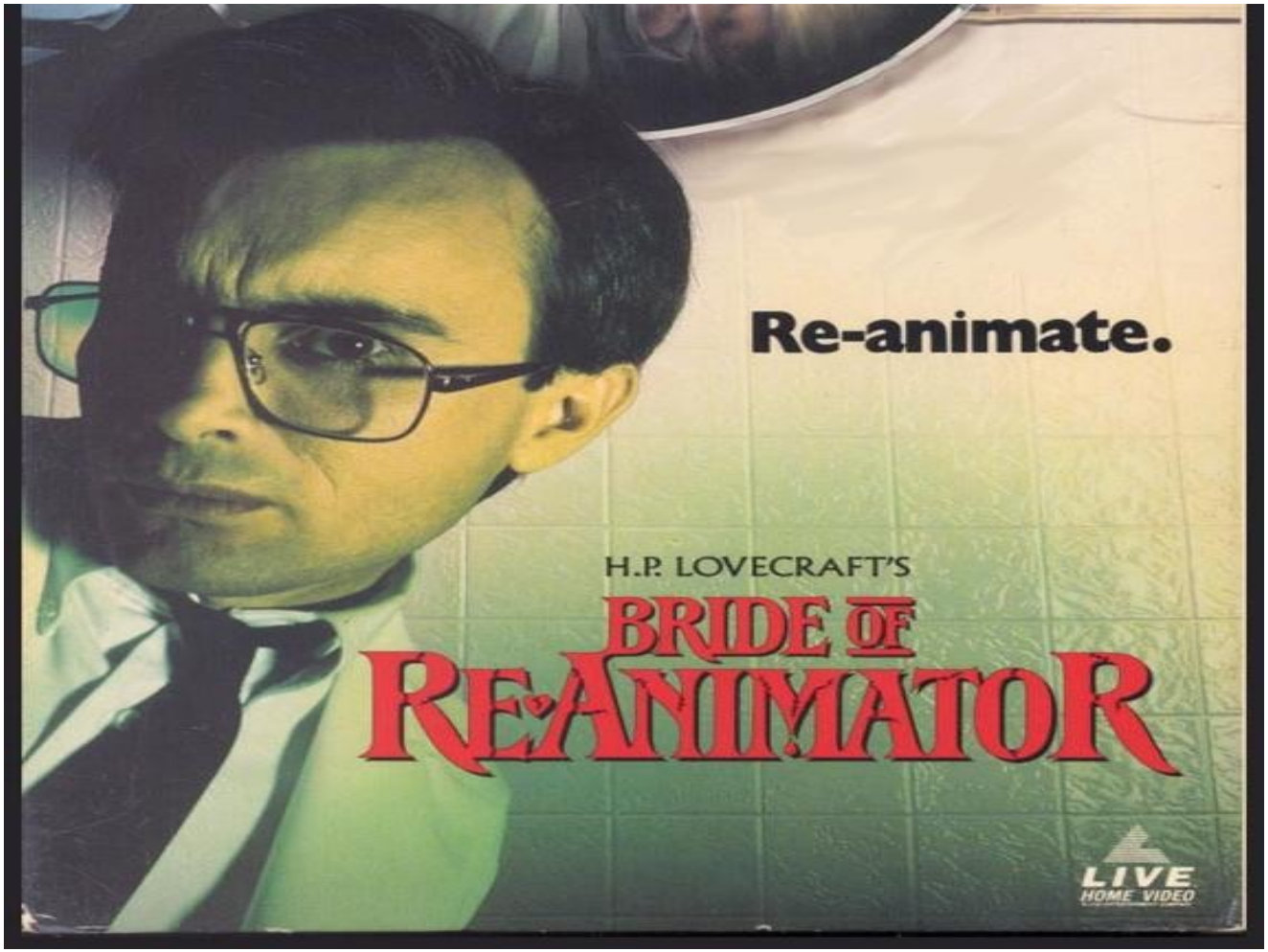
Feature enhancements - encrypted CIFS

- NFSv4 has this, so we need it too ☺.
- Bootstap encryption using the krb5 session key, also used for SMB signing.
  - We need a way to request re-keying within a long lived session (new TRANS2 call ?).
  - Hard to add new error codes, so have a counter giving the number of packets this session key is valid for.
- We need security review of any protocol we invent.

Integrating Windows clients into extended CIFS

- No one wants to add new Windows client code.
  - Definitely a “hack” solution for customers needing encrypted transport, not a mass market solution.
- Investigate using Windows client AFS code to create a CIFS to CIFS proxy, although this is not a high priority for the Samba developers.
- We are attempting to hijack this protocol. This is our only chance....

Far from being the death of CIFS....



Questions and Comments

Novell