SAMBA Opening Windows to a Wider World

Understanding Filenames in the Samba File Server

Jeremy Allison
Samba Team
jra@samba.org
jra@google.com

What is a filename ?

- Not as simple a question as it seems (streams..).
  - Reference to a server object the client wants to access or create. Separated into directory components.
- The contents of a filename have evolved over the life of the SMB protocols from versions earlier than SMB1 (CORE, CORE+ etc.) to the modern SMB3+ version.
  - As SMB1 and previous protocols are deprecated, I will avoid covering these details except where required.
- On the wire a filename is a UTF-16 string, length specified (not null terminated).
- On the server it's a nul terminated bag of bytes.
  - Only disallowed characters are '/' and '\0'.

How does smbd see filenames ?

- Internally and calling the VFS interface smbd uses:

```
struct smb_filename {
        char *base_name;  // Filename on disk
        char *stream_name; // Non-null if a stream name
        char *original_lcomp; // rarely used.
        uint32_t flags; // Internal name attributes.
        SMB_STRUCT_STAT st; // Valid if link count != 0
};
```

- So how to go from UTF-16 → struct smb_filename ?

First – copy off the wire

- convert_string_talloc() uses iconv() internally to convert from UTF-16 → UTF8 (default unix character set).
  - Will error out if the UTF-16 is not well formed.
- Creates a nul-terminated string in UTF8 format for further processing.
  - For create operations on IPC$ or printer shares, we're done.
  - Name is passed directly to pipe and printer functions as it's not used for any filesystem access.

Part #2 – DFS normalization

- On the wire names can be DFS pathnames.
  - Detected by a flag bit in the SMB2 request.
- DFS names are: \hostname\sharename\path
- DFS names are parsed by:
  - resolve_dfspath_wcard() → dfs_redirect() → parse_dfs_path()
- If this ends up as a self-referral, or any other means of getting to the same server, the ‘path’ component of the DFS name is treated as passed in pathname, and continues as..

Part #3 – Name Sanitization

- check_path_syntax() is the function responsible to ensuring a passed in name is well formed.
  - Allows both '/' and '\' as input path separators → '/' out.
  - Uses the next_codepoint() function to walk through the path, one UTF-8 character at a time, keeping needed state.
  - Copes with '.' and '..' components, moving the current component pointer as needed.
  - Copes with stream name weirdness.
  - Copes with wildcards in last name component (SMB1).
- If it returns NT_STATUS_OK we have a well-formed pathname, relative to base of share.

Side Note: Snapshot pathnames

- In SMB2, snapshots are specified by a create context (TWRP).
- In SMB1, snapshots are embedded into the pathname:
  - a\snapshot\@GMT-YYYY-MM-DD-HH-MM-SS\path
  - @GMT- can be anywhere in the path.
- Smbd canonicalizes this by rewriting all such pathnames inside canonicalize_snapshot_path() as:
  - @GMT-YYYY-MM-DD-HH-MM-SS/path/name
- Pre-SMB2 code, so TWRP contexts are written as the start of path.
  - Must be done *after* DFS normalization.

Fun with $cwd (current working directory)

- In UNIX $cwd is per-process. In smbd we change $cwd to be the root of the current share we are working on, read from the TreeID field of the SMB2 packet.
  - All pathnames sent over the wire are assumed to be relative to this base $cwd.
- On Linux, $cwd can be per-thread, which allows pthreads to do pathname processing.
- On all other UNIXs, $cwd is per process so pthreads cannot make any pathname processing calls.
- Samba implementation of @GMT- names also complicates this, as do symlinks.

The Gory Details – Part 1

- filename_convert() → unix_convert() is where the magic happens.
- unix_convert() takes a set of flags (UCF_XXX) that can control behavior (POSIX pathnames for UNIX extensions etc.)
  - First canonicalize any snapshot @GMT- paths: canonicalize_snapshot_path().
  - Canonicalize case if requested from smb.conf (allows case normalization to simulate case-insensitivity on case sensitive filesystems).
  - Save off last component of incoming path if requested by flag (original_lcomp field in smb_filename).

The Gory Details – Part 2

- Strip off and save any :stream_name – we don't use this to check the base path, but will need it later to check for stream existence.
- Does the name exist in the stat cache ? If so, done.
  - Common case – non-wildcard name:
    - Do a Samba VFS_STAT call on the base path. If it succeeds we are (mostly) done – hurrah !
    - Store the successful name into the stat cache.
  - Stream check - build_stream_path():
    - Restore the stream name and make sure that VFS_STAT on the stream name succeeds.
    - If not, search the stream on the file for a name match, replace stream name if found.
    - If stream not found, just set stat invalid and return OK.

The stat cache

- An optimization, but a very useful one.
- Keeps an in-memory (memcache) mapping of an uppercased version of a pathname → pathname as exists on disk.
  - If we find a match, we avoid pathname walks.
- On lookup, a VFS_STAT call is done to confirm the mapping is still valid.
- Partial name mappings can be done: eg – client sends ‘a/silly/name’, statcache contains ‘A/SILLY → A/Silly’, return from statcache will be ‘A/Silly/name’.
  - Only remaining components need searching.

Meanwhile, back in unix_convert()..

- VFS_STAT failed, we have to search.
  - Remember, any component of the name may be the wrong case, so we need to walk down the directory tree, one component at a time.
- We have to cope with mangling (later slide – ignore for now).
- Walk down the path, cutting at each '/' character.
  - Call get_real_filename() to try and figure out if the last component exists in the containing directory.
- Special code to handle "file dropbox" case, where we can't list a directory (UCF_PREP_CREATEFILE).
  - If we get EACCES on directory search, continue and hope for the best.

get_real_filename() - slow walk

- Tries to discover the correct case of the last component of the passed in path.
- This is so painful that an optimization function was added to the VFS to short-cut this for OEM filesystems that can quickly look this up.
  - VFS_GET_REAL_FILENAME.
- Without it, we have to open the directory and scan until we find a case-insensitive match, or reach the end of the directory.
  - Directories with 100,000+ files make this **slow**.

Mangling madness: mangle_is_mangled() and friends

- SMB3+ has more restrictions on filenames than UNIX.
  - Names like ‘afile:name’, ‘aux:’, ‘con:’, ‘A strange\\ path’ are allowed on UNIX, but not over SMB.
  - Pre-SMB2 Samba returns these as DOS ‘8.3’ names.
- This code was re-used to allow SMB2+ UNIX names to be reported to the client.
  - Works, but really ugly. Once we ditch SMB1, we could revisit mapping the disallowed characters to the UTF-16 private set.
- Inside unix_convert() we need to demangle each component as we come across it, complicating the walk code (component changes size).

Final steps

- Once we've done the pathname walk to success we return the converted smb_filename.
  - If the target exists, we return a valid stat struct so the caller has all the info to do locking around dev/ino.
  - If we're trying to create a file/directory/stream, the last component may not exist – stat.st_nlinks == 0 but we still return success.
- check_name() → Converted path is checked against veto (deny) lists, then validated using VFS_REALPATH call inside check_reduced_name() to ensure it's under exported share path.
- We end up with a validated pathname as an smb_filename struct, relative to the share export path.

UNIX Extension pathnames

- SMB1 (and SMB3+ in my experimental tree) pathnames coming in as UNIX pathnames are flagged with a UCF_POSIX_PATHNAMES flag in unix_convert().
  - Turns off many of the NTFS features, name restrictions, streams, case insensitivity, mangling etc.
  - In hindsight, this should be a separate codepath (unix_convert_posix()).
  - Complicates the logic inside unix_convert(), makes it harder to understand.
  - Returned UNIX names have an smb_filename flag.
- Remember to always use VFS_LSTAT for UNIX extension names (clients should follow symlinks).

Working with pathnames in the VFS

- Easier now VFS only takes smb_filename structs.
  - Still too much complexity pushed onto VFS authors.
  - Vendors with custom Samba can now add fields to the filename struct.
- VFS modules still need to know the details of smbd pathname processing.
  - Don’t assume all pathnames are relative to root of share, due to symlink race hardening we’re using more $cwd + last component paths.
- Smbd has almost any pathname utility function you might need (getting parent name etc.).
  - Before adding helper code, see if we already have it.

Future plans

- Get SMB3+ UNIX extensions upstream – will eliminate any server symlink race issue.
  - Refactor out the posix_pathnames code.
- Remove SMB1 fileserver code.
  - How to keep SMB1 client code tested ? Do we care ?
- Add TWRP (snapshot) field into smb_filename, remove @GMT- parsing.
- Break the $cwd == root of share assumption.
  - Make the code cope with arbitrary $cwd, partially there.
- Move towards using the pre-thread $cwd abilities in Linux (will FreeBSD have this ?).

Conclusions

- Filename processing is too complex, and greatly needs simplification.
  - Too much is done in too few functions, the code needs serious refactoring.
- Most of the complexity comes from old SMB1 and prior protocol semantics (having to deal with wildcards etc.) and could be greatly simplified by removing the SMB1 code.
- Clients and the protocol bear some blame, forcing clients to walk a path one component at a time would greatly simplify server pathname processing.
- Symlinks and streams are a *disaster* in security and complexity terms.

Questions and Comments ?

Email: jra@samba.org
Email: jra@google.com