SMB 2.1 & SMB 3

Protocol features, Status, Architecture, Implementation

Gordon Ross <gwr@nexenta.com>
Nexenta Systems, Inc.

illumos day 2014

Overview of this presentation

- Summary of SMB versions and what do we support?
- SMB Server Design changes
- Development approach
- Lessons learned
- Next steps
- Links, more info

Summary of SMB versions

SMB 1 … 2.0 … 2.1 … 3.0 …
What does each bring to SMB?
What do we support?

Summary of SMB versions: SMB 1

SMB1 vs. … (Why a new SMB version?)

- SMB1 has become very complex
  - Several variants of many operations, many used only in "fall back" paths
  - Need for backward-compatibility increased compliexity
- A "Heinz 57 variety" of calls, from several generations of clients spanning DOS 3.x WfW through today (Windows, Mac, etc.)
- Limited performance
  - small MTU, misaligned data, limited parallelism
  - “chatty” (many round trips required)
- Limited scale (16-bit IDs)
- Difficult to document or test.

Summary of SMB versions: SMB 2.0

What's in SMB 2.0? (vs SMB 1)

- Far simpler protocol (18 commands vs "Heinz 57")
- Improved scalability (larger ID sizes)
- Improved network utilization
  - Compound requests (fewer round trips)
  - Connection per user
  - Allows larger I/O requests *
  - Client caching of files/dirs *
- Durable handles *
- Symbolic links *

* optional features, not currently implemented

Summary of SMB versions: SMB 2.1

What's in SMB 2.1 (vs SMB 2.0)

- Previous versions
  exposes ZFS snapshots via SMB
- Large MTU support *
  also enables a per-client credit system
- oplock leasing model *
  cache delegations per object (vs per handle)
- clients can sleep in more situations
  (where they can reclaim handles after resume)

* optional features, not currently implemented

Summary of SMB versions: SMB 3.0

- What's in SMB 3.0 (vs SMB 2.1)
- New signing algorithm (AES-256-CBC)
- Optional SMB 3.0 features:
  - SMB Transparent Failover *
  - SMB Scale Out *
  - SMB Multichannel *
  - SMB Direct (RDMA) *
  - SMB Encryption *
  - VSS for SMB file shares *
  - SMB Directory Leasing *
  - SMB PowerShell *

* optional features, not currently implemented

For more details, see: MS KB 2709568

Current Status of SMB support

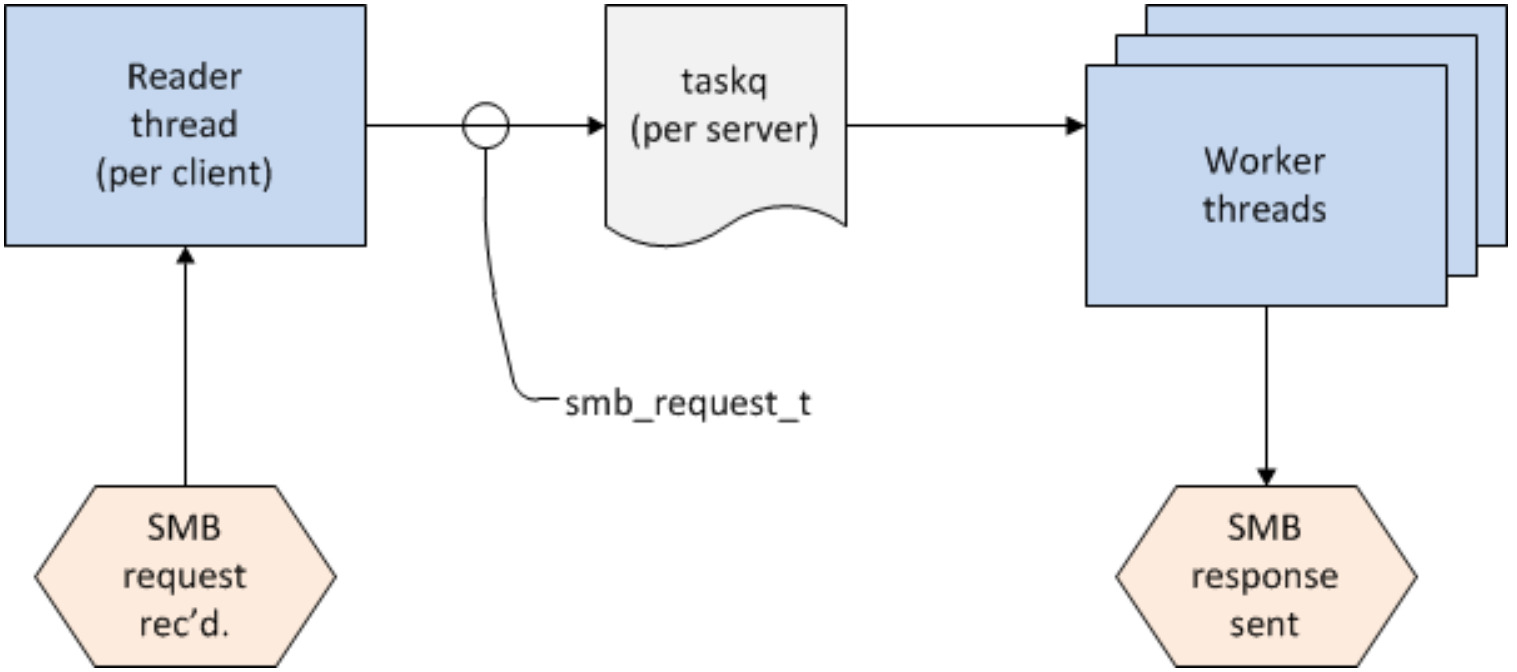
What SMB versions and features do we support?

- Available now, in NexentaStor 4.0.3
  - SMB 2.1 with all *required* features
    (some "optional" features still to come)
  - AD-member improvements (new DC locator)
  - Extended security, including Kerberos
- Prototype stage:
  - SMB 3.0 protocol level (no optional features)
- Under development, or planned:
  - SMB 2.1 optional features: Large MTU, durable handles…
  - SMB 3.0 optional features: Multi-channel, server-side copy
- When does this arrive in illumos?
  - later, see "next steps"

SMB Sever Design Changes

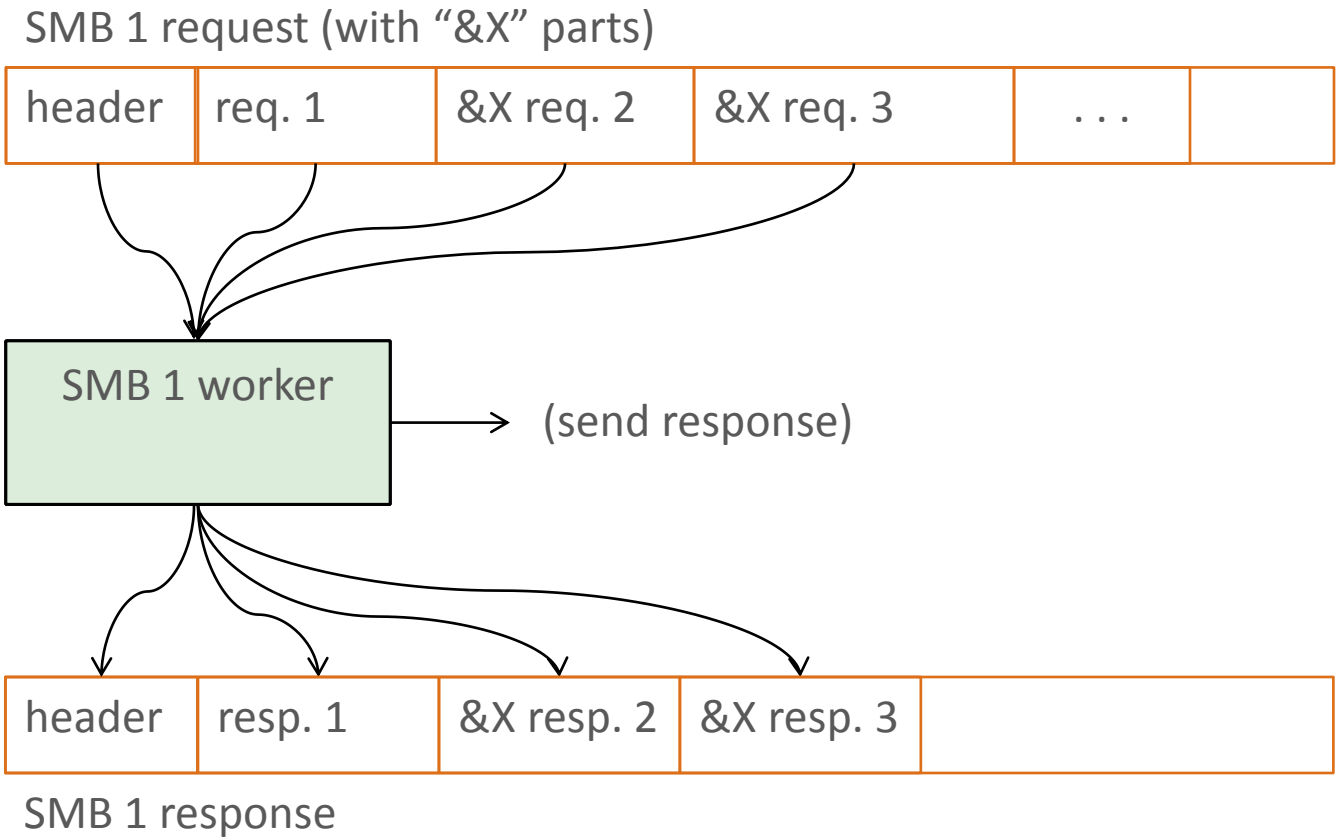
Existing multi-thread design
Message processing models
Challenges, approach

SMB Server: existing multi-thread design

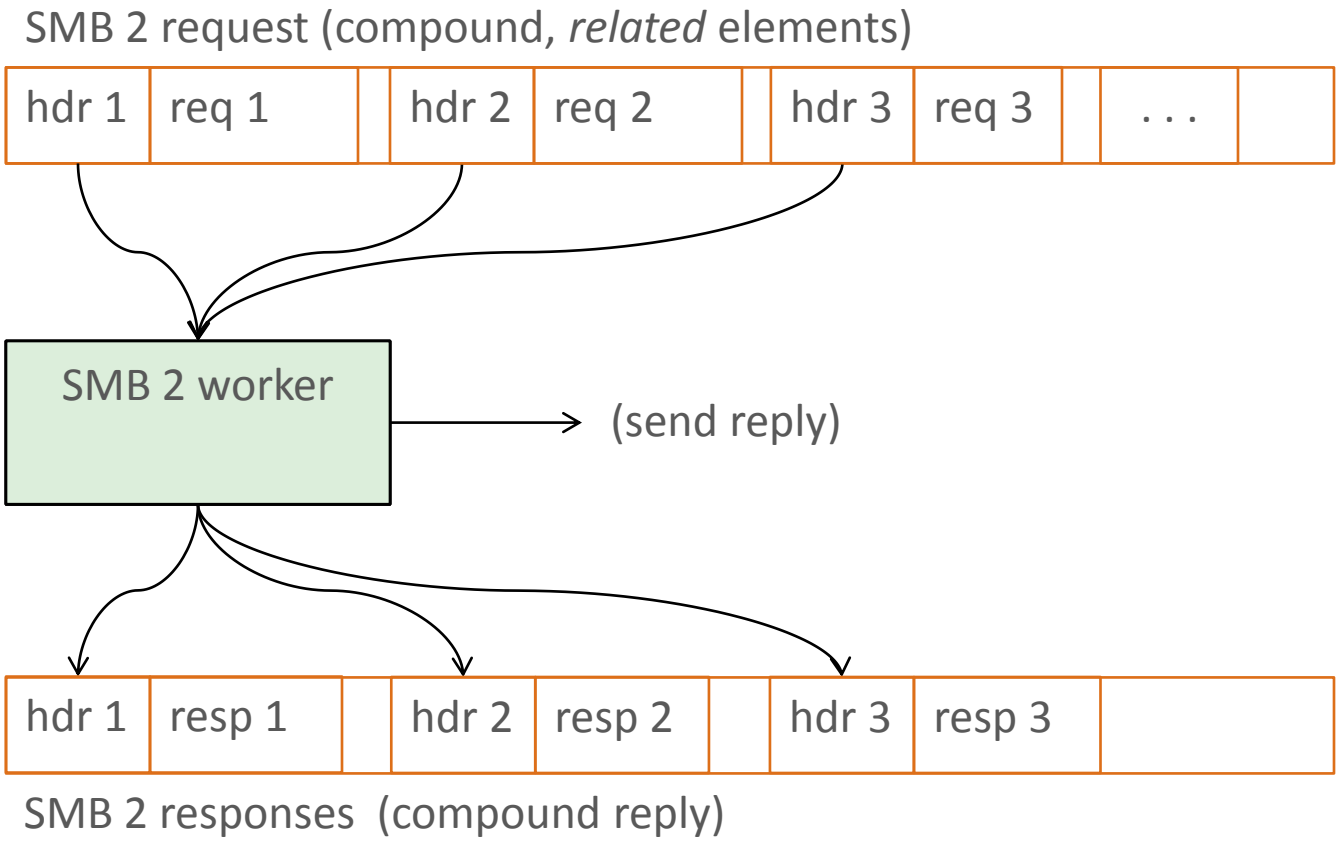


Relatively simple threads & locking model:
one thread per smb_request_t

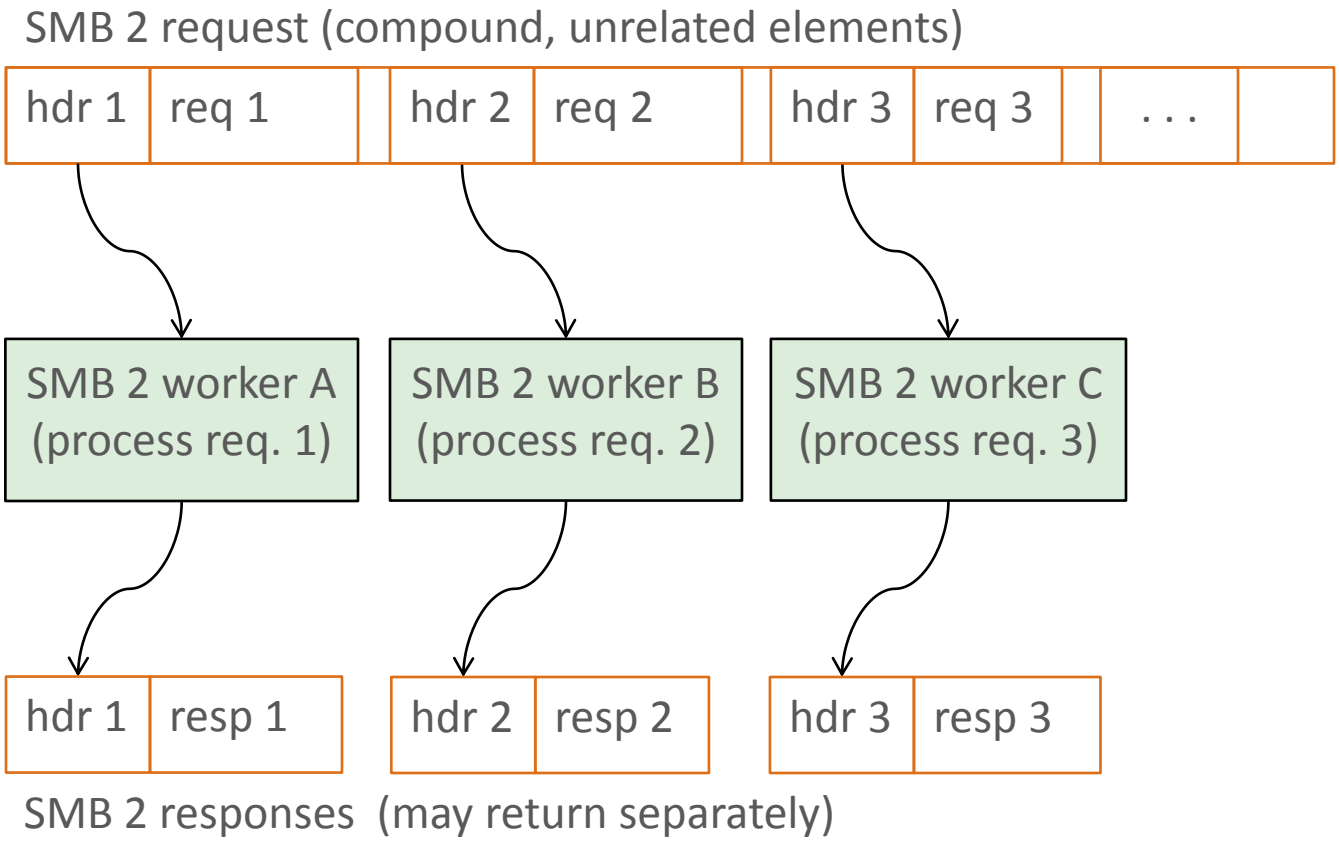
Message processing (SMB1)



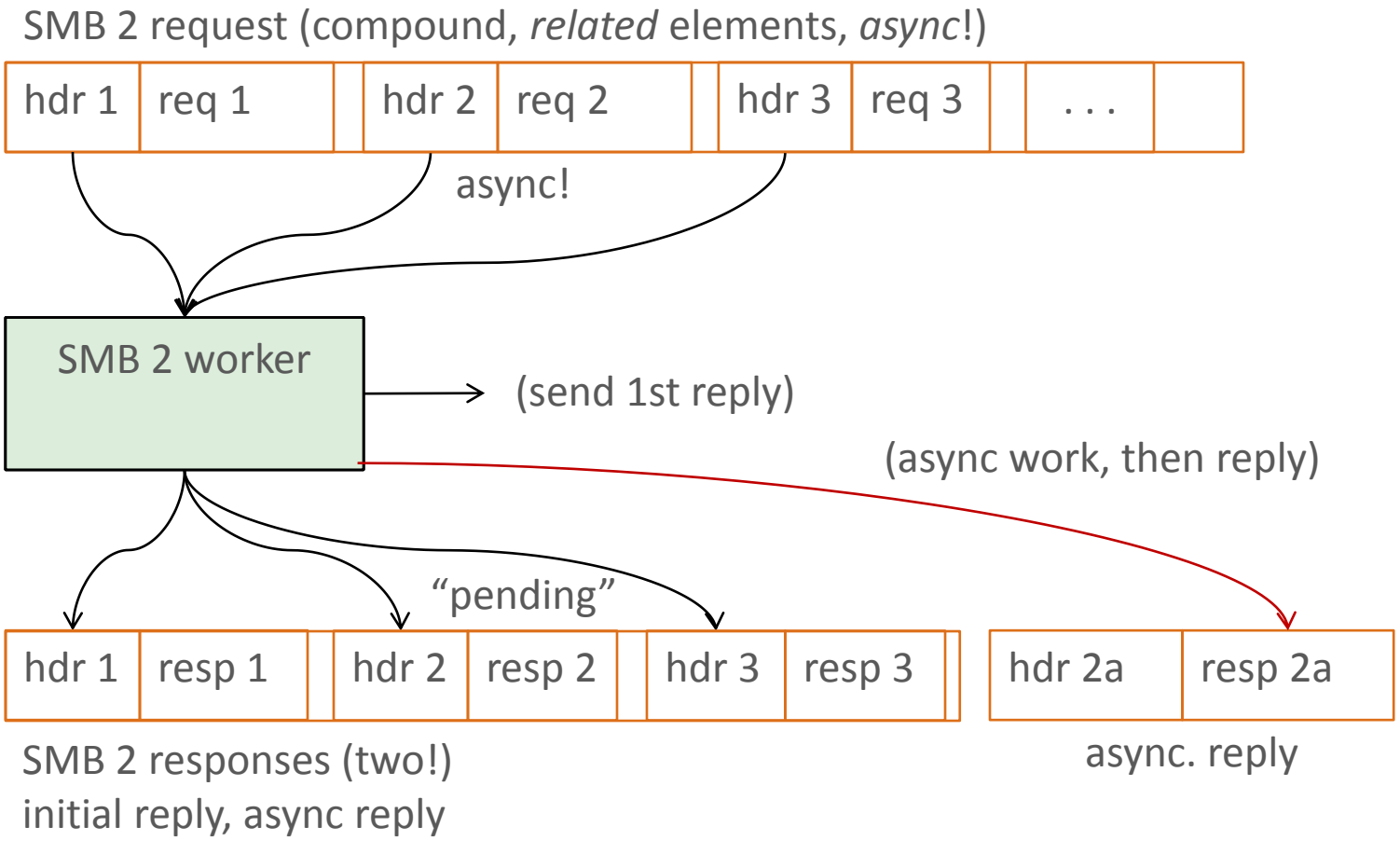
Message processing (SMB2, “related”)



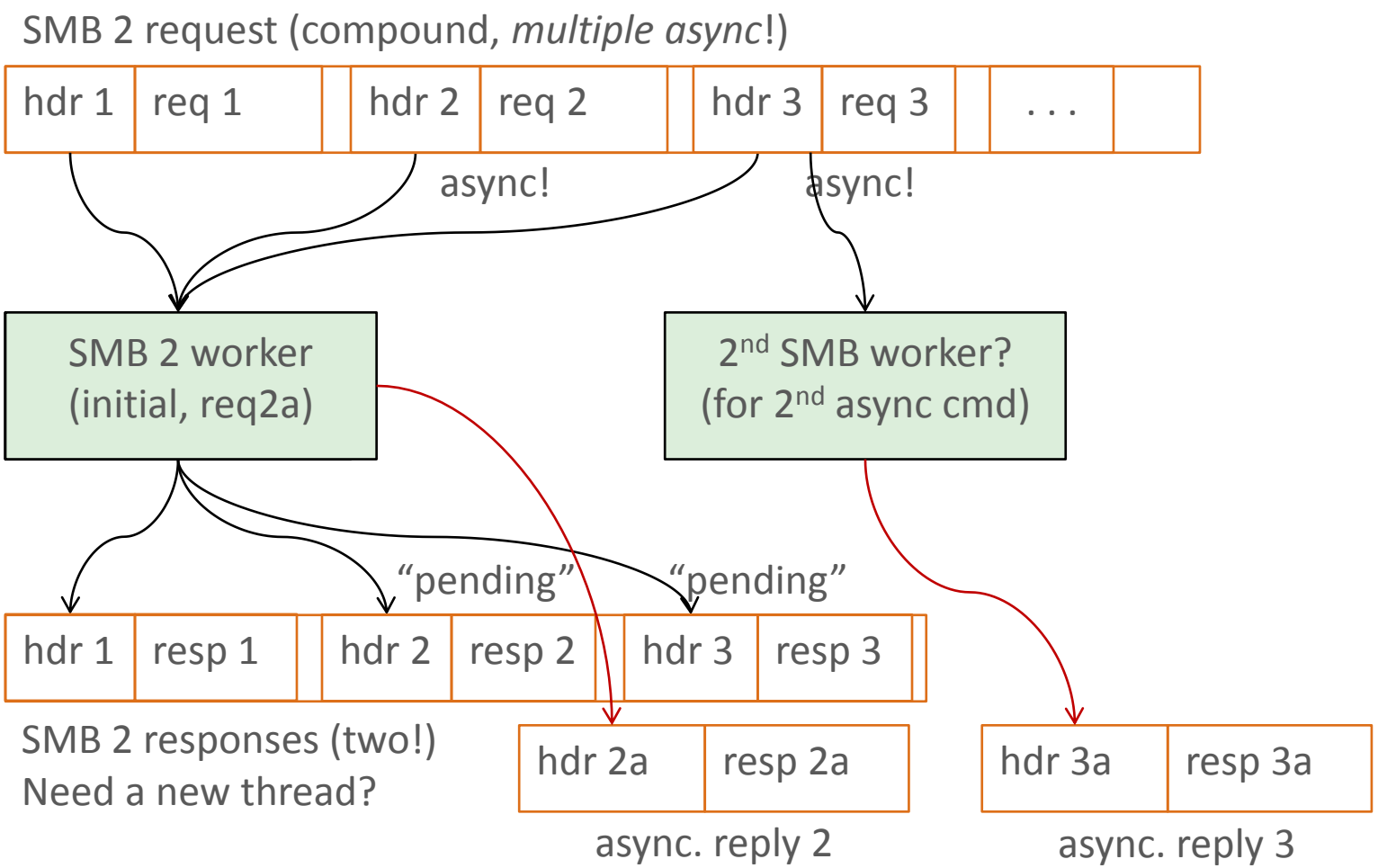
Message processing (SMB2, “unrelated”)



Message processing (SMB2, related, async)



Message processing, multiple async (Uh oh…)

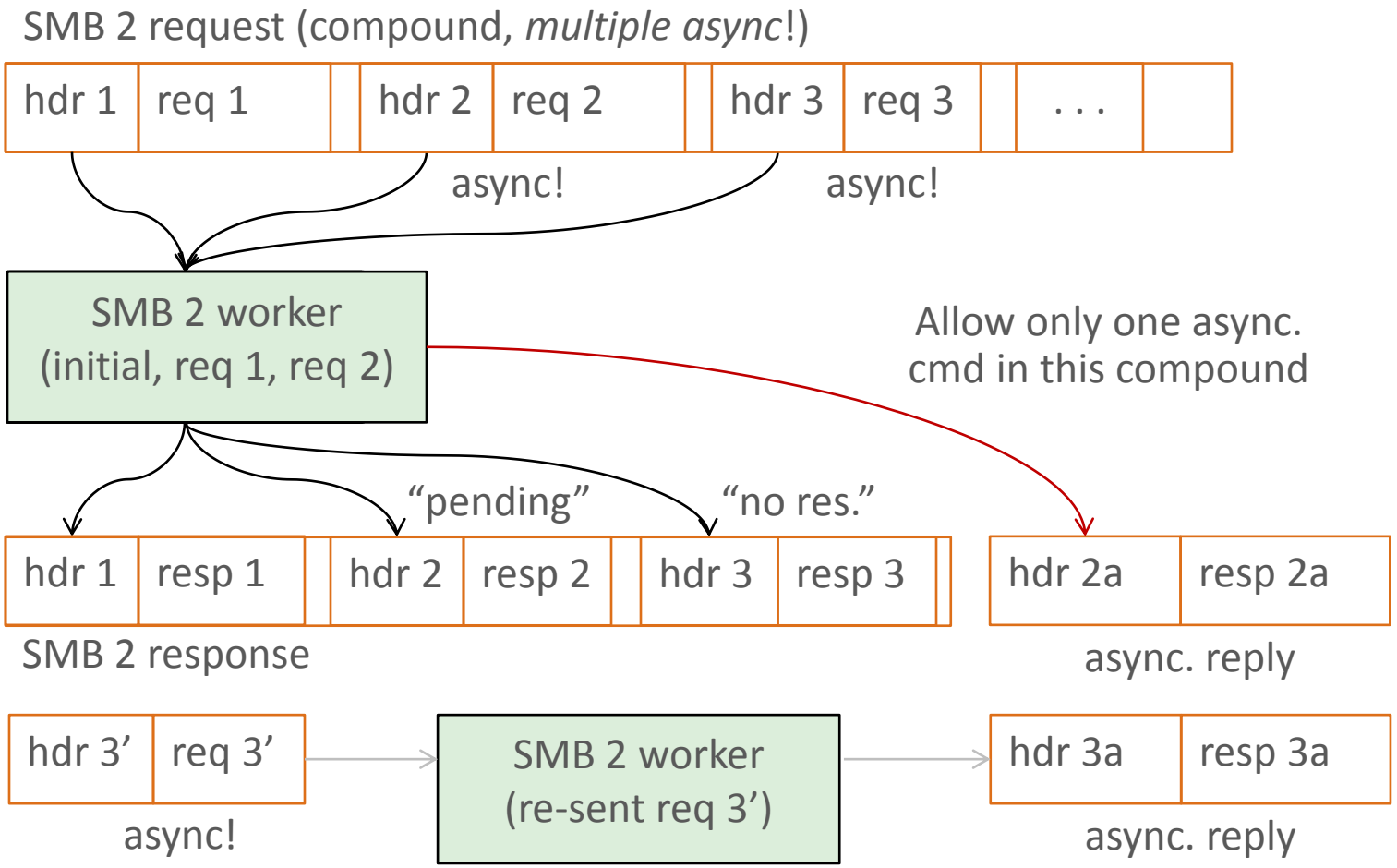


Message processing, multiple async. – How?

An important simplification

- "The server is allowed to say 'no' to any compounded request.“ (Thanks to David Kruze of MS for this tip.)
  - Server can return STATUS_INSUFFICIENT_RESOURCES for any request in a compound and the client will resubmit that request separately.
  - If we ever see more than one request in a compound that needs to "go async", we can just return the special error for the 2nd and later.
- It turns out we never need to actually do this!
  - We've never seen a compound request with more than one command that needed to “go async”
  - Commands needing an async. response appear to always be last.

Message processing, limited async.

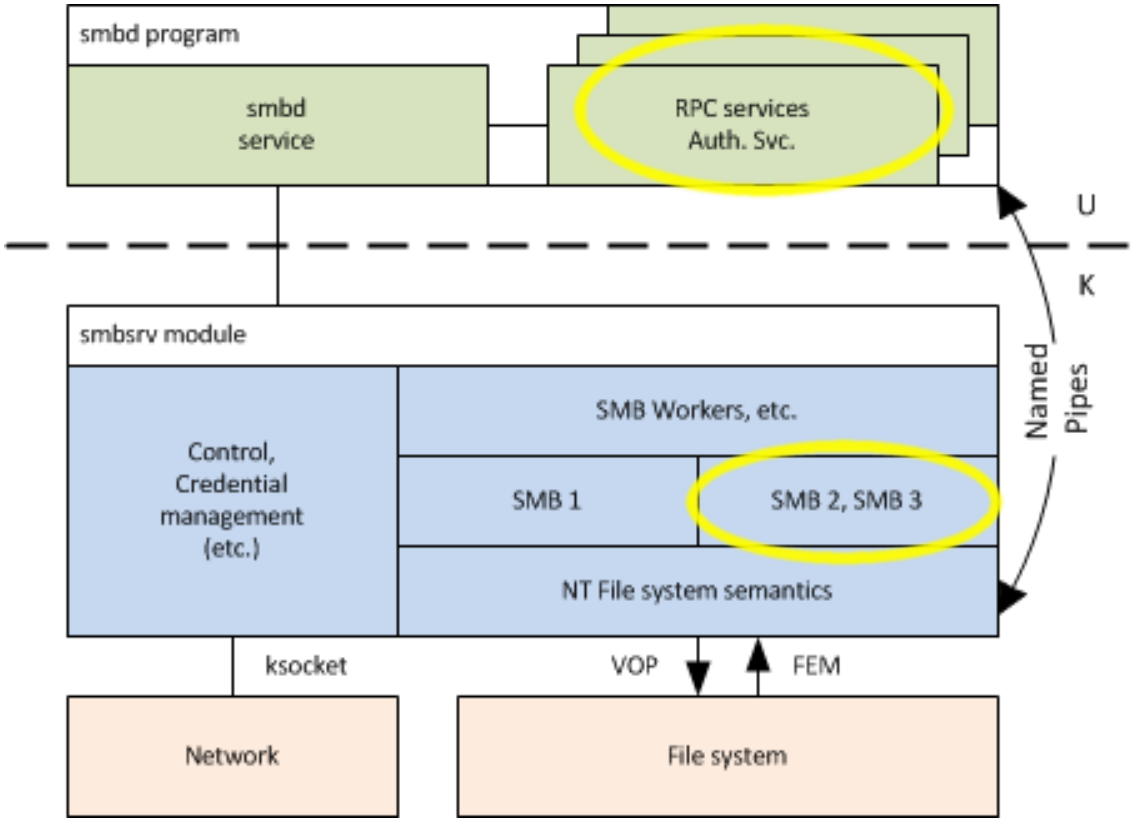


SMB Server design changes (summary)

- The existing multi-thread model continues to work in SMB2, because we can limit compounds to at most one async element.
- Largest design work was the new SMB2 dispatch loop, and it's handling of a possible async. command.
- Next largest was finding SMB1-specific logic in what would now become common code shared by SMB1 & SMB2
- Significant design changes for authentication and RPC over SMB "named pipes" (both use AF_UNIX sockets)

SMB Server design changes — impact

Where the major changes land



Development Approach

Challenges we faced, Solutions chosen, and Architecture changes to ease development

SMB 2 Development: challenges

- 400 page specification [MS-SMB2]
  - New dispatch model (sync. vs async. replies)
  - 18 protocol commands. Many are simple…
    a few are *huge* ("smb2_create")
- Aggressive schedule (wanted alpha in ~ 6 mo.)
- Had some "technical debt" to pay
  - some "common" code was really SMB1-specific
  - extended security necessary for SMB2
- Concurrently support shipping versions

SMB 2 Development: desires

Want accelerated development

We knew we had a lot of code to write and debug, much of that in-kernel code. Wish list:

- Very fast edit/compile/debug cycle (< 1 min!)
- Ability to use source-level debuggers
- Retain ability to use mdb, ::smblist etc.

SMB 2 Development: approach taken

- Re-use as much of the current code as possible
- Work incrementally (no time to “start over”)
- Keep a similar reader + taskq dispatch model, *if possible* (was not obvious whether we could)
- Remove SMB1-specific actions from common code, i.e. smbsr_put_error calls (return status instead)
- Improved testing for both SMB1 & SMB2 using open source and commercial testing tools

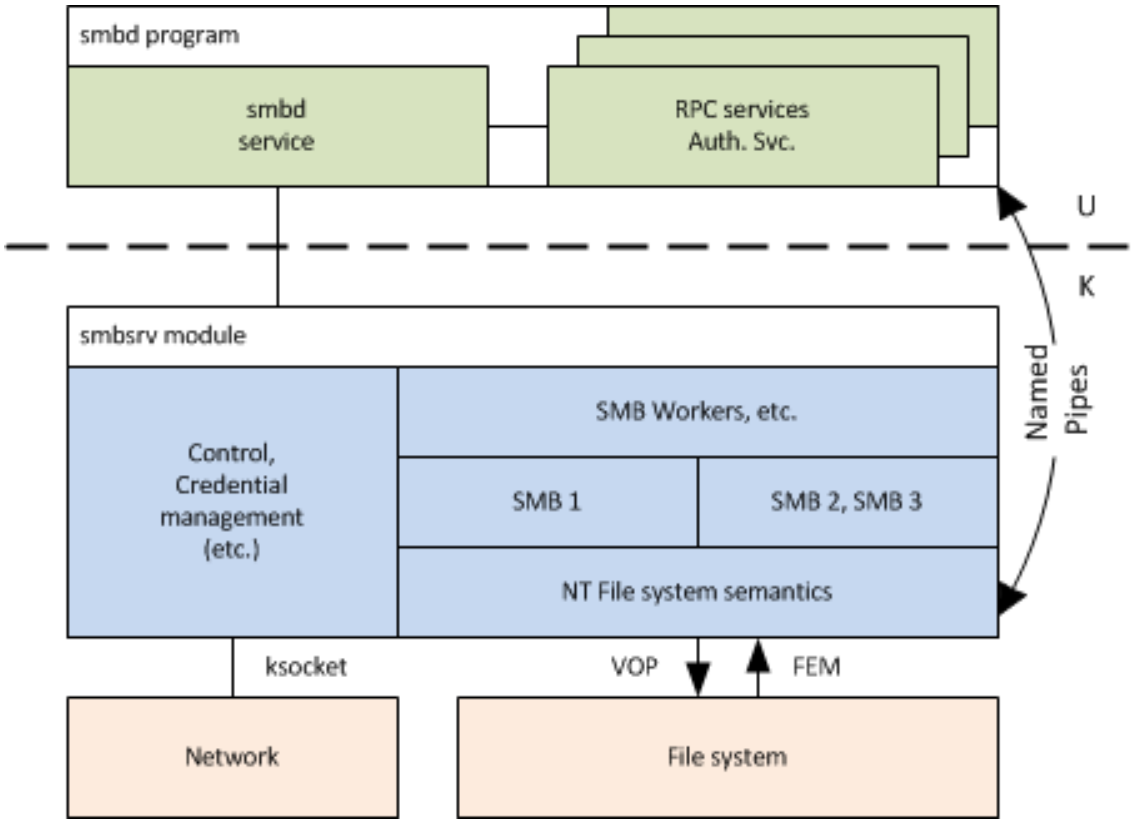
SMB 2 Development: clever tricks!

Do development in user-space when we can!

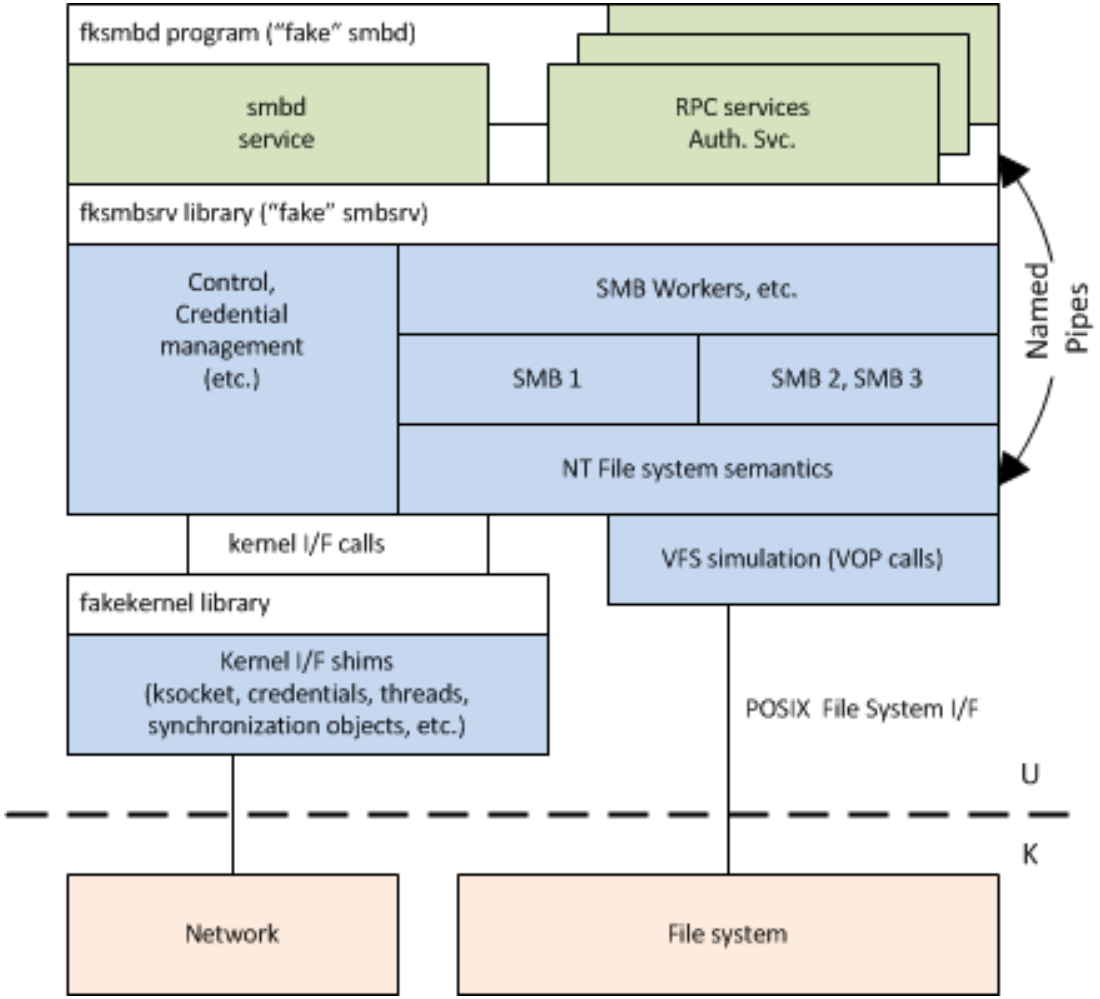
- Implemented a user-space version of the SMB server, allowing source-level debuggers, fast edit/compile/debug (We call this "fake smbd", or "fksmbd")
- How does this work? What can it do?
  - Shims for kernel interfaces (“libfakekernel”)
  - Stubs for things we don't need
  - Minimal implementations of many things
  - Many limitations, not suitable for deployment.
- Credit where credit is due:
  Borrowed ideas and code from ZFS (libzpool)

How does this affect the architecture? …

SMB Server Architecture (“real” server)



fksmbd Server Architecture (“fake kernel”)



fksmbd debugging features

- Sufficient functionality for most protocol-level debugging.
- Source-level debuggers work (dbx/gdb)
- mdb "just works" (kernel or user-mode)
  commands like "::smblist" work
  (mdb module for “libfksmbsrv”)
- Also have "quick" scripts (for partial builds)
  sufficient to compile & run "fksmbd“
  (1 min. edit/compile/debug cycle)

All of these were key for enabling productivity,
particularly when making extensive changes.

fksmbd limitations

- Authentication works, but fksmbd uses *only* the credentials of the user who runs it. (all cred. interfaces are stubs)
- Just one zone; other zone interfaces stubbed out.
- VFS layer simulation very limited
  - Ordinary things like fop_lookup work.
  - Fancy things like fop_setsecattr are just stubs.
  - Would not scale (open FD per active vnode)

This is intended as a debugging tool, not for deployment.

Lessons Learned

What worked well?
(or not so well?)

Lessons Learned

What worked well? (or not so well...)

- The ability to use *all* our favorite tools was a huge help in accelerating development. (dtrace, mdb, dbx|gdb)
- This gave us room to "pay down some technical debt" i.e.
  - Reduce SMB1-specific logic in common code
  - Improve the named pipe implementation (use AF_UNIX sockets)
- Incremental development is a safe bet (small steps)
- Want the code testable at all times.

Lessons Learned (cont.)

- Automated testing helps a lot.
  - We ran Samba's "smbtorture" after builds (via Jenkins)
  - Also ran some commercial test tools periodically
- Interfaces with no consumers might not work as advertised
  - Rename bugs in “common” SMB code
  - SO_RCVTIMO bug for AF_UNIX, etc.
- Early testing in the field would have helped
  (as much as you test, there’s nothing like field experience)
- Like to use “feature preview” for major new features,
  (better not to enable by default in the first release)

Next Steps

Getting this work upstream to illumos
continuing SMB development

Next Steps

- Work on getting what's "ready" into illumos
- SMB 2.1 and optional features
  - Large MTU (and multi-credit)
  - Durable handles
- SMB 3.x, multi-channel, etc.
  - AES-256-CBC signing (done)
  - mult-channel support
  - server-side copy (ODX)
- Windows Management Interfaces
  New, wbem-based administration APIs

Usual disclaimers: No promises about future plans here.

Upstream this work to illumos?

Getting the SMB work up to illumos will take some time.

- There are about 150 commits to look at. With some reorganization and related commits "squashed" it's still about 50 commits in about 8 chunks.
- See: smbsrv2-rework in github.com/gwr
  https://github.com/gwr/illumos-gate/compare/smbsrv2-rework
- The XXXX markers are the points that make sense as testable milestones, and reasonable chunks to push.

Anyone available to help with this?

Upstream work, by subject

Subject	Inserts	Deletes
Kerberos auth. inbound	898	119
New DC Locator, AD join fixes	5,861	2,224
SMB 2.1 (and follow-up fixes)	14,111	3,878
Extended Security, inbound	4,615	2,753
Improved SMB named pipes	1,071	1,865
User-mode server (for debug) *	13,708	2,090
Fixes (part B)	284	918
SMB server in a zone	896	885
Fixes (part A)	5,457	3,705
totals:	46,901	18,437

* Inflated by some necessary copying & repeated changes

Links: (for more information)

- NexentaStor (nexenta.com)
- Summary of SMB 2.1 features
  http://technet.microsoft.com/en-us/library/ff625695.aspx
- Summary of SMB 3.0 features
  http://support.microsoft.com/kb/2709568
- Staging area for integration into illumos
  http://github.com/gwr/illumos-gate/compare/smbsrv2-rework
- Nexenta's open fork of illumos gate
  http://github.com/Nexenta/illumos-nexenta

Thank-you!

www.nexenta.com