Advanced Interoperability in a hetrogeneous environment

By Jeremy Allison

Development Team

email: jra@samba.org

Hetrogeneous problems

- All IT departments have to deal with integrating Windows systems with other Operating Systems and network platforms.
- The Windows family of Operating Systems is designed to be compatible only with itself.
  - Uses different user and group models.
  - Uses different authentication and security methods.
  - Uses proprietary file and printer sharing protocols.
  - Uses a proprietary directory service.
- Windows hides the API's needed to replace these models on a client.

Hetrogeneous solutions

- Client side changes are not scalable in a large organization.
  - Always easier to make few changes on servers.
  - Hidden API's on clients make full conversion of Windows clients to UNIX protocols currently impossible.
- Various solutions exist to make UNIX/Linux servers fit into a Windows Domain environment.
- Samba is the most popular non-Windows server solution (estimates are 30% of Windows clients talk to a Samba server).

New Samba 2.2 design philosophy

- For Samba 2.2 the design philosophy of the code was changed.
  - Previous to 2.2, Samba tried to track the X/Open and Microsoft specification documents.
- The goal for Samba 2.2 was correctness, defined as "the way Windows 2000 acts".
  - Test tools were written to test the behaviour of Windows 2000 over the wire, and Samba was modified accordingly.
  - These Open Source test tools are now used by all the major SMB/CIFS vendors (including Microsoft) for compatibility testing.

New Features in Samba 2.2

- Ability to act as a Domain controller for Windows 2000 and Windows NT clients.
- Support of Windows 2000/NT access control lists (ACLs) by mapping them into POSIX ACLs.
- Full 64 bit locking, even on 32 bit (x86) platforms.
- Full implementation of Windows 2000/NT "point and print" auto printer driver download ability.
- Management of Samba shares using native Windows tools.
- Interoperability with winbindd - single sign-on.

Quick Overview

- Samba consists of two user mode daemons.
    - nmbd - NetBIOS naming daemon. Not covered further in this talk.
    - smbd - Main file and print serving code.
- smbd has evolved over seven years of coding.
- Originally a file server, it has expanded to include print services, authentication services and now an implementation of an entire RPC protocol.
    - smbd is too complex. Much work is being done to simplify it and break it into manageable parts.

smbd design

- smbd consists of a single process per connected "client".
  - Multi-user Windows servers such as Citrix or Terminal server can break this assumption.
- UNIX user context is used for security.
  - This is a very important point. smbd does not enforce security itself, it sets the effective userid to the UNIX uid mapped to the client context and lets the OS determine access. No "root race" holes.
  - As a consequence of this smbd is single threaded. POSIX threads are not guaranteed to have a security context.

smbd event flow

- Dealing with incoming SMB messages [smbd/process.c]:
  - select()
  - read message from socket into 64k buffer [inbuf].
  - check message for correctness
  - check incoming user context, change effective uid if needed.
  - process message (done in switch_message()) - generating reply into 64k buffer [outbuf].
  - send reply to client.
  - periodically do general housekeeping (time-out events etc.)

smbd design (continued)

- As close to Windows semantics as POSIX allows.
- Try to overlay POSIX filesystem with Windows semantics in the core code.
  - Don't create a "shadow" filesystem with dot files.
  - Don't create mappings that have no meaning to the underlying system (ACL or user databases).
  - No modifying file contents (no CR/LF translation).
- VFS layer in Samba 3.0 will provide "pluggable" mechanisms to provide this kind of OEM customization [examples/VFS/skel.c].

VFS hooks in smbd 3.0

- All calls into POSIX (open/close/read/write etc...) are vectored via a shared library loaded function table.
- Default POSIX operations are available via import.
  - "pass-through" interfaces are thus possible, a sample audit VFS plug-in is supplied.
- VFS plug-ins are loaded per share, all pathnames passed to the VFS are UNIX character set format (conversion from DOS codepage is done before call).

Mapping Win32 concepts to POSIX

- Win32 has some concepts that don't map well to POSIX.
  - Deny modes.
  - Oplocks.
  - Byte range locks.
  - ChangeNotify.
  - Timed lock requests.
- Samba implements deny modes between smbd processes via a shared memory area, implemented differently within 2.0.x and above.

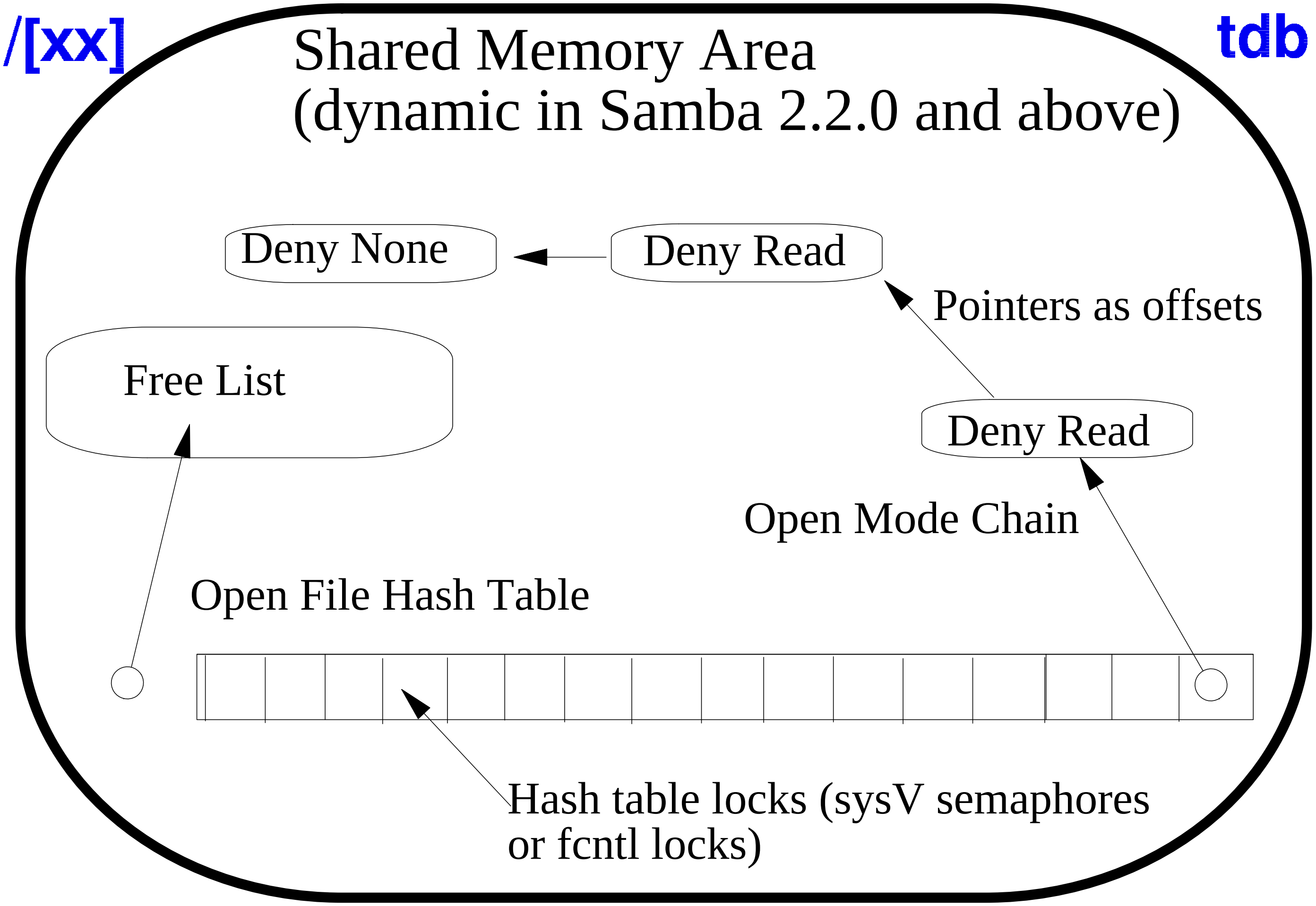
Deny mode semantics in POSIX

- POSIX has no "deny modes". Samba layers these over ordinary POSIX open calls [smbd/open.c].
  - POSIX apps do not interact with DENY modes.
  - Reason - what happens if someone opens /etc/passwd with DENY_ALL ?
  - DENY mode semantics are not logical - adding this to POSIX is not good design.
- Samba implements a fast, smbd to smbd mechanism to convey deny modes between user processes.
  - No centralized deny mode daemon needed.

Samba shared memory Deny mode database

locking/[xx] in 2.0.x

tdb in 2.2.0



Creating Oplocks in POSIX

- Allowing Oplocks on top of POSIX breaks consistent view of filesystem (and Samba philosophy) [smbd/oplocks.c]
  - However, too useful not to implement. Needed for SMB speed.
- Deny mode database holds all shared info about open file state. Oplock records added to this data.
- Blocking IPC mechanism between smbds needed that would integrate into select()/poll().
- UDP messages on loopback interface chosen.

Oplock communications

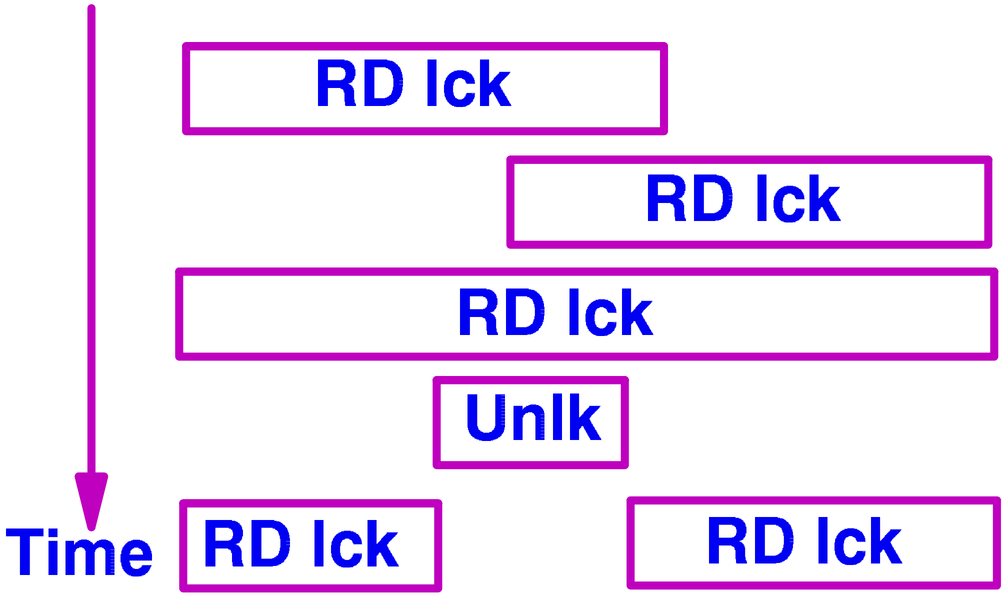
- On break request, smbd locks db, finds holder of oplock, sends break request via UDP port, releases db lock then blocks awaiting reply).
  - Code in [smbd/open.c] and [smd/oplock.c] - request_oplock_break() function.
- Receiver smbd gives priority to incoming UDP messages in select(), recurses into secondary smbd processing loop [smbd/oplock.c].
  - "Dangerous" messages that may cause an oplock break from the receiving smbd are queued at this time.
- On exit from recursed state, queued messages are given priority [smbd/process.c] - receive_message_or_smb().

The swamp - mapping Win32 byte range locks to POSIX

- Win32 byte range locks seem to be easy to map into POSIX.
  - Approach chosen in all Samba versions 2.0.x and before.
  - Depends upon locking conflicts being handled at client redirector.
  - Not possible to give exact Windows semantics.
- Samba 2.2.x and 3.0 have correct Win32 semantics.
  - "Correct" here means 'what NT does'. Has little relation to Win32 documentation or the spec.

POSIX locks - the exact semantics

- Lock ranges can be merged/split.
- Lock ranges can be upgraded/downgraded.
- 32/64 bit signed, not unsigned ranges.



Kernel view of locks

POSIX lock semantics (continued).

- Killer issue : POSIX locks are **<u>per process</u>**, not per file descriptor.
- Eg:

```
int fd1 = open("/tmp/bibble", O_RDWR);
fcntl(fd1, F_SETLK, &lock_struct);
fd2 = dup(fd1);
close(fd2);
```

<u>SURPRISE !</u> The lock you thought you had on fd1 is now gone !

In anyones wildest dreams this is not desirable behavior.

POSIX lock semantics (continued).

- Samba 2.0.x solution to this problem was to reference count all opens on a file onto a single fd, open read/write (if possible).
  - Conserves fd usage.
  - Samba checks prohibited security overrides.
- Disadvantages are :
  - Multiple opens under different uids - need to use fork() as a procedure call to check return.
  - smbd is lying to operating system about access mode.
- 2.2.0/3.0 solution - store pending closes in a tdb.
  - Allows multiple opens to obey Samba philosophy.

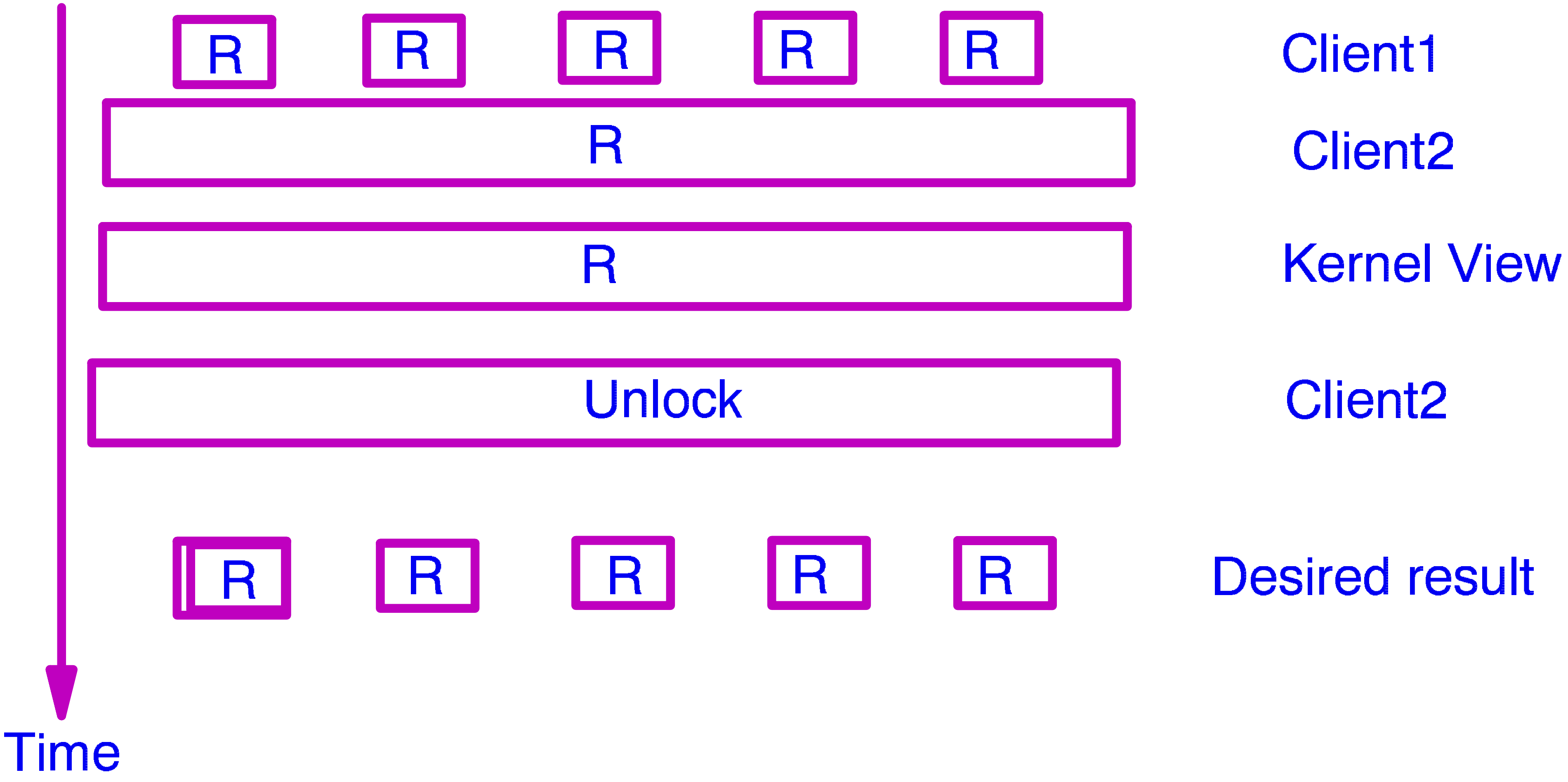
"Welcome to Fantasy Island" : The Win32 lock spec.

- Win32 locks as described in Win32 docs are not what is implemented in Windows NT.
  - Locks can be downgraded by overlaying read locks onto write locks and then doing one unlock.
  - Compatible locks can be stacked on top of each other and are then reference counted.
- The only way Samba can implement this is with a locking database.
  - This tdb database [locking/brlock.c] implements full 64 bit Win32 lock semantics, indexed by dev/inode pairs.
  - Any locks passed by this are (optionally) passed to a POSIX lock mapping layer [locking/posix.c].

Mapping Win32 locks to POSIX

- POSIX lock layer attempts to map given 64 bit unsigned lock onto signed (64 or 32, depending on filesystem) bit POSIX lock.
  - If no POSIX mapping possible - discard the request (return True - POSIX app can't get to this range anyway).
- Locks that pass are then stored in a second, lower level tdb that contains full record of all existent POSIX locks on a dev/inode pair.
  - This is needed as POSIX kernel will lose information when locks are overlapped.

Mapping Win32 locks onto POSIX (continued).



ChangeNotify and timed locks

- ChangeNotify is a problem as it is resource intensive.
  - Similar to FAM on IRIX ((kernel interface)- this is now available on Linux.
  - For portability reasons, Samba currently does a periodic scan, with no depth.
  - Produces a hash of the directory contents and checks this in the idle loop [smbd/nttrans.c].
- Timed locks are implemented by all lock requests being instantaneously checked with the request packet being queued until a check succeeds in the idle loop (or timeout) [smbd/blocking.c].

Samba DCE/RPC subsystem: incoming

- Pipe opens are done on a IPC$ share, smbd redirects into pipe handling code [smbd/pipes.c].
- All writes onto pipe handle are buffered into a continuously growing (length limited) memory buffer [rpc_server/srv_pipe_hnd.c].
    - On an authenticated RPC bind (NTLM handshake), the user credentials are stored with the pipe [rpc_server/srv_pipe.c].
    - As a PDU's worth of data is received, the header is processed, stripped off (all sign & seal processing is done here) and the incoming PDU data is appended.
    - When the complete RPC is received then the pipe/function specific processing is invoked.

Samba DCE/RPC subsystem: outgoing

- After successful processing of the RPC request the outgoing data stream is marshalled into an auto-growing buffer via [rpc_parse/parse_XXX] calls.
- When the client does a read on the RPC pipe the outgoing data is split into PDU sized chunks [rpc_server/srv_pipe_hnd.c] and returned as the read data.
- Additional pipes (eg. MS-DFS pipe) can be added into pipes tables in [rpc_parse/parse_rpc.c] - uuid, and [rpc_server/srv_pipe.c] - pipe function table.

Resources

- Main Samba Web site :
  - http://samba.org
- Newsgroup :
  - news:comp.protocols.smb
- Samba discussion list :
  - email: samba@samba.org
- Samba development list :
  - email: samba-technical@samba.org