SAMBA Opening Windows to a Wider World

Asynchronous Samba

Jeremy Allison
Samba Team

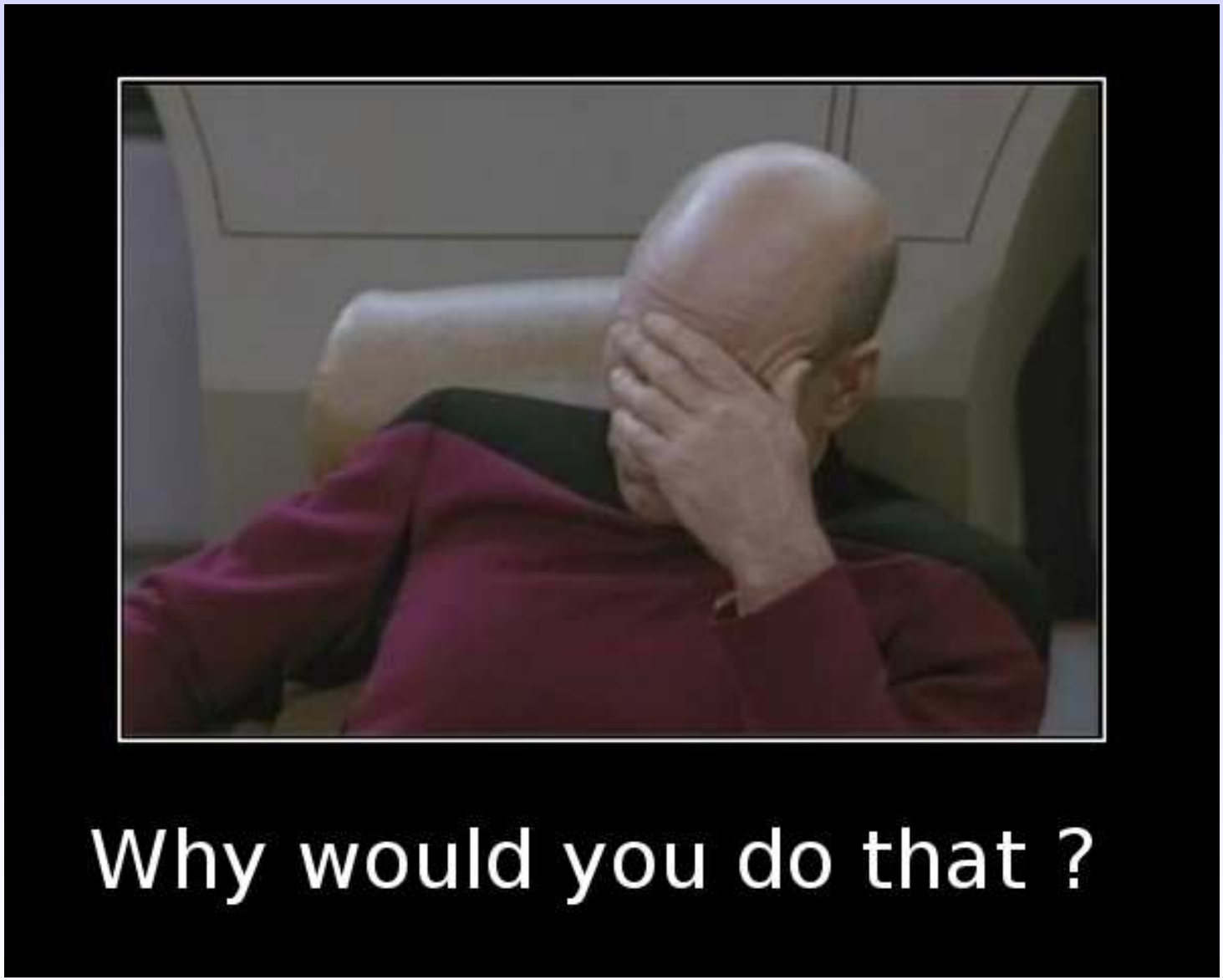
jra@samba.org

Tridge's view

"Threads are evil. Processes are ugly. State machines send you mad."

Andrew Tridgell, "Musings on Software Engineering" 2005.

So of course Samba utilizes all three simultaneously..



Using processes

- Historical reasons.
  - Earliest Samba architectural decision, one process per client (single connected TCP socket).
- Still (IMHO) a good decision.
  - Robustness.
  - Scalability (depending on underlying OS).
  - Allowed easy clustering separation.
  - Security separation.
    - Linux (and Windows I guess :-) still the only OS(es) where threads can have separate credentials.
- Still an ideal choice for programmers when dealing with code complexity.

Using processes

- Ideal choice when splitting off large areas of functionality.
  - Splitting out of processes to implement server end of names pipes is an ideal case (spoolssd).
  - Depends upon good inter-process communication (IPC).
    - Volker's recent work on our IPC layer (for Samba 4.2) has improved our scalability immensely.
- Little shared state by default (so long as you're careful after the fork() call).
  - winbindd code spins off separate processes for background DNS resolution.

Using processes - downsides

- Just too heavyweight.
  - Using a separate process for every required asynchronous event simply overwhelms machine resources.
  - Copy-on-write issues.
    - Any memory allocation / free that modifies the malloc pool can cause many copy on write issues as pages have to be duplicated.
- Cleanup issues.
  - Unless properly reaped, zombie children can pile up under a process.
- Heavyweight IPC mechanisms needed.

Using threads

- First use of threads within Samba (at least on Linux) was under the covers via a library.
  - Glibc POSIX aio uses threads to implement asynchronous IO (badly, see my 2012 talk..).
- Ideal way to ensure efficient use of all available cores.
  - Lightweight way of sharing kernel vm mappings for a process.
  - Modern Samba uses threads for some blocking system calls (pread, pwrite, open, fsync).
- Volker created two good abstraction layers for Samba (pthreadpool, asys) to make using threads easier.
  - pthreadpool even works without threads :-).

Using threads - downsides

- Monstrous complexity.
  - Shared-everything means all shared data (and there is a LOT in Samba) needs protecting by mutexes or reader-writer locks.
- Samba code was not designed this way.
  - Going fully threaded is a rewrite, not a retro-fit.
- A fully threaded server depends on either:
  - Per-thread credentials to protect file system access.
  - Careful coding as root to avoid race conditions (Ganesha, user space NFS does this – very tricky).
    - POSIX ACL evaluation becomes impossible.

Using state machines

- Tries to manage asynchronous complexity by breaking it up and linearizing it.
- As only one event is being processed at a time it removes the locking requirements of threaded code.
- Coupled with non-blocking sockets allows work to be done whilst waiting for I/O to complete.
- No dependence on impossible to debug race conditions.
- Depends upon an underlying event library.
  - Choices were libevent, libev or create our own..
  - In the grand Samba tradition, tevent was created :-).

The tevent solution

- Tevent integrates directly with talloc(), making it really easy to use within Samba.
  - https://tevent.samba.org/tevent_tutorial.html
- Tevent handles things libevent originally did not (POSIX real-time signals, used in the Linux kernel for leases).
- Allows for usage that libevent disallows.
  - Separate read and write events attached to the same file descriptor.
- Efficiently uses epoll() on Linux (similar versions for Solaris ports created, kqueue on FreeBSD in process).

How is tevent used ? (a variable called 'req')

- A client request ends up calling a XXXX_send() function, which calls tevent_req_create(), with associated state structure.
  - tevent_req_create() returns an opaque pointer which can be used to reference the outstanding event.
  - Completion callback is attached to this pointer - by convention XXXX_done().
  - Asynchronous operations scheduled (often using pthreadpool and asys).
- Control passes back to main loop.

How is tevent used (continued) ? (a variable called 'subreq')

- When async operation completes it calls tevent_req_done().
  - This calls the callback XXXX_done().
  - XXXX_done() calls XXXX_recv() to return the results of the operation.
  - Opaque pointer is talloc_free()'d and the operation is over (talloc destructors take care of cleanups).
- Multi-part events can be nested by being split into multiple requests (usually named 'subreq') and chained together.
  - As each event completes it calls back up the chain.

The tevent Swiss Army Knife (more variables called 'req')

- Many kinds of async events can be handled by tevent.
  - Timer functions.
  - Lossless signal handling (normal and real-time).
  - File descriptor I/O.
    - Including non-blocking IO with partial reads/writes.
  - Immediate events.
    - 'Run me next' kind of requests from low down in the event stack.
- Queueing API's built in.
  - Allows events to be buffered within the event main loop.

Tevent downsides (all variables called 'req')

- Currently only used inside Samba, sssd and ctdb.
  - Little sample code for new users to use as examples.
- Complex set of naming rules/conventions.
  - Easy to get wrong and get confused between tevent_req_done() / tevent_req_post() / tevent_req_error() etc.
- Complex API, lots of 'unwritten rules'.
  - Currently the best guide is looking at existing code.
  - Monstrous complexity, just not as bad as threads.
  - Hard to program – non natural code flow.

Integrating tevent with the Samba VFS

- pread_send() and pwrite_send() return tevent structures (req).
  - Older interfaces still exist, used by the SMB1 non-asynchronous code.
  - SMB2+ (and SMB1 aio code) natively use the tevent-based versions of the VFS calls.
- Much more work needs to be done.
  - Firstly, VFS interfaces need to have the tevent variants (XXXX_send / XXXX_recv) added.
  - Secondly, the main server code needs to be split into state-machine style of functions at every blocking point.

Who goes there ?

- Subtle but really important issue to consider.
  - User id of tevent request that may be deferred.
  - After async request finishes, and callback is re-scheduled the uid of the process may have changed.
    - Oh :-(. That's a “bad thing”.
- Currently this state has to be stored manually (full token list and then restored in the callback function.
  - Only current example of this is code inside async SMB2 logoff / tree disconnect code path.
  - But metze has plans (flamethrower for this code :-).

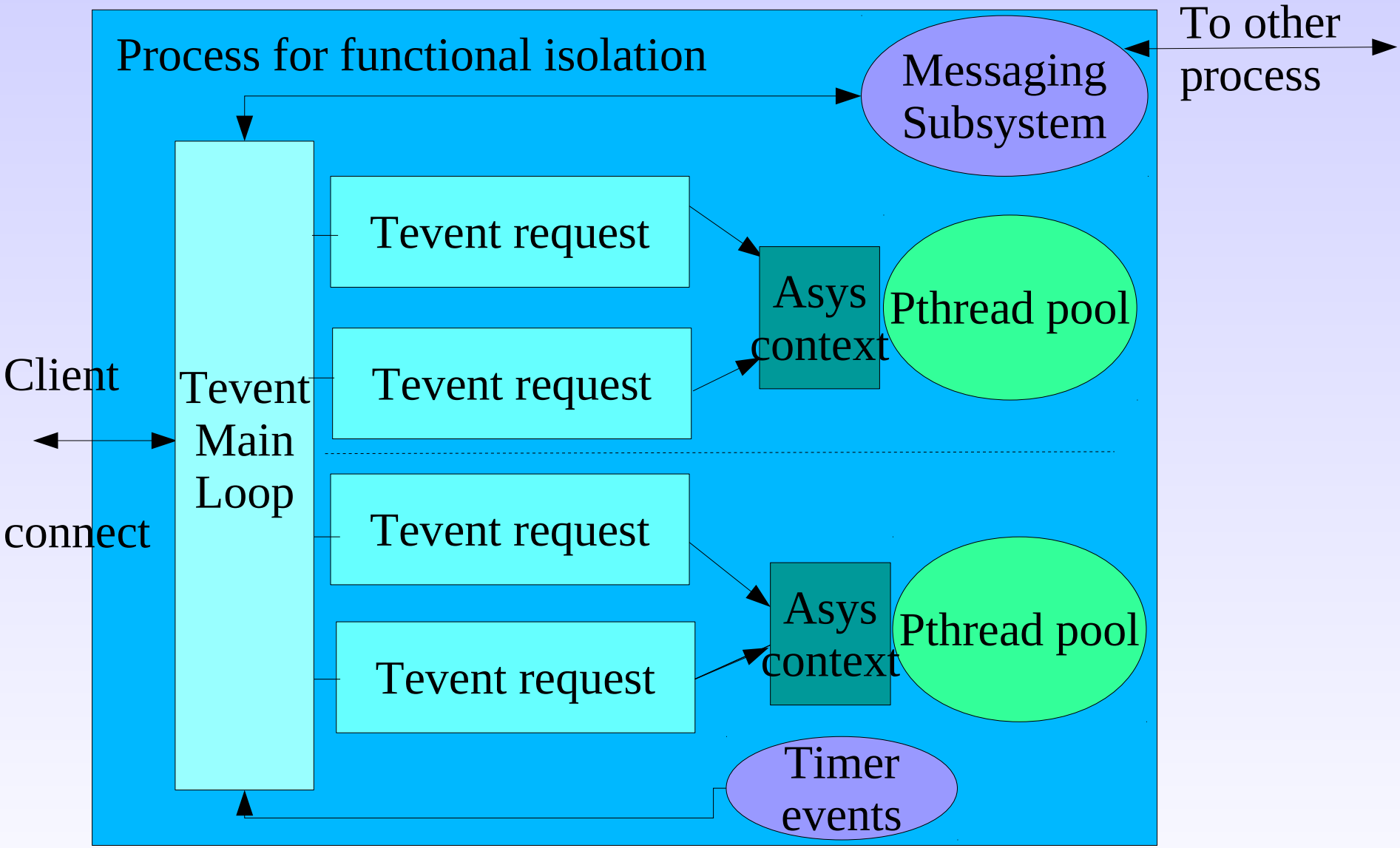
How not to do it..

- The SMB2 CreateFile() code path also uses tevent internally.
  - DO NOT USE THIS AS EXAMPLE CODE :-).
  - First integration of tevent into 'backend' file server code (as opposed to tevent use in 'front end' request serving code).
  - Mangled tevent semantics to match existing asynchronous interface created for SMB1 deferred opens.
- New code written for SMB2 read/write requests is a much cleaner sample.

Hierarchy

- At the lowest level, a tevent_req can spin off a pthreadpool (or use an asys asynchronous system call) to provide true parallelism and utilize multiple cores per proces.
- Within Samba:
  - Processes contain:
    - tevent requests which contain:
      - asys contexts which utilize:
        - pthreadpools to implement blocking system calls asynchronously.
      - Timed events.
      - File descriptor events.
      - Immediate events.
      - Signal events.

Why we did it 'that way'



Making it so..

- To be a proficient Samba programmer today, you MUST understand tevent semantics.
  - Like it or not, tevent is the way most asynchronous complexity is going to be handled in the Samba code going forward.
  - Unless nirvana (complete thread-safe code within all of Samba) is achieved, tevent is the best middle ground available.
  - The more code is converted, the easier converting the remaining code will become (experience and added boilerplate to use / copy from).

Questions and Comments ?

Email: jra@samba.org

Slides available at:

ftp://samba.org/pub/samba/slides/sambaxp-2014-async-samba.odp