Threads vs Processes

Andrew Tridgell
tridge@samba.org

“Aren't threads faster?”

- A very common question but a complex answer
  - Faster for what?
  - What do threads/processes actually do?
  - What can the hardware do?
- Systems programming
  - Answering this question reveals a lot about systems programming
  - I hope it will also make you think a bit about how operating systems work

What is a thread/process?

- An abstraction of a unit of execution
  - We'll generically call them both 'tasks'
- What is in a task?
  - Instructions to execute
  - Memory (shared and non-shared)
  - File descriptors
  - Credentials
  - Locks
  - Network resources
- Relationship to CPUs
  - Many/most computers have more than 1 CPU now
  - It is common to have many tasks per CPU

The key differences

- Threads
  - ???
- Processes
  - ???

The key differences

- Threads
    - Will by default share memory
    - Will share file descriptors
    - Will share filesystem context
    - Will share signal handling
- Processes
    - Will by default not share memory
    - Most file descriptors not shared
    - Don't share filesystem context
    - Don't share signal handling

Underneath the hood

- On Linux, both processes and threads are created with clone()

Creating a thread:

```
clone(child_stack=0x420cc260, flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
parent_tidptr=0x420cc9e0, tls=0x420cc950, child_tidptr=0x420cc9e0)
```

Creating a process:

```
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|
CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f4936ecc770)
```

A Sample Workload

- A network media server
    - Clients connect, and transfer images/videos/music
    - Users login with their own accounts
    - Files stored in a local filesystem
- What work needs to be done?
    - ???

Network Media server ....

- What work needs to be done
  - Computation: for format conversions etc
  - Memory manipulation
  - File IO
  - Database access?
  - Locking
  - Network IO
  - Credential handling

- Should it use threads?

malloc()

- Memory allocation
  - Extremely common task in almost all programs
- What work needs to be done?
  - ???

malloc()

- What work needs to be done?
  - Possibly grab more pages from the OS
  - Lock data structures?
  - Find a free region
  - Initialise a block header?
- Are locks needed?
  - ???

malloc()

- Are locks needed?
  - For threads, locks are needed for most data structure manipulations in malloc()
  - Kernel needs locks for page allocation
  - Processes need no user space locks for malloc()

read()/write()

- What about file IO?
  - is file IO different in threads vs processes?
- What does an OS need to do for file IO?
  - ???

Hint: Common IO system calls

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

read()/write()

- What does an OS need to do for file IO?
    - Map from file descriptor to file structure
    - Copy data to/from page cache
    - Possibly initiate disk IO
- How do you map from a fd to a file?
    - Simple array? Tree structure?
    - Either way, it needs locking
    - With threads that can give contention

Hardware vs Software

- What about the MMU?
  - Memory Management Unit
  - Gives hardware memory protection between tasks
  - Virtualises memory addressing
- Another way to look at things
  - Threads use software to protect data structures
  - Processes use hardware to protect data structures

thread_perf

- Threads vs processes benchmark
    - http://samba.org/~tridge/junkcode/thread_perf.c
- Compares common tasks
    - malloc, setreuid, readwrite, stat, fstat, create etc.
- Thread library
    - Linking to a thread library can matter for processes!

setreuid()

- A very interesting case
    - setreuid() used to change task credentials
    - Posix requires change of all thread credentials
    - Applications often want to change just one task
- thread_perf result
    - setreuid() with threads over 200x slower on Linux
    - Why??

Memory Footprint

- The memory hierarchy matters
  - How much faster is memory than disk?
  - What about cache memory vs main memory?
- Reducing memory footprint
  - Allows more tasks to run
  - May give better use of cache
- Threads
  - Easier to pack structures in shared memory
  - Less fragmentation of memory?
  - May use less memory?

Conclusions

- Non-obvious choice
  - Threads and processes have significant differences
  - Which is best depends on the application
- Systems programming matters
  - Think about the OS level implementation!
  - Learn to use strace!