COMP 3430

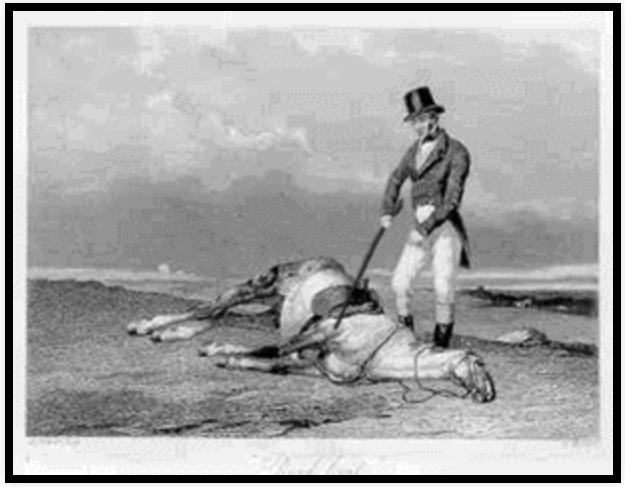
Operating Systems

May 29th, 2019

Beating threadpools to death.

Thread pools are used **extensively** in the kernel.

Optional reading: Concurrency-managed workqueues

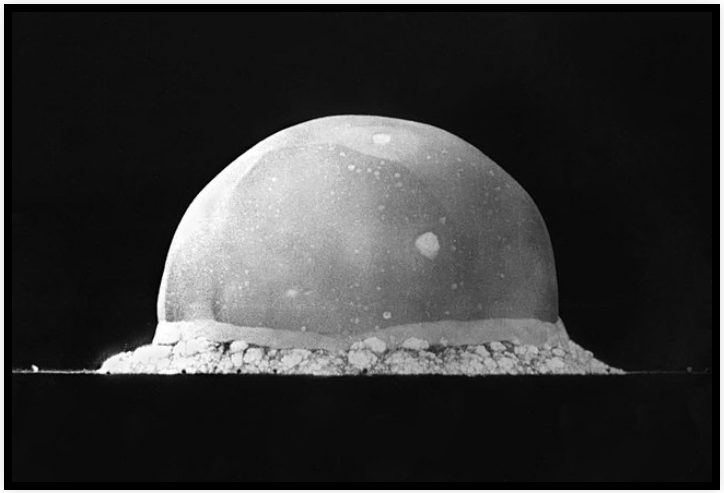


Us and thread pools.

Goals

By the end of today's lecture, you should be able to:

- Describe different strategies for message passing
- Write a UNIX program that handles signals (syscalls)
- Explain how pipes and signals can be used as a messaging mechanism
- Write a UNIX program that uses pipes for messaging

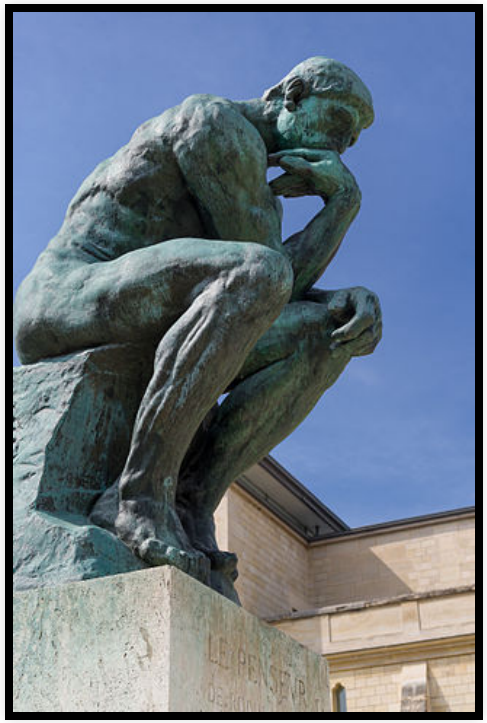


Public Domain

Something we thought about last time

How do these three approaches (based only on their names) resemble communication with *threads*?

- Signals
- Files (pipes)
- Shared memory
- Which came *first*? Communication w/ threads or communication with processes?



Signals

- A *primitive* method for communication among processes.
- Idea: We want to **do** something **when** some event happens.

Pixabay License

Signal steps

1. Process expresses interest to the kernel in observing that event.
   - Indicates what code should be executed.
2. *Different* process sends an event notification.
3. Kernel checks if anyone has expressed interest
   - If interest was expressed, calls the function.

Signals, in code

Let's take a look at a couple of code samples.

For each, let's *predict* the behaviour we'd see when it runs.

1. `fork-signal.c`
2. `signal.c`

How does that *work*?

1. Register a signal handler (see `man 2 signal`)
2. … wait for someone else to `kill` us.
   - Actually, just proceed with regular operation.
3. After a while *someone* sends a signal! (Do we know who?) (Do we care?)
   - What we *were* doing stops.
   - Our process starts executing the handler.

Pixabay License

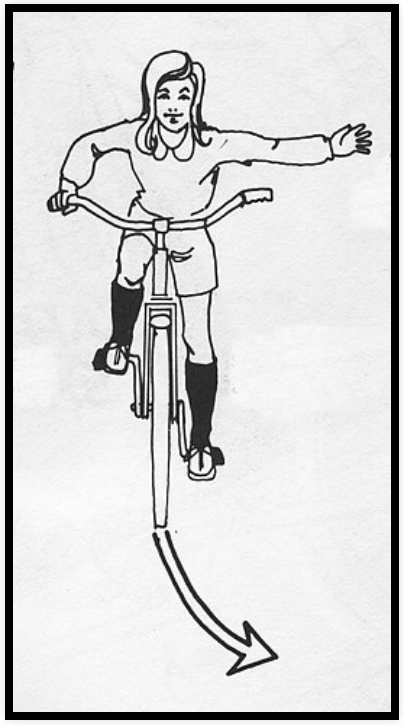
What's the kernel doing?

- The kernel *mediates* this whole process.
  - `kill` is a *system call*.
  - The kernel performs a *switch* (context? mode? …?)
- Think about it: What kind of **data structures** might the kernel use to implement signals?
  - Let's take a look at `signal.c`

Pixabay License

Signals!

- Signals are great!
  - We can send messages to our *processes* now.
  - Kernel mediates the whole process.
- … but:
  1. We're limited to the set of pre-defined signals. (… some of which we're not even *allowed* to handle)
  2. We can't send any *additional* information outside of the signal.
  3. We can't send messages to processes on *different* machines (COMP 3010).



Public Domain

Pipes

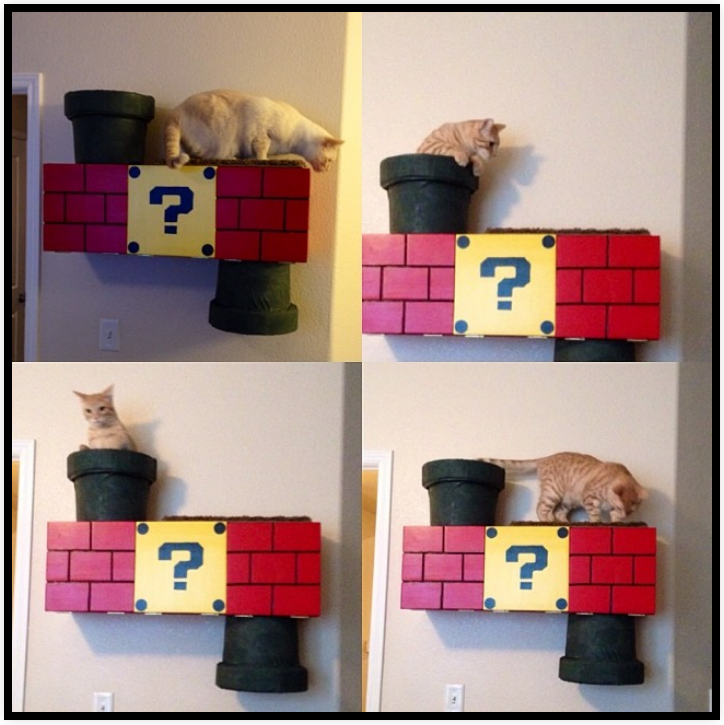
- Ok. Signals are *great*.
- A pipe is an inter-process communication tool.
- Two types:
  1. Anonymous pipes. (|)
  2. *Named* pipes.

Uh, not *those* kind of pipes. (Pixabay License)

Pipes are … *better*?

Let's take a look at some examples.

1. Anonymous pipes. (`anonymous_pipe.c`)
2. Named pipes. (`client.c` and `server.c`)



More like *this* kind of pipe. © Wes Woodward

How are pipes better?

Discussion: How are pipes *better* than signals?

What's the kernel doing?

- The kernel *mediates* this whole process.
  - The kernel performs a *switch* (context? mode? …?)
- Think about it: What kind of **data structures** might the kernel use to implement pipes?
  - Let's take a look at `pipe.c`

Pixabay License

Pipes!

Pipes are *much* more flexible than signals.

- We can actually get *information* with the message.
  - … but we still need *structure* (protocols) (COMP 3010)
- Kernel *still* mediates the whole process.
- We can send messages to processes on *other* machines (it's just a file!)
  - *Highly* similar to `sockets`.
- Pipes can be **slow** .

Shared memory

- Shared memory with threads?
  - It's just there.
- Shared memory with processes?
  - Use `mmap` (`man 2 mmap`)
- Let's look at the `man` page.

Pixabay License

Shared memory

- *Straightforward*. (kind of like `malloc`)
- Fast!
- Dangerous!
- Fast!!
- Dangerous!!!
  - Remember `pthread_mutex_init` et al?
  - Yeah. We've got the same bag of problems.

IPC

- Now we've got three methods for communicating among processes.
- One of those methods is *still* heavily file-based.
  - … but that's OK.
- These are *all* mediated by the kernel w/ system calls.
  - They *sometimes* share the same problems as threads.

"Businessman". Pixabay License

Next week

Let's take a look at the schedule.

- Most importantly: We'll answer the question "WTF does any of this have to do with scheduling?"

Pixabay License

Teacher: "Guys, this is a big project so don't wait until the night before to start."

Me, the night before: