Lab Handout 6: Threads vs Processes

Students are encouraged to share their ideas in the #lab6 Slack channel. SCPD students are welcome to reach out to me directly if they have questions that can't be properly addressed without being physically present for a discussion section.

Problem 1: Threads vs Processes

According to some CAs, a good number of students are pondering the pros and cons of threads versus processes. Some have even asked what the pros and cons are. To provide some answers, we'll lead you through a collection of short answer questions about multiprocessing and multithreading that focuses more on the big picture.

- What does it mean when we say that a process has a private address space?
- What are the advantages of a private address space?
- What are the disadvantages?
- What programming directives have we used in prior assignments and discussion section handouts to circumvent address space privacy?
- In what cases do the processes whose private address spaces are being publicized have any say in the matter?
- When architecting a larger program like `farm` or `stsh` that relies on multiprocessing, what did we need to do to exchange information across process boundaries?
- Can a process be used to execute multiple executables? Restated, can it `execvp` twice to run multiple programs?
- Threads are often called lightweight processes. In what sense are they processes? And why the lightweight distinction?
- Threads are often called virtual processes as well. In what sense are threads an example of virtualization?
- Threads running within the same process all share the same address space. What are the advantages and disadvantages of allowing threads to access pretty much all of virtual memory?

Each thread within a larger process is given a thread id, also called a **tid**. In fact, the thread id concept is just an extension of the process id. For singly threaded processes, the pid and the main thread's tid are precisely the same. If a process with pid 12345 creates three additional threads beyond the main thread (and no other processes or threads within other processes are created), then the tid of the main thread would be 12345, and the thread ids of the three other threads would be 12346, 12347, and 12348.

- What are the advantages of leveraging the pid abstraction for thread ids?
- What happens if you pass a thread id that isn't a process id to `waitpid`?

- What happens if you pass a thread id to `sched_setaffinity`?
- What are the advantages of requiring that a thread always be assigned to the same CPU?

In some situations, the decision to rely on multithreading instead of multiprocessing is dictated solely by whether the code to be run apart from the main thread is available in executable or in library form. But in other situations, we have a choice.

- Why might you prefer multithreading over multiprocessing if both are reasonably good options?
- Why might you prefer multiprocessing over multithreading if both are reasonably good options?
- What happens if a thread within a larger process calls `fork`?
- What happens if a thread within a larger process calls `execvp`?

And some final questions about how `farm`, `aggregate`, and `stsh` could have been implemented:

- Assume you know nothing about multiprocessing but needed to emulate the functionality of `farm`. How could you use multithreading to design an equally parallel solution without ever relying on multiprocessing?
- Inversely, how could you have implemented `aggregate` to rely on multiprocessing (without threading) to arrive at an equally parallel solution?
- Could multithreading have contributed to the implementation of `stsh` in any meaningful way? Why or why not?

Problem 2: Threading Short Answer Questions

Here are some more short answer questions about the specifics of threads and the directives that help threads communicate.

- Is `i--` thread safe? Why or why not?
- What's the difference between a `mutex` and a `semaphore` with an initial value of 1? Can one be substituted for the other?
- What is the `lock_guard` class used for, and why is it useful?
- What is busy waiting? Is it ever a good idea? Does your answer to the good-idea question depend on the number of CPUs your multithreaded application has access to?
- As it turns out, the semaphore's constructor allows a negative number to be passed in, as with `semaphore s(-11)`. Identify a scenario where -11 might be a sensible initial value.
- What would the implementation of `semaphore::signal(size_t increase = 1)` need to look like if we wanted to allow a `semaphore`'s encapsulated value to be promoted by the `increase` amount? Note that `increase` defaults to 1, so that this version could just

replace the standard `semaphore::signal` that's officially exported by the `semaphore` abstraction.

- What's the multiprocessing equivalent of the `mutex`?
- What's the multiprocessing equivalent of the `condition_variable_any`?