RECITATION 12

INFO

- Jon Rutkauskas
- Recitation: Tue 12-12:50
- Office Hours: Tue 11-11:50
  Thur 11-12:50
  SENSQ 5806
  (additional hours by appointment if needed)
- On discord: @jrutkauskas
- By email: jsr68@pitt.edu
- Website:
  https://github.com/jrutkauskas/spring2019-449-rec
- Ask me any questions you have!!!

WARMUP POLL

“STICK” AROUND…

- Still have stickers!

Which of these is NOT a valid use case for threads.

A program makes multiple threads to run after the main process ends, keeping a daemon alive

1 Thread to run a GUI while 1 manages data in the background

A program makes multiple threads to handle slow I/O operations

A program makes multiple threads to split up a workload between them

Which of these is NOT a valid use case for threads.

1. When a PROCESS ends, its threads end with it.
   - When the main thread ends, you might be able to keep the threads from dying by using "Joining" with those threads, but if a process is dead, so are its threads, generally
2. The rest are all valid use cases for interactive programs, handling data, slow IO operations, or splitting up workloads.

A program makes multiple threads to run after the main process ends, keeping a daemon alive

1 Thread to run a GUI while 1 manages data in the background

A program makes multiple threads to handle slow I/O operations

A program makes multiple threads to split up a workload between them

Which of the following is NOT true about threading?

User threading can be done regardless of kernel or hardware support.

User threading relies on collaborative scheduling, which is usually not a problem.

Kernel threading is faster than user threading because it takes advantage of the special features of the kernel

Kernel threading lets you run multiple threads at the same time.

Hardware accelerated threading duplicates some of the parts of the CPU to increase the speed of threading

Which of the following is NOT true about threading?

1. User threading is done by a library that runs in userspace and handles threading. To the kernel, it's just one process running, it just jumps around in its execution.
2. Since user-mode processes can't use interrupts, they rely on collaborative scheduling, that is, threads willingly yielding control back to other threads. This is usually not a problem since all your threads are typically running only code you wrote, so just write good code. The process holding them all is still scheduled normally using the kernel
3. Kernel threading is slower because switching between threads now requires context switching.
4. Kernel threading DOES let you run multiple threads at the same time, though, on multiple CPU cores.
5. H/W accelerated threading will have some of the CPU components, like registers, duplicated so when we switch threads, instead of saving all those registers and clearing out the other parts, we can just switch which parts we're using.

User threading can be done regardless of kernel or hardware support.

User threading relies on collaborative scheduling, which is usually not a problem.

Kernel threading is faster than user threading because it takes advantage of the special features of the kernel

Kernel threading lets you run multiple threads at the same time.

Hardware accelerated threading duplicates some of the parts of the CPU to increase the speed of threading

LAB 8: BASIC MULTITHREADING

- Making a program to set alarms for n seconds from now
- Follows very closely the code in the example files
- Plus, all the user interaction is already written for you.
- Honestly not a lot of code, mostly just knowing what to copy and modify, and the basic concept of the program.
- Ask me about mutexes, since you haven't talked about them in class yet.