ECE 344: Operating Systems

Lecture 18

Quiz 2 Review

1.0.0

Jon Eyolfson

October 20, 2022

A Forking Question

Consider the following code:

```
int main() {
  pid_t first = fork();
  pid_t second = fork();
  pid_t third = fork();
  printf("first=%d second=%d third=%d\n", first, second, third);
}
```

What is one reasonable set of outputs (assume the initial process is pid 2)?

What order are the outputs in?

What do the relationships between processes look like?

A Threading Question

Assume you have a global variable, **`int`** `i;` and 3 threads.

Before any thread executes, `i` is initialized to `0`

Two threads execute `i++` and one executes `i--`

What are all the possible values of `i` after all threads execute?

Unix Systems Clone Processes with a Parent/Child Relationship

- You can only create new processes with `fork`
- After a `fork` both processes are exactly the same
    - except for the value of `pid` (the child is always 0)
- The scheduler decides when to run either process

You’re Responsible for Managing Processes

The operating system maintains a strict parent/child relationship

You should be able to identify (and prevent) the following:

- Zombie processes
- Orphan processes

For quiz 2: the focus would be more on `wait`

We Explored Basic IPC in an Operating System

Some basic IPC includes:

- `read` and `write` through file descriptors (could be a regular file)
- Redirecting file descriptors for communcation
- Signals

Signals are like interrupts for user processes

The kernel has to handle all 3 kinds of "interrupts"

For quiz 2: this isn't covered

Threads Enable Concurrency

We explored threads, and related them to something we already know (processes)

- Threads are lighter weight, and share memory by default
- Each process can have multiple threads (but just one at the start)

Both Processes and (Kernel) Threads Enable Parallelization

- Each process can have multiple (kernel) threads
- Most implementations use one-to-one user-to-kernel thread mapping
- The operating system has to manage what happens during a fork, or signals
- We now have synchronization issues

We Want Critical Sections to Protect Against Data Races

We should know what data races are, and how to prevent them:

- Mutex or spinlocks are the most straightforward locks
- We need hardware support to implement locks
- We need some kernel support for wake up notifications
- If we know we have a lot of readers, we should use a read-write lock

We Used Semaphores to Ensure Proper Order

Previously we ensured mutual exclusion, now we can ensure order

- Semaphores contain an initial value you choose
- You can increment the value using post
- You can decrement the value using wait (it blocks if the current value is 0)
- You still need to be prevent data races

We Explored More Advanced Locking

We have another tool to ensure order

- Condition variables are clearer for complex condition signaling
- Locking granularity matters
- You must prevent deadlocks

Final Questions or Concerns?

Ashvin and I will be on Discord, we'll announce any issues (hopefully none!)

Format: 2 true/false, 2 multiple choice, 2 short answer (free form)

Expect to take around 47 minutes total:

1 minute for true/false,

10 minutes for each multiple choice,

10 minutes for the first short answer,

and 15 minutes for the final short answer.

Good luck!