2023 Fall ECE 344: Operating Systems

Lecture 24 1.0.1

Midterm Review

Jon Eyolfson

2023 Fall

There are 3 Major Concepts in This Course

You'll learn how the following applies to operating systems:

- Virtualization
- Concurrency
- Persistence

Kernel Interfaces Operate Between CPU Mode Boundaries

The lessons from the lecture:

- Code running in kernel mode is part of your kernel
- System calls are the interface between user and kernel mode
    - Every program must use this interface!
- File format and instructions to define a simple “Hello world” (in 168 bytes)
    - Difference between API and ABI
    - How to explore system calls
- Different kernel architectures shift how much code runs in kernel mode

Operating Systems Provide the Foundation for Libraries

We learned:

- Dynamic libraries and a comparison to static libraries
    - How to manipulate the dynamic loader
- Example of issues from ABI changes without API changes

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

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”

Scheduling Involves Trade-Offs

We looked at few different algorithms:

- First Come First Served (FCFS) is the most basic scheduling algorithm
- Shortest Job First (SJF) is a tweak that reduces waiting time
- Shortest Remaining Time First (SRTF) uses SJF ideas with preemptions
- SRTF optimizes lowest waiting time (or turnaround time)
- Round-robin (RR) optimizes fairness and response time

Scheduling Gets Even More Complex

There are more solutions, and more issues:

- Introducing priority also introduces priority inversion
- Some processes need good interactivity, others not so much
- Multiprocessors may require per-CPU queues
- Real-time requires predictability
- Completely Fair Scheduler (CFS) tries to model the ideal fairness

Page Tables Translate Virtual to Physical Addresses

The MMU is the hardware that uses page tables, which may:

- Be a single large table (wasteful, even for 32-bit machines)
- Use the kernel allocated pages from a free list
- Be a multi-level to save space for sparse allocations
- Use a TLB to speed up memory accesses

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

A Forking Question

Consider the following code:

```c
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)?

Are the outputs in any specific order?

What do the relationships between processes look like?

Example Midterms

This is the style of midterm I typically write:
https://eyolfson.com/media/courses/utoronto/ece353/2023-winter/midterm.pdf
https://eyolfson.com/media/courses/ucla/cs111/21fall/midterm.pdf