CS 111: Operating System Principles

Lecture 18

Course Recap

3.0.0

Jon Eyolfson

December 2, 2021

There are 4 Major Concepts in This Course

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

- Virtualization
- Concurrency
- Persistence
- Security (out of scope, somewhat touched on in Virtual Machines)

Kernel Interfaces Operate Between CPU Mode Boundaries

The lessons from the lecture:

- Code running in kernel mode is part of your kernel
- Different kernel architectures shift how much code runs in kernel mode
- System calls are the interface between user and kernel mode
- Everything involved to define a simple “Hello world” (in 178 bytes)
  - Difference between API and ABI
  - How to explore system calls

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
- Standard file descriptor conventions for UNIX

The Operating System Creates and Runs Processes

The operating system has to:

- Loads a program, and create a process with context
- Maintain process control blocks, including state
- Switch between running processes using a context switch
- Unix kernels start an `init` process
- Unix processes have to maintain a parent and child relationship

We Used System Calls to Create Processes

You should be comfortable with:

- `execve`
- `fork`
- `wait`

This includes understanding processes and their relationships

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
- Pipes (which you'll explore)
- Signals
- Shared Memory

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)
- Be a multi-level to save space for sparse allocations
- Use the kernel allocate pages from a free list
- Use a TLB to speed up memory accesses

Page Replacement Algorithms Aim to Reduce Page Faults

We saw the following:

- Optimal (good for comparison but not realistic)
- Random (actually works surprisingly well, avoids the worst case)
- FIFO (easy to implement but Bélády’s anomaly)
- LRU (gets close to optimal but expensive to implement)
- Clock (a decent approximation of LRU)

Both Processes and (Kernel) Threads Enable Parallelization

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 (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 Explored More Advanced Locking

Before we did mutual exclusion, now we can ensure order

- Semaphores are an atomic value that can be used for signaling
- Condition variables are clearer for complex condition signaling
- Locking granularity matters, you’ll find out in Lab 3
- You must prevent deadlocks

The Kernel Has To Implement It’s Own Memory Allocations

The concepts are the same for user space memory allocation
(the kernel just gives them more contiguous virtual memory pages):

- There’s static and dynamic allocations
- For dynamic allocations, fragmentation is a big concern
- Dynamic allocation returns blocks of memory
    - Fragmentation between blocks is external
    - Fragmentation within a blocks is internal
- There’s 3 general allocation strategies for different sized allocations
    - Best fit
    - Worst fit
    - First fit
- Buddy allocator is a real world restricted implementation
- Slab allocator takes advantage of fixed sized objects to reduce fragmentation

Disks Enable Persistence

We explored two kinds of disks: SSDs and HDDs

- Magnetic disks have poor random access (need to be scheduled)
- Shortest Positioning Time First (SPTF) is the best scheduling for throughput
- SSDs are more like RAM except accessed in pages and blocks
- SSDs also need to work with the OS for best performance (TRIM)
- Use RAID to tolerate failures and improve performance using multiple disks

Filesystems Enable Persistence

They describe how files are stored on disks:

- API-wise you can open files, and change the position to read/write at
- Each process has a local open file and there’s a global open file table
- There’s multiple allocation strategies: contiguous, linked, FAT, indexed
- Linux uses a hybrid inode approach
- Everything is a file on UNIX, names in a directory can be hard or soft links

Distributed Systems Start with Sockets

There’s networking and distributed systems courses!

However, today we learned the basics:

- Sockets are IPC across physical machines
    - Sockets require an address (e.g. local and IPv4/IPv6)
    - There are two types of sockets: stream and datagram
    - Servers need to bind to an address, listen, and accept connections
    - Clients need to connect to an address
- Networked file systems (NFS)
- Distributed file systems (GFS)
- Denial-of-service attacks are a unique concern

Virtual Machines Virtualize a Physical Machine

They allow multiple operating systems to share the same hardware

- Virtual machines provide isolation, the hypervisor allocates resources
- Type 2 hypervisors are slower due to trap-and-emulate and binary translation
- Type 1 hypervisors are supported by hardware, IOMMU speeds up devices
- Hypervisors may overcommit resources and need to physically move VM
- Containers aim to have the benefits of VMs, without the overhead

Thank you!