ECE 353: Systems Software
Lecture 1

# Overview

1.1.0

Jon Eyolfson
January 9, 2023

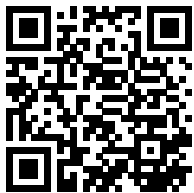
# Hello, I’m Jon

# Quercus is the Main Hub

You'll find links on Quercus for everything

We'll primarily use Quercus for scheduling announcements and grades

Materials posted to: https://eyolfson.com/courses/ece353/



# Discussion Will Be Interactive on Discord

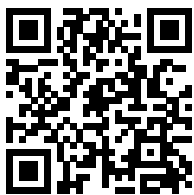
Discord

https://discord.gg/p5SUTrNa9P

# Code Submission Will Be Through GitLab

GitLab

https://laforge.eecg.utoronto.ca/



# Lectures Will Be Streamed with VODs


https://www.youtube.com/@eyolfson

# Lecture Attendence is Still Extremely Important, for Both of Us

Please still attend lectures if you're not sick!

This is the easiest and most direct way to get feedback

Feel free to raise your hand or ask questions at any time if anything is unclear

# Office Hours are Wednesdays 1 PM to 2 PM

Let me know if there's any conflict

The TAs and I will be on Discord for discussions

Office hours aren't streamed or recorded; this is a resource only for you

# There are 6 Labs, a Midterm, and a Final Exam

| Assessment | Weight | Due Date |
|---|---|---|
| Lab 1 | 8% | January 30 |
| Lab 2 | 8% | February 13 |
| Lab 3 | 8% | February 27 |
| Midterm Exam | 17% | March 1 (tentatively) |
| Lab 4 | 8% | March 13 |
| Lab 5 | 8% | March 27 |
| Lab 6 | 8% | April 10 |
| Final Exam | 35% | April 18 to April 30 |

# Academic Integrity Includes Not Posting on GitHub

You can study together; you cannot do labs together

Cheating hurts everyone involved

Any cheating will not be tolerated

# The Books are Optional, and Complement Lectures

"Operating Systems: Three Easy Pieces"
by Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau

"The C Programming Language"
by Brian Kernighan and Dennis Ritchie

# Please Provide Feedback

I want you to get the most out of this course

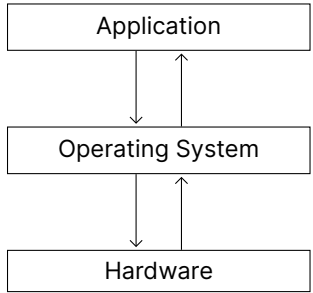
Let me know if you like, dislike, or want to see more of something

I’m open to suggestions!

“All problems in computer science can be solved by another level of indirection”

- David Wheeler

# An Operating System Sits between Applications and Hardware




The primary role of an operating system is to manage and coordinate resources

# Ubuntu and Android are Considered Different Operating Systems

Both use a Linux kernel, but they run different applications

There isn't a clear line, especially with "Linux"

For desktop applications, you'd draw the line at the Display System

"Linux" uses Wayland, and Android uses SurfaceFlinger

# Operating Systems Allow Running More than One Application

Without an operating system, a CPU starts executing code at a fixed address

You could put your application here, but it would be the only one

You would have to handle specific hardware in your application

Instead, we start executing an operating system at boot

# Our First Abstraction is a Process

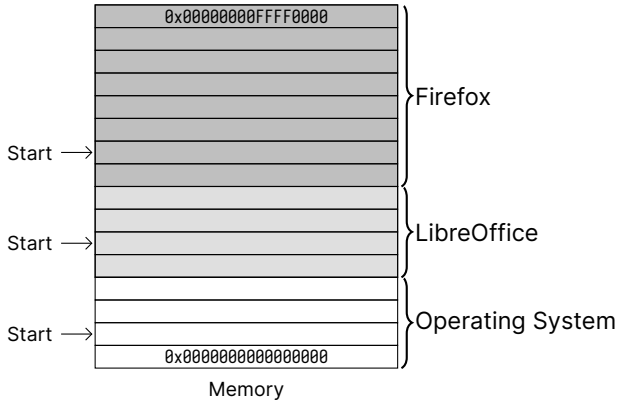
Each process contains its own set of registers, including the program counter

When starting a process, it specifies where the CPU should start executing

The operating systems has to:

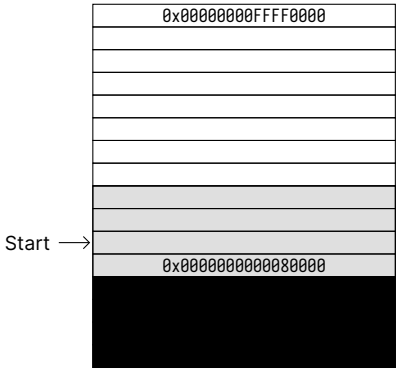
- Keeps track of registers for each process
- Switch between different processes
- Decide when to switch between processes

# We Could Put Applications in Different Parts of Memory

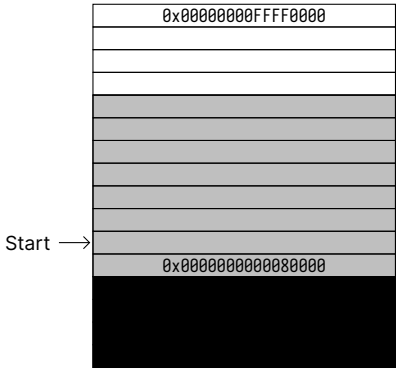



This isn't very flexible

# Virtualization Fools Something into Thinking it Has All Resources




“LibreOffice Memory”




“Firefox Memory”

# Virtual Memory Abstracts Away Physical Memory

Each process believes it has access to all the memory

Different processes can have the same starting address

The operating system has to:

- Map virtual memory access to physical memory
- Keep track of memory usage (allocate and deallocate)
- Handle out-of-memory scenarios

# Virtualization is a Powerful Concept

Applies to both processes and virtual memory

We can extend this to an entire machine

A single physical machine can run multiple operating systems at once

# Concurrency is Multiple Things Happening at the Same Time

We want multiple applications running at once

We want applications to do multiple things at once

We don’t want applications isolated

We want applications and libraries to communicate

# Concurrency is Necessary for Operating Systems

Running one application at a time isn't a good experience

Completely isolated applications aren't useful

The simplest applications still communicate with the terminal

The operating system has to:

- Allow multiple executions at once, safely
- Manage abstractions for different kinds of inter-process communication (IPC)
- Provide permission checking and access control

# Finally, We Need Persistence for a Basic Operating System

We want to be able to access data between boots

A file system specifies how to organize data on a storage medium

The operating system has to:

- Store and retrieve data
- Ensure integrity of data

# File Descriptors Abstract Both Communication and Persistence

A file descriptor is just a number identifier (per process) that you can:

- Read bytes from
- Write bytes to

The operating system can direct the bytes to whatever it represents

You could imagine it representing a file, or one side of communication

# Code Example: `01-overview/read-four-bytes.c`

We'll have a bunch of code examples in this class

Compile the examples:

```
cd 01-overview
meson setup build
meson compile -C build
```

Execute the example:

```
build/read-four-bytes /usr/bin/ls
```

You'll need be familar with this kind of code for the course

# Most Kernel Code is Device Drivers

Device drivers implement the abstractions we'll learn to the physical hardware

It involves reading manufacturer specifications, and finding bugs

Sometimes there's inconsistencies between documentation and reality

# An Actual Comment Linux Source (arch/x86/kernel/apm_32.c)

```
/*
 *  Check for clue free BIOS implementations who use
 *  the following QA technique
 *
 *      [ Write BIOS Code ]<------
 *               |                ^
 *      < Does it Compile >----N--
 *               |Y               ^
 *      < Does it Boot Win98 >-N--
 *               |Y
 *           [Ship It]
 *
 *      Phoenix A04  08/24/2000 is known bad (Dell Inspiron 5000e)
 *      Phoenix A07  09/29/2000 is known good (Dell Inspiron 5000)
 */
```

# Believe It or Not, This Is “Hello World”

```
0x7F 0x45 0x4C 0x46 0x02 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x02 0x00 0xB7 0x00 0x01 0x00 0x00 0x00 0x78 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x40 0x00 0x38 0x00 0x01 0x00 0x40 0x00 0x00 0x00 0x00 0x00
0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x08 0x08 0x80 0xD2 0x20 0x00 0x80 0xD2
0x81 0x13 0x80 0xD2 0x21 0x00 0xA0 0xF2 0x82 0x01 0x80 0xD2 0x01 0x00 0x00 0xD4
0xC8 0x0B 0x80 0xD2 0x00 0x00 0x80 0xD2 0x01 0x00 0x00 0xD4 0x48 0x65 0x6C 0x6C
0x6F 0x20 0x77 0x6F 0x72 0x6C 0x64 0x0A
```

# There are 3 Major Concepts in This Course

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

- Virtualization
- Concurrency
- Persistence