CS 326: Operating Systems

Lecture 1

Welcome to CS 326!

- Looking forward to a great semester!
- *Lecture:* **MWF** · 10:30am – 11:35am · **LM** 244A
- *Lab Session:* **Monday** · 11:40am – 12:45pm · **CO** 413

Staff

- **Instructor**: Matthew Malensek
  - mmalensek@usfca.edu
  - Office Hours: **W, F** 11:45am – 12:45pm in **HR** 407B
- **TA**: Jaz Ku
  - Office Hours: **T, Th** 6:00pm – 7:00pm (via Zoom)

So… What's an OS?

- Chances are, you already know of at least one operating system…
  - …or even a few…
- …but what do they do? And more importantly, what's the point of having one? Why should we even care?

Operating Systems

- The OS abstracts the underlying hardware and exposes a clean, easy-to-use interface to software applications
- What's the point of having one?
  - Without the OS, you don't have things like:
    - Multitasking
    - Hardware drivers
    - File systems
- Why should we even care?!
  - If you understand how the OS works, you will work more efficiently, and the ***software you write for it*** will be better!

Today's Agenda

- Syllabus
- Introduction to Operating Systems

Today's Agenda

- **Syllabus**
- Introduction to Operating Systems

Quick Note: Prerequisites

- CS 220 or CS 221 with a C or better
  - If it's been a while since you've programmed in C, plan on spending some time refreshing your memory!
- CS 245 with a C or better
  - Strictly enforced

Updates & Communication

- I will regularly update the **schedule** page on the course website
  - https://www.cs.usfca.edu/~mmalensek/cs326
  - Upcoming topics, readings, due dates
  - Lecture slides, videos
- Grades will be posted on Canvas
- Project submissions: GitHub
- If you need help: CampusWire
  - Q&A link on the website takes you there

You Will Learn

- How to work effectively from the command line
- Operating system components and their implementations
  - Threads, processes, synchronization primitives, system calls, scheduling, virtual memory, file systems
- How to develop UNIX systems software
- UNIX tools
- Debugging complex systems and low-level software

Course Overview

1. Command line, C refresher
2. Booting up and processes
3. System calls
4. Inter-process communication
5. CPU Scheduling
6. Memory management
7. Disk I/O
8. Networking, containers, distributed systems, etc…

Books

- *xv6: a simple, Unix-like teaching operating system*. Russ Cox, Frans Kaashoek, and Robert Morris.
- *Operating Systems: Three Easy Pieces*
  - Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
  - Free and available online as a PDF!
    - http://pages.cs.wisc.edu/~remzi/OSTEP
- *Extreme C*. Kamran Amini.
- *The C Programming Language, 2nd Edition*. Brian W. Kernighan and Dennis M. Ritchie

Course Structure

- There's a decent amount of theory in CS 326, but we'll spend a lot of time applying it in practical situations
  - We'll be writing low-level code and working with the command line extensively
- Assignments: labs and ~4 **large** projects
- Assessment: quizzes every ~3 weeks
- And there's also a lab session on Mondays…

Lab Session

- Lab time gives us the opportunity to work on labs/projects as a class and get feedback
- We'll review concepts as a class, answer questions, check your lab assignments, etc.

Grade Distribution

- Labs: 30%
- Quizzes: 30%
- Projects: 40%

Lab Assignments

- Small assignments that we start or complete in class
- Grading is fairly simple, you either get credit or you don't.
- Use class time, lab time, or office hours to check your work.

Quizzes

- Given roughly every three weeks
- Helps us go back and review what we've covered
- The strategy to do well is:
  1. Take note of concepts that are demoed in class. These are most important.
     - Sometimes questions are even revealed in advance…
  2. Review the slides.
  3. If something isn't clear, review the lecture videos and book chapters

Group Quiz

- After the regular quiz, you'll take it again
  - this time as a group (probably 4-5 students)
- Good way to review and discuss the problems
- Afterward, your group will prepare project questions to discuss
  - (assuming we have a project going on at the time… which we probably will 😉)

Final ~~Exam~~ Quiz

- There is no final exam, but there ***is*** a final quiz
  - Slightly larger than the usual quizzes
- Review only (no new material)
- I will drop your lowest quiz score
  - Do well throughout the semester and you won't have to take the final quiz!

Projects

- CS 326 projects are large and time-consuming
  - Start them early
- Remember, C can be much more difficult to work with
  - You might only need to implement, X, Y, and Z. Really, that's not tooSEGFAULT SEGFAULT SEGFAULT
    - BUS ERROR
- *Plan* out your design, work methodically, and debug as you go

Academic Honesty

- **Do not cheat**. Review the Honor Code, and if in doubt about whether or not something is cheating, ask the professor.
  - The course staff will run cheat detection software that includes past assignments.
  - "Collaboration" that involves sharing code/solutions is considered cheating.
  - If you cheat, **you will get a 0 on the assignment or an F in the class**.
- Submit code via GitHub. Commit your changes frequently as you work on the assignments.

Today's Agenda

- Syllabus
- **Introduction to Operating Systems**

Your OS

- You'll be working on your own OS, based on **xv6**
  - ...more on that later!
- It will target RISC-V CPUs and run on our virtualization/emulation server `gojira`
- So in other words, you'll be running your OS on top of another OS (Linux)

gojira

- Dual 16-core AMD Epyc CPUs
  - 32 cores
  - 64 hardware threads
- 128 GB of RAM
- Laser-guided missile defense system
- Dual cupholders

Operating Systems Survey

- I mentioned earlier that you all probably know of a few OS. Let's see how many we've all used.
- Dust off your CS credentials and log into stargate:
  - `ssh username@stargate.cs.usfca.edu`
- Next, `ssh gojira`
- If your login credentials don't work, it's time to remember them, or email support@cs.usfca.edu
- Ok, let's create our list of operating systems…

OS List

- `cd` to `/tmp/cs326`
  - I will create this now
- Create a file named after your CS username with:
  - `touch $(whoami)`
- Open it in an editor (`vim` , `nano` , `emacs` , …)
- List each OS you've used, one per line
  - If you want to explain what you used it for, add a comment after it with `#`

An Example

```
MS-DOS
Windows 3.11   # Trying to impress my friends
Windows 95
Windows 98     # Mostly for Space Cadet Pinball
Windows 10     # After my friends were not impressed
macOS 11
```

Next

- After you're finished, see if you can figure out how to:
  1. View who is logged into `gojira`
  2. See the ***processes*** currently running on the machine
- Then we'll do some thorough and very scientific analysis on the files to see what kind of OS users this class has

Wow Nice Job, Matthew 

- Well, that was pointless
  - or was it?
- We just saw:
  - UNIX's multi-user capabilities
  - Remote logins
  - The command line
  - File creation, modification
  - Shell pipelines
  - Permissions handling
  - And a whole bunch of processes running it all

Operating Systems

- Let's get back to business. What **is** an OS?
  - An OS provides a layer of abstraction between your computer's hardware and software
- Computer science is all about layers of abstraction
  - Nobody wants to program in binary, so we have assembly
    - Nobody wants to program in assembly, so we have C
      - Nobody wants to manage their own memory, so there's Java
        - Nobody wants to write another AbstractFactoryBuilderPattern, so there's Python

Virtualization

- The basic concept of OS is **virtualization**
- Rather than exposing the hardware directly to programs, a nice set of virtual interfaces are provided
  - Razer Viper Ultimate Mouse ➡ USB Mouse
  - Samsung Evo 1 TB SSD ➡ Mass storage device
  - etc.
- Programs don't have to worry about what's going on in the system
  - Appear to have their own CPU, memory, etc.

Our Focus

- This course will cover two domains:
  - User space – most OS include bundled programs. These really aren't much different than programs we've already written
  - Kernel space – provides the functionality that makes **virtualizing** the hardware possible
    - Exposes an API to user space: **system calls**
- We'll write code for both!

Wrapping Up

- Welcome to class (again!)
  - Ask questions, and come to instructor/TA office hours. We're here to make sure you succeed.
  - Don't take the projects in this course lightly (especially if you haven't taken a 300-level yet)
- We do have one last thing to take care of today: **Lab 0**
  - Find it on the assignments page here:
  - https://www.cs.usfca.edu/~mmalensek/cs326/