Computer Systems
Fall 2020
Course Information

Prof. Kaplan

Be sure to read **all** of this document!

1 Introduction

What is this course about? When you learn how to program, you use a complex set of existing tools to compile and run the code that you write. What are those tools? How does the code you write get carried out? Even the simplest of programs (e.g., *Hello World*) triggers a number of complex mechanisms to complete its task.

This course seeks to reveal how a general-purpose computer system can execute the programs you write (and those that others write). The design and implementation of this type of system requires a thoughtful structuring of a number of *layers of abstraction*, each of which simplifies programming and handles complex problems automatically. The goal of a computer system is to *enable programmers to solve their problems*. That is, the programmer should be able to concentrate on the problem at hand, and not have to worry neither about the details of what the underlying system is doing, nor how it is doing it.

Why do systems matter? When you write a program for a "regular" system (e.g., your laptop running a typical operating system), do you really need to know *how* that programming language and operating system work? I can drive a car without being an auto mechanic, so why is it important to know how computer systems work?

No abstraction is perfect. In using any programming language and system, you are relying on a great deal of structure whose behavior may affect how your code behaves in ways you don't expect. Whether you're working in cryptography, graphics, AI/machine learning, databases, or just about any other computing topic, you will work with code that relies on some system. At some point, *your code will not behave the way you expected*. It will be slow, or consume too much memory, or exhibit strange bugs, and thus become an obstacle to the work you are trying to do.

In those moments, understanding how the system is structured, and understanding how the layers of the systems interact with one another, is essential to understanding what is wrong and how to fix it. A strong familiarity with these topics not only allows you to fix misbehaving or broken code, but also allows you to structure it correctly in the first place. Perhaps even more importantly, knowing how systems are structured can make clear *why* elements of a system or language are structured the way they are. Holding the interior logic and design of any system layer improves your ability to use it effectively, and to create your own layers.

And if you want to be a master hacker, knowing how to intentionally *misuse* abstractions and implementations within a system is *the* name of the game.

How will the course progress? We will start at a low level—individual *processor instructions*—and build our way, layer by layer, to full individual systems, and then modern *distributed* and *cloud* layers that allow programs to use the massive computational and memory capacity of many computers. Along the way, we will see repeated problems and themes that recur at many levels. We will also address the problem of *measuring and comparing* systems for various characteristics (e.g., speed). By the course's end, although you will hardly know every detail of a computer system, you will have enough experience to learn about and reason through the details of any system and its various layers.

2 The topics

The basic topics are given below, roughly in the order that will we cover them. If you have no idea what some (or many) of them are, don't be alarmed—that is why you're taking this course, after all.

- **Instruction set architectures**: Programming to the processor's circuits.
- **Caching and the memory hierarchy**: How are programs and their data stored to maximize performance?
- **Allocators and garbage collectors**: What happens when a program needs more memory? And then doesn't?
- **Operating systems and runtime libraries**: Programming to large collections of pre-made code.
- **Virtual memory**: How can multiple programs share the computer's memory without clobbering one another?
- **File systems**: How can data be stored for later?
- **Virtual machines**: Wheels within wheels…
- **Distributed computing**: Clusters, map/reduce, and other ways of spreading your computation.

These topics are what we directly will be covering, but underlying it all will be the concepts of *interface*, *abstraction*, *implementation*, and *resource allocation*. As we move from topic to topic, I will come back to this bigger picture to highlight these recurring themes.

This course will be project-intensive. Much of the material will seem easy enough to comprehend when presented in class, but the only way to understand this material thoroughly is to use it. In this case, *using* these ideas requires that you understand an existing implementation of a layer, and then modify or enhance it. Your projects will require you to understand existing code before you then write your own.

3 Lectures, labs, and help

Lecture/discussion times: Our class meetings will be on Zoom. Here are the times and links for each section:

- **Section 01: MWF** at **12:40 pm to 1:30 pm**. [Zoom link]
- **Section 02: MWF** at **5:10 pm to 6:00 pm**. [Zoom link]

You are expected to be present for **all class meetings**. I will not teach material twice, so if you miss a class meeting, then you're on your own for whatever material was covered that day. Note that each lecture will be recorded and shared with you on Google Drive. These recordings are meant for review, unusual circumstances, and to assist students in distant time zones; it is *not* an invitation to skip lectures.

Individual meetings (a.k.a., *office hours*): If you seek assistance, reinforcement, review, or other opportunities to discuss the course material or assignments, you should see me. There is a link on the course web page for scheduling a time to meet. I encourage you to use these hours to delve deeping into the material and the projects; chatting with me outside of class is one of the reasons you came to a small college.

TA help sessions: There will also be **TA help sessions**. These will also be held via Zoom, and they will be scheduled before the semester begins.

Slack channel: In addition to using Zoom for class meetings, office hours, and other real-time discussions, we will also make regular use of Slack. Once re-registration is complete, you wiill be added to the `#cosc-171` Slack channel, where you will be able to send questions directly to me, to TA's, or for the whole class to see. It will also be the mechanism by which I distribute announcements quickly, send files/documents that may be immediately helpful, and try to keep a running sequence of questions and answers.

Email: Many questions and issues are best addressed via the asynchrony of email, so feel free to contact me at `sfkaplan@amherst.edu` with your questions or concerns. Be forewarned, however, that I do not typically respond to email quickly, so do not expect a quick turnaround. For a quicker response, Slack is likely to be better.

4 Texts and materials

The textbook for this course is, *Computer Systems: A Programmer's Perspective, 3rd edition*, by Byrant and O'Hallaron.[1] All other tools for this course—all of the software and documentation—will be provided.

[1] ISBN-13: 978-0134092669

5 Assignments, deadlines, and extensions

There will be a number of programming projects. The deadline for each will be stated clearly on the assignment. **Late submissions may receive failing grades**. Turn in what you have, and do so on time.

An extension for any assignment **must be requested, in writing** (email counts as *writing*), **at least 48 hours prior to the deadline**. The determination as to whether or not a particular situation merits an extension will be made on a case-by-case basis. Scheduled events are **not** sufficient reason to warrant an extension. Rather, extensions are intended for unusual circumstances that prevent you from planning your time well in order to meet the deadline. Note that a sudden onset of illness or other emergency situation that occurs less than 48 hours before a deadline will be treated as a special case.

6 Exams

There will be **one mid-term exam** given during a regular class lecture hour; there will also be a **comprehensive final exam** given during the final exam period at the semester's end. The mid-term exam will be given during week 7 of the semester (with the specific day of that week to be announced); the final exam will be self-scheduled.

7 Grading

Your final grade will be chosen by my evaluation of how well you have mastered the course material at the semester's end. All of the work that you submit, as well as your participation in class, contributes to my impression of that mastery.[2]

8 Academic dishonesty

You will be expected to do your own work on all assignments and exams in this course. While I encourage you to interact with your classmates and discuss the material and assignments, there is a limit to the specificity of such discussions. I seek to make that limit clear here.

It is acceptable to discuss any assignment for the class with a classmate. You may even discuss your approach to a particular problem, or review relevant material for a problem with another person. However, you **may not show another student your work, nor see another student's work. If in doubt, *ask me***. If you are unusure whether or not a particular kind of communication would rise to the level of academic dishonesty, then you should contact me immediately and find out.

[2]This description of the grading policy makes some students break out in a flop sweat. Don't panic. The intent is for me to allow you to show, however you show it best (i.e., through projects vs. exams), your grasp of the material. Rather that try to invent a formula that captures that flexibility, it is more direct and honest to state that I will be looking for your depth of understanding and capability with the topics, and your grade will reflect what you've shown.