SIENA*college*
Computer Science

Computer Science 330
Operating Systems
Siena College
Fall 2022

Topic Notes: Introduction and Overview

Welcome to Operating Systems!

What do you think of when you talk about an operating system? ("I installed a new operating system", "Windows is my least favorite operating system", "That must be a bug in the operating system", "What operating system is on your phone?")

What do you expect to learn in a course about operating systems?

Operating system topics are always in the news – there are daily developments in the operating system world. Things change quickly. This course has been presented in different contexts each time I've taught it, as new hardware and operating systems are in fashion, though the concepts remain the same.

Where This Fits In

You've seen enough computer science to realize that there are many layers of *abstraction* involved in any computer system. There are the basics of the physics of circuits up through logic gates, circuits that can compute, construction of memory, right on up through high level languages.

You learned high-level language programming in your introductory programming sequence up through data structures and, for most of you, beyond.

Many of you have learned about hardware and assembly language and the rest are doing so this semester in CSIS-220, or an equivalent. Those who have taken that course have a notion of how to get from circuits to CPUs and memory.

A course in compilers and/or programming languages would teach you about how high-level languages let you program the hardware in a more convenient way.

Many of the things that fit between those (compiled) high-level language programs and the hardware are topics for this course.

We want to think about what it takes to get from the basic hardware you study in computer organization courses to the multi-user systems we are used to on modern computers.

A computer system is made up of a collection of resources, such as processors, memory, disks, a keyboard, printers, and network interfaces.

The operating system attempts to regulate the use of these resources for efficiency, fairness when multiple users or processes want to use them, and safety to make sure multiple users don't interfere with each other.

We will consider the operating system from two points of view: users and systems.

To a user, the operating system provides a more convenient interface. This allows the user to log in, manipulate files and run programs in a reasonably intuitive and convenient manner. Meanwhile, it provides protection of the user's data from unauthorized access, and ensures that the user is allocated a fair share of the computer's resources.

The user would like to do things like run programs and read and write files and communicate over the network without worrying about the details of what goes on at the lower levels. The overriding theme here (as in so much of computer science) is back to *abstraction!*

To a system, the operating system provides safe and efficient access to the actual hardware. The operating system tries to share resources when safe to do so and restrict access when necessary.

We can think of the operating system as a big resource manager.

Examples of Problems

Many important ideas in Computer Science arise in the study of Operating Systems:

- There are 3 users, each wishing to use the computer at the same time. Each has a program that needs to run for 5 minutes. Is it better for the system to run the first to completion, then the second to completion, then the third? Should it switch among them once a minute? Once a second? Once a millisecond? After every instruction? Which makes the most efficient use of the system's resources? Which makes the users happiest? Does the answer depend on what the program is doing for those 5 minutes?

- Suppose we have two programs, one that generates output values that are used as inputs to the other. We either do not want to or cannot have the second program wait for completion of the first before it starts to work: the programs must execute concurrently. How can we manage this situation if values may be generated by the first more quickly than they can be processed by the second? Or vice versa? Or if the situation changes over time?

- Suppose we have a one-lane bridge. How can you most efficiently manage traffic across the bridge? This sounds simple enough, but the best answer is not always clear. Some ideas include: just let people take turns, have a traffic light that alternates turns, have a pair of flaggers, give one direction precedence. But there are many potential problems: what if cars come in on both sides and meet in the middle? Someone's going to have to back up. The traffic light can be pretty annoying if you're stuck at the red and you wait and wait and don't see anyone come the other way. This is an unnecessary wait.

- Suppose we have a shared printer. If multiple people want to print at the same time something has to make sure the jobs don't get intermingled.

- The one-lane bridge example is a situation where a deadlock can arise. It can come up in more subtle ways. Think of this like gridlock. Everyone is waiting for someone else to do something before they can proceed. No one gets anywhere.

  In a computer system, this could be a situation where two users need exclusive access to two resources.

A simple example is two users who need to copy tapes (or in a more modern environment, substitute any removeable media that requires a hardware device). The system has two tape drives, and a tape drive is necessarily granted to one user at a time. User 1 requests a drive and gets it. User 2 requests a drive and gets it. User 1 requests a second drive, but must wait until User 2 finishes with the one he has. User 2 requests a second drive, but must wait until User 1 finished with the one he has. Uh oh. We can think of this as two antagonistic users, but even "friendly" users may not be aware that they are holding a resource that is preventing the other resource they need from ever becoming available.

- Suppose we have a collection of processes that are cooperating on a task. They need to coordinate. We'll look in some detail at process synchronization both from the point of view of algorithms that use it and what hardware and operating system support is needed to make these kinds of programs correct and efficient.
- If you have a disk attached to the computer, and several users of the computer, how do we organize data on the disk so it
  - is convenient to write and read,
  - is organized efficiently (quick to access, not a lot of wasted space), and
  - enforces appropriate protection on the files, to make sure users can't read or worse yet modify or delete the files that belong to some other user, but preserving the ability to share data effectively when appropriate?
- Suppose we have a network of computers – like the college's public labs.

  If we have a collection of computers shared among a collection of users, how can we devise a system where the resources are efficient and easy to use, yet secure?

  An approach that works well in our lab, where the systems typically have only one user at a time – the one who is sitting in front of a given computer, might not work well in a lab where the computers are used for long, CPU intensive jobs, such as graphics rendering or scientific computation.

Most of the problems that come up are not specific to a given operating system or type of computer. In fact, many of these same problems have been evident from the eariest historical systems right up to current systems. And they are as relevant now as they were when I took my first OS course in 1991 or so, and when I taught my first OS course in 1998.

An interesting thing about operating systems, and in fact much of computer science, is that an important "historical example" is often no more than a few decades old.

Some of the historical examples will likely be very familiar to you. What old systems were in your childhood? Some from mine include the Commodore 64/128.

And even when you might think some of the issues that were important on your parents' (or your grandparents'?) old Commodore 64 or Apple][keep coming back in your tablets and smartphones and watches or other smaller-scale special purpose computers.

The course is not about which is the best operating system (though we can make our opinions known from time to time). One thing we'll see as we go is that different systems have strengths and weaknesses that make them appropriate in different situations, but that there is also a whole lot they have in common.

When we compare approaches to a particular problem, trying to determine which approach is better, the answer will often be "it depends."

We use Unix-like operating systems as our model, but modern Windows systems have most of the same ideas underneath. Macs are really just Unix systems with a shiny user interface. Android devices run a variant of Unix. iOS devices run the XNU kernel from Darwin, which is a BSD-based Unix variant.