Syllabus: CIS 310 Operating Systems

Brian C. Ladd

Fall 2023

Catalog Description

> Principles of operating systems, concurrency, scheduling, virtual memory, device management, security and protection, deadlocks, introduction to networking. Prerequisite: CIS 205 and CIS 300. Fall

Overview

What is the **operating system** (OS) on a *digital*, *binary*. *general-purpose* computer? At a very (very) high level, it is a *resource manager*, controlling each running process's access to the CPU, the RAM, and the I/O (I love TLAs; [1] I blame the Army, even though CS has no shortage of them). Beyond that, the OS *shares* those resources so that the computer is not limited to only running a single process at a time.

The operating system starts and stops *processes*, **running** computer programs along with all the metadata needed to let them share the hardware with other processes. The program part of the process is the *machine code* that tell the CPU what to do. The **general-purpose** nature of the modern computer is laid bare because the OS treats the code as **data** that it manipulates while the CPU can interpret and run the **algorithm** encoded in the instructions. The same memory pattern has two *simultaneous* meanings as code and data.

Our text for the semester talks about "three easy pieces" of an operating system:

1. Virtualization — Using hardware safely requires careful programming [2]. Rather than trust each programmer to get it right, the OS transparently "pretends" to be hardware, *virtual hardware*. This interposition can enforce protection of the real hardware, provide a higher-level of abstraction to the programmer, and share the resources across multiple processes at the "same" time.
2. Concurrency — If multiple processes, or even *threads* within one process, are active *concurrently*, then access to hardware must be partitioned (in time, space, or both). The operating system (and the physical hardware) provide low-level primitives for implementing safe concurrency.
3. Persistence — Memory in a computer, RAM, is *volatile* (it can only maintain state while it has power); the OS implements file systems on *nonvolatile* devices (*e.g.* hard disk drives (HDD), solid-state drives (SSD), etc.) in order to keep/restore digital data across reboots.

Each of these pieces requires some level of hardware support and each provides a higher-level, abstract machine for the programmer. The higher abstraction smooths out many of the rough edges of programming the computer.

Calendar Considerations

- The last day to **add/drop**: 2023-09-01.
- The last day to withdraw or select S/U: 2023-11-03.
- **Fall Recess**: 2023-10-09 – 2023-10-10 (no classes on these dates).
- **Thanksgiving Recess**: 2023-11-22 – 2023-11-24 (no classes on these dates).
- **Last Day of Classes**: 2023-12-08.
- **Final Exam**: 2023-12-15 1015-1215.
- Additional Calendar Information: SUNY Potsdam Academic Calendar.

[1] *Three Letter Acronym*

[2] The Hercules Monochrome Graphics Adapter (c. 1980) could be set on fire from software. Fun times, fun times.

Student Learning Outcomes

Operating Systems provides a thin stalk reaching from just above the physical hardware of the machine to just below the run-time library of your favorite high-level programming language. Upon completing this course, students should be able to:

1. **OSTEP** *Explain*, individually and in combination, the "three easy pieces" of operating systems: *persistence*, *virtualization*, and *concurrency* along with how *abstraction* is an ever present **fourth** piece.
2. **Hardware** *Explain* multiple instances of *hardware assistance* required for various operating system mechanisms such as interrupts, context-switches, concurrency primitives, translation from a disk block number to a platter-track-sector coordinate, and limited-direct execution.
3. **System-level Programming** *Implement* system simulations and write tools that interact with the operating system in the `C++` language. This includes **locks**, **file systems**, **scheduling**, and **memory address translation**.
4. **Low-level Execution** *Trace* operating system actions from interrupt to completion, for example **context switching**, **page-based memory address translation**, and using **atomic** assembly instructions to implement safe concurrency.

Note: It would be a very good idea to review this section of the syllabus before exams (especially the final). You can also use it when filling out course evaluations to review what the class taught and what it was expected to teach.

Computer Science Program Student Learning Objectives

The Computer Science department has adopted five **program** learning objectives for students completing any undergraduate major in the department [See https://cs.potsdam.edu for the complete list.] `Foundations of Computer Science` addresses PSLO #1:
Students graduating from the Computer Science department at SUNY Potsdam are expected to be able to

> 2. *Solve problems through analysis and implementation of tested programs that use data structures and algorithms.*
>
> **System-level Programming** focuses on programming in `C++` to learn how operating systems work.
>
> 3. *Program the multiple layers (e.g., compiler, operating system, network, assembly language) between a high-level programming language and the underlying hardware.*
>
> **OSTEP** outlines the multiple layers in an operating system and **Hardware** explores the interface to the lowest layers. Both **Low-level Execution** and **System-level Programming** work with the *interfaces* for programming at lower levels of abstraction.

Feedback

It is important to me (Dr. Ladd) that you *learn* how operating systems work, how to interact with them programmatically, and how to communicate about related technical topics. This is a 300-level, core course in the Computer Science majors and one of the points where a non-Java programming language is introduced.

To the end of learning to **think** about operating systems and **communicate** that thinking, you will be doing a lot of programming, a lot of reading, and a lot of group work. The assignments are designed to reinforce each other: the readings explain topics explored in the programs which build on exercises you have done in groups.

There are also *reflection* exercises, small written responses where you will explain what (and sometimes how) you learned the material. Reflection is **very important**: cognitive scientists have shown that reflection greatly improves the fixation of new knowledge for the learner; your answers provide me with a moving snapshot of how you are learning.

You are also expected to write *readable* code. Comments, white space, appropriate variable names, appropriate scope of functions (single-concern), etc.

Grading

You will be assigned a grade at the end of the semester as that is required. I am much more interested in what you learn than in putting a number on it.

Reading	**No Grade**
Homework	10%
In-class Work	10%
Programs	40%
Exams	20%
Final Exam	20%

Reading Not optional but not directly graded, the readings of the text, assignments, and other provided information are provided to give you something to **think** about OS. Note that **Reflection** will often require you to refer to recent (and not-so-recent) readings to provide evidence for your thinking.
Homework Designed and assigned to help you grow your knowledge about *operating systems*, this is primarily **practice**. That does not mean *optional*: just as playing an instrument, speaking a foreign language, or participating in a sport requires practice, so, too, does learning something as involved as OS.
Cognitive science shows that thinking about thinking has an outsized impact on learning. Guided reflection, where you are asked to think about specific questions about the material and what you know about it is a way that I can teach you how to self-reflect for learning deeper in the future. This carries a grade; it will come from the seriousness with which you approach the process and the **evidence** you bring to any statements that you make.
In-class Work To include group work. This, too, is practice for you. The point of doing it in class? So that I can ask, "What makes you say that¿' as I eavesdrop or you present your answers to the class. I will be using a class roster to select call-on-order so that I do not default to the first (or last) raised hand.
Programs Programs for learning C++ and several multi-week projects.
It is imperative that you read assignments when they are assigned!
Some assignments will have *multiple phases*, perhaps a design document and then working code or a test plan. Always read the assignment as early as possible so that you can ask questions if you happen to not understand it and also so the your subconscious can be plugging away at it in the background.
All programs **must** compile so that I have something to assess. If you have been given tests, those are expected to pass, too. I count on *you* to start early enough that if you run into problems, you can **talk to me**.
Quizzes, Exams & Final Exams are my chance to assess what you know from reading, doing the homework, programming, and the work in class. They are done without access to ChatGPT or StackOverflow. Thinking about operating systems still requires facility with *details*. Expect a majority of the questions to be *novel*, compared to previous in-class and home work.

Textbook and Readings

Arpaci-Dusseau, R. and Arpaci-Dusseau, A., *Operating Systems: Three Easy Pieces 1.01E*.

Operating Systems: Three Easy Pieces (OSTEP) is available for free on-line: `https://pages.cs.wisc.edu/~remzi/OSTEP/`; that page also has references for electronic and print copies from various locations.

Readings are a primary way you will learn the material is this course. You should make sure you read assigned sections *before* class to activate your brain in the realm of operating systems. You will probably have to read any given chapter two or three times to integrate the information; this is normal when *learning* something for the longer term.

Schedule

Week	Unit Concept	Topics
1	*Getting Started*	**Programming in C++** Course Introduction C++ and `make` TDD in C++
2		**What is a *computer*?** Three Easy Pieces — The Hardware Hardware, Software The `List` ADT
3		**What is an *operating system*?** Resource Manager Interrupts Exam 1: High-level Operating Systems
4	*Persistence*	**I/O Devices** Device Interface Using What We Know File and *Directory* API
5		**File System Implementation** What is a *File*? Finding the `inode` What is slow?
6		**Faster, Bigger, Safer** What is *still* slow? What could go wrong? Exam 2: Persistence
7	*Virtual Processors*	**Programming w/ Processes** Fall Break Context, *k-stack*, Switch `fork`, `exec`
8		**Code as Data** Program ⇒ Process Who's Next: Scheduling Batch Scheduling
9		**Scheduling** MLFQ I MLFQ II Exam 3: Virtual Processors
10	*Virtual Memory*	**Address Spaces** Virtual ⟶ Physical Segments in Virtual Memory More and More Segments
11		**Paging** No *External* Fragmentation Page Table Translation Table Lookaside Buffer
12		**Paging and the Heap** Address Translation `malloc/free` Fragmentation Redux
13		**Thanksgiving** Exam 4: Virtual Memory Thanksgiving
14	*Concurrency*	**The Parallelism *Problem*** What could go wrong? Atomic Instructions Tracing Concurrency
15		**Building Locks** Atomic Assembler ⟶ Lock Locks, Queues, Process State **Deadlock**
Exam		

General Rules

Your choices make your fate. Information arms you to make your best decisions and enjoy the best fate. This section describes the class expectations so that you can meet them (or decide not to).

Instructor Expectations of Students

Communication

Read/respond to e-mail. Read/respond to the course Moodle site. These are the two primary means of communication in the class. You should make use of them.

Moodle is the communications hub of this class. It is where readings, writings, programs, presentations, and homework deadlines appear. If you find any discrepancies or have questions about due dates or the meaning of assignments **ask in person, in class, or in e-mail**. My office hours are listed in my Moodle profile.

Students are expected to have a copy of the textbooks (if any) and are expected to complete reading assignments *before* the beginning of class the day they are due. In-class participation requires you to have engaged the reading.

Students are expected to listen actively in class. They are also expected to take notes in class. These two activities (which are linked) correlate strongly with understanding the material presented. Do not copy slides or board notes directly: digest them and restate them in your own words. Mark where you expect test questions to lurk (sometimes the instructor will even help with this).

Advanced courses often use a version control system (such as `git`) to retrieve and submit source code and programming assignments. This requires you to understand how to use `git`, to understand the difference between the *class* repository (which belongs to the instructor) and *assignment* repositories (which belong to you).

Attendance

American undergraduates are old enough to join the army, vote, and even get married. You are each mature enough to make your own decisions about attending class. Be aware that you decisions have *consequences*.

Students are expected to attend every class. Students are responsible for all material covered in every class meeting. There is no taking of attendance in this class (see 'old enough' in the previous paragraph).

The study of computer science is cumulative; past experience shows a strong correlation between high absences and low grades.

The registrar schedules the final exams every semester; I am not able to move them and do not give early finals. Take-home final exams are due *before* the end of the scheduled exam period (it *will* be accepted early).

Do Your Own Work

Do your own work. That should go without saying but it appears that for some students it bears repeating. Doing your own work means **not** using online resources such as `chegg.com` to look up answers or asking *ChatGPT* (or the generative AI of the moment) to solve problems or design code. Those constitute **academic dishonesty.**

Code you plan to turn in for an individual grade should not be shared with any other student; you should never look at another student's project code, not even for 'guidance.'

I *want* you to discuss assignments and show off results of your playing with programming different things. Discussion at a high-level, even if it includes details on data structures is fine; debugging assistance is also acceptable though the depth of code analysis necessary for debugging skates on thinner ice; directing someone else on exactly what methods a given class should have or the exact format of a programmer-controlled file should take is beyond the pale.

This does *not* apply to group work: in a group project, all members of the team are expected to see and understand all parts of the project. Cross-disciplinary understanding is a highly valued skill in the real world and a major reason for group work in our curriculum.

Turning In Work

Start early, everything takes longer than you think.

As mentioned earlier, reading is to be done *before* class. Homework assignments are also due at the beginning of class so please complete them before class.

Assignments have a due time: a date and a time when they are due. If they are due electronically (they all are), make sure you submit them the expected way by the expected hour.

Make sure you check the spelling and arithmetic in your assignments. It does not help your grade (or my mood) to turn in sloppy work.

Attitude

Learning is not always easy. Learning is not always comfortable. Students must actively engage the material and be able to ask for help. This does not mean look at the problem for fifteen seconds and throw up your hands because it is 'hard.' It does mean beating your head against it for a bit and then asking me for guidance.

Be respectful. Trust me that I actually have a plan and I know where we are going (okay, I don't *always* know exactly where we are going at every moment but I do have a plan). The readings, the assignments, the lectures, they all go together.

Start early! Everything takes longer than you think.

Student Expectations of the Instructor

Communication

You may expect me to state and keep office hours (you know where my office is, right? 301 Dunn Hall) They are part of my Moodle profile.

You may expect fairly prompt answers to e-mail (next morning is typical).

You may expect confidentiality: I will not discuss your grades with anyone except my departmental colleagues (and then only to make sure that I am doing my job correctly). I will not share work you have done without your permission.

Attendance

You may expect me to be prepared. This means for class every day and it means with assignments, too. You should expect adequate time to complete an assignment after it is published. You may expect some flexibility (with in the limitations of getting through the course materials) in due dates.

Assignments

Expect assignments to be clear (correct spelling, correct grammar). Expect most sample code to compile and run. Expect assignments to be accompanied by a clear system by which they will be evaluated. Expect me to follow that system.

Students should expect that their work will be evaluated on its own contained merits and not based on how closely it follows the instructor's personal/political views (unless I put parroting my political views into the rubric).

Student Expectations of One Another

Wait, students have expectations of one another? Yes, they should. The most important expectation we should all have of each other is to create a safe learning environment. By safe I mean we should all feel safe to ask questions, to admit mistakes, to try and to be wrong.

Students should be willing to help one another and should all be aware of the limits.

In group work, all students in the group should expect equitable division of labor and should expect to evaluate every member of the group.

Students should expect to evaluate other students" work (and to be evaluated by other students). Both parts of this process have valuable learning outcomes.

Students should expect to feel safe being different. What does this mean in computer science? I actually am not sure. It means respecting each other, evaluating each other in openly communicated terms related to computer science (since that is the topic of the course), and being flexible, accommodating each other whenever possible.

Grading Grievances

When graded work is returned you may have questions about how it was graded. If the question pertains to process (*e.g.,* addition, number of points per subsection), by all means ask me as soon as possible. Typically, if the grade is changed, you will be asked to e-mail me the grade update so that the Moodle grade book is current.

If, however, you feel that your grade fails to reflect the quality of your work (you want to argue that your answer is, in fact, correct, rather than incorrect), take a deep breath and wait. You must wait no less than one and no more than seven days after the work is returned to you to raise the issue with me.

After the cooling-off period, resubmit the work along with a *written* explanation of your concern. This is to assure that your concern gets a cool, thoughtful consideration. If I agree to regrade the work, I will regrade the entire project, replacing the old grade with the new one.