Operating Systems

Syllabus

Class Resources

Course Website http://pdinda.org/os: All course details and schedule.
Canvas: Grade reports, class recordings, and zoom links. Already enrolled.
Piazza: Class discussions, questions, and messages. We will enroll you.
Github Classroom: Lab access and submission. We will enroll you.

Instructor

Peter Dinda
pdinda@northwestern.edu

Teaching Assistance

Nick Wanninger (TA)
ncw@u.northwestern.edu

Griffin Dube (TA)
gadube@u.northwestern.edu

Thomas Filipiuk (PM)
thomasfilipiuk2024@u.northwestern.edu

Garrett Weil (PM)
gsw@u.northwestern.edu

Location and Time

Lectures:	M/W 9:30-10:50, Tech M345 (**in person**)
Discussion:	W 4pm, Location TBD + Zoom (recorded)
Office Hours:	See Canvas or Piazza
Midterm Exam:	TBD, mid-quarter, out of class Details to come.
Final Exam:	Monday, March 13, 12-2, Details to come, **may move**

Prerequisites

Required	CS 213 or CE 205 or equivalent
Required	CS 214 or equivalent
Required	Experience with C or C++
Required	Some experience with programming in a Unix environment (e.g., as in CS 211 and CS 213)

Any version of CS 213 or CE 205 is acceptable, but we will expect that you have seen basic concepts such as the existence of exceptional control flow and virtual memory, and the typical Unix system calls for processes, threads, and files+I/O. The syllabus shown in pdinda.org/ics is a good starting point.

Any version of CS 214 is acceptable, but we will expect that you have seen basic data structures, algorithms, and their implementation. These include linked lists, balanced search trees, hashing and hash tables, heaps, graphs, sorting, etc.

Experience with C or C++ in part means familiarity with arrays, structs, unions, and, most importantly, pointers and pointer-based data structures. Low-level pointer-based mechanisms are used throughout an OS, and by the underlying hardware.

Experience with programming on Unix means being able to navigate the Unix command line, remote access, use/extend Makefiles, etc.

CS 343 satisfies one of the **Systems Breadth**, Tech Elective, and Project requirements in in the Computer Science curriculum in both McCormick and Weinberg. CS 343 can also be taken for credit within the Computer Engineering curriculum.

Textbook

Andrew S. Tanenbaum and Herbert Bos*, Modern Operating Systems, **4th Edition**,* Pearson, 2014, (ISBN-13: 978-0133591620, ISBN-10**:** 013359162X)

We also considered several other books for this course, which may be useful as further references:

- Abraham Silberschatz, Peter Galvin, and Greg Gagne, *Operating Systems Concepts,* 10th edition, Wiley, 2018.
- Remzi Arpaci-Dusseau, and Andrea Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 2018. **[freely available]**
- Thomas Anderson, and Michael Dahlin, *Operating Systems: Principles and Practice*, Recursive, 2014.
- William Stallings, *Operating Systems: Internals and Design Principles*, 9th edition, Pearson, 2017.

The choice of Tanenbaum as the textbook for this course is a compromise. All of these books have strengths and weaknesses. Tanenbaum allows us to pair high-level presentation of OS concepts with their implementation within particular OSes. In your assigned readings, you will often see this pairing, combining early chapters (concepts) with much later chapters (implementations). There are also several handouts created specifically for this course on topics that we think need a treatment beyond what Tanenbaum provides.

It is important to note that your CS 213 textbook (Randal Bryant, and David O'Hallaron, *Computer Systems: A Programmer's Perspective*) has an excellent "what every programmer should know" treatment of some of the topics we will cover, including threads, processes, virtual memory, and the various Linux/Unix system call interfaces. If you've taken any version of CS 213 in Fall 2021 or later, you will have seen this treatment.

There are also books on specific operating systems that advanced students might be interested in, particularly on FreeBSD and Linux. Ask if you're curious.

Objectives, framework, philosophy, and caveats

This course introduces you to the basic, foundational concepts and principles of operating systems, many of which generalize to other areas of computer science and engineering. You will learn many of these concepts and principles by applying them in practice on a modern machine though labs that are designed to put you in the shoes of a systems-level developer. OS (and systems more broadly) is very much a learn-by-doing kind of area.

The following concepts and principles are included:

- **OS Structure:** kernel, device drivers, file systems, network stacks, schedulers, system calls, libraries, toolchains, language virtual machines, user interface/shell, applications, etc.
- **OS Models:** monolithic kernel, microkernel, virtual machine monitor/hypervisor, jail/zone/container, exokernel, unikernel, ...
- **OS Abstractions:** thread, name space, address space, process, IPC, virtual machine, container, file, directory stream, plus abstraction design within the kernel (devices, file systems, ...)
- **Concurrency Sources:** multiprocessors, devices, interrupts, threads, processes, horror stories, ...
- **Concurrency Challenges:** memory system coherence/consistency, race conditions, deadlock, livelock, horror stories, ...
- **Concurrency Control:** interrupt control, atomics, spinlocks, critical sections, blocking vs waiting, mutexes, semaphores, condvars, monitors, barriers, lockfree/waitfree models, plus typical synchronization problems such as producer-consumer, reader-writer, and dining philosophers.

- **Scheduling and Resource Management:** theory, FCFS, GPS, SRPT, dynamic priority (e.g. Unix), lottery, fixed priority, stride, preemptive vs non-preemptive, real-time vs non-real-time, horror stories, ...
- **Virtual Memory:** hardware-software co-design, paging, swapping, segmentation and (possibly) current alternatives.
- **Device Drivers:** interrupts, DMA vs PIO, MMIO vs PMIO, atomics, hardware memory barriers, software memory barriers.
- **Protection and Security:** kernel/user mode, mode/ring transitions, role of encryption, interaction with virtual memory, horror stories.
- **Memory management:** page allocation versus heap allocation, garbage collection, allocation in special contexts (e.g. interrupt context), page replacement, working set.
- **File systems:** issues/interfaces, data structures on block devices, examples (V7, FAT+, ext2+)
- **Principles:** policy versus mechanism, orthogonality, worse-is-better, lazy evaluation, caching, end-to-end argument, mythical man-month, no silver bullet, hardware/software co-design

The hardware environment that we will focus on is Intel/AMD machines running in 64 bit mode ("x64"), which is the commonplace platform for systems ranging from laptops to supercomputers today.[1] Your lab work will be done on Linux in the C programming language.[2] Two of your labs (on concurrency and scheduling) will be done in user-level Linux. The remaining labs will be in the context of the Nautilus kernel framework ("NK"), a research kernel develop at Northwestern and other institutions. The experience you gain in NK will generalize to the Linux kernel, for the most part.[3]

Note that we will prioritize among these points so that if more time is needed to cover a high priority topic, there may be less or no coverage of a lower priority topic. There are also only 19 lectures in this instance of the course.

---

[1] Most of what you learn about x64 vis a vis OS will generalize to the other main platform, ARM, which is the basis for phones and tablets, as well as the next generation of Macs.

[2] Linux is the common OS on everything except laptops and desktops. It is also the OS underlying Android. C is the lingua franca of low-level software development.

[3] In the design of this course, we considered several other options. The most desirable would have been to have you work within the Linux kernel itself. This proved to be intractable from a pedagogical point of view. The complexity we would have to shield you from, particularly in a lab based on paging, would have been overwhelming to manage. We also considered the teaching OS xv6 for IA32 and for RISC-V. IA32 and RISC-V both would require revisiting material students have already learned, for x64, in CS 213, plus xv6 for IA32 would have made a device driver lab particularly challenging to pull off. Another consideration was to use CMU's Pebbles OS specification and have students build Pebbles from scratch as in CMU's course. This was also limited to IA32, and seemed intractable to execute in a single quarter. The intent behind using NK is to give a view inside a modern, x64 codebase with clear internal interfaces that has a development model (e.g., Kbuild, C, etc) that is similar to Linux.

Lectures / Attendance Requirement

Lectures will be held in person in the scheduled room at the scheduled time. It is important that you complete the reading assigned for each officially scheduled class session before that session (the reading for the first session is an exception). **Based on your reading, you should prepare at least one question for each class session.**

There is plenty of content that is separate from the textbook, handouts, and codebases, and I do not use slides in lecture.

You are encouraged to ask questions or provide comments, either verbally, in chat, or on Piazza. You can also help answer other student's questions and comments. What I'm asking of you is: Read. Attend. Ask. Answer. There is no such thing as a dumb question (or too esoteric of a question) - we will try our best to answer or comment on all questions.

Optional Discussion Session and Other Ways of Getting Help

Your TAs and peer mentors will run an optional weekly discussion. The goal of the optional weekly discussion is to provide a place to learn more and to get help in a more structured way than office hours. The discussion session will also be available on zoom and will be recorded.

Your instructor, TA, and peer mentors will also have regularly scheduled office hours and be available by appointment if these do not work. We have scheduled our office hours to try to spread them across the week.

We will use an online discussion group on Piazza as well. We will enroll you. The link is on the course web page. The intent is to have multiple venues for discussion with different styles so that all students feel comfortable participating. If you have a question, answer, or comment, please put it forward. We will try our best to answer.

Labs will be done using GitHub Classroom. One goal here is to make it straightforward for us to see the current state of your lab work, so that we do not have to spend a lot of time reconstructing setups during office hours, etc. Push early and often!

Computing Resources

You will have Linux accounts on the Wilkinson machines, and it should be possible to do some of your work on them, or other 64-bit Linux machines. You will also have access to a high-end server which has a range of software set up for use by this course. This is the easiest option and is also where we will grade labs. The very first lab is intended to get you familiar with this environment by having you build and run a kernel on it.

It is also possible to work on your own machine. Generally speaking, using Linux will be easiest. I often do development with Ubuntu installed in a VMWare VM on my Mac or PC. We will provide instructions in Piazza for those who would like to set up their own environment.

Labs

We will have five programming labs. Except for the first lab, labs should be done in groups of up to three. **Start looking for a partner on day one.**

There are five labs. 60% of the grade in the class will be based on lab work, with a breakdown as follows:

5% Getting Started Lab (done individually)
10% Producer-Consumer Lab
15% Queueing/Scheduling Lab
15% Device Driver Lab
15% Paging Lab

We will use GitHub Classroom for disseminating and handing in labs. It is important that you and your partners make sure that your repositories are private. Only your group and the course staff should be able to see your repos.

The Producer-Consumer Lab and Queueing/Scheduling Lab are user-level Linux labs. The others are all done within a research kernel developed at Northwestern. All hardware is x64. All code is in C.

Exams

There will be a midterm exam and a final exam. The final exam will not be cumulative. I do not provide practice exams. Instead, we will schedule midterm and final exam review sessions. Details on how exams will be handled remotely will come soon.

Grading

60% Labs (breakdown above)
20 % Midterm (covers first half of the course)
20 % Final (covers second half of the course)

There is extra credit in many of the programming labs.

Your score in the course is the weighted average of your scores on each of the components. You can view all currently graded material, and your score, at any time on Canvas. Final grades are based on the course score (the weighted average), with the basic model being that the 90s are A territory, 80s are B territory, and so on. This model will be adapted toward lower thresholds if necessary based on overall class performance. That is, this is NOT a curved class.

The instructor ultimately assigns scores and grades in consultation with the TA and peer mentors. If you have a problem with a score on an assignment/exam or your grade, you are welcome to bring it up with him or the TA, but only the instructor is empowered to change grades.

Lab Late Policy

For each calendar day, or portion thereof, after the due date for a lab, 10% is lost. After 1 day, the maximum score is 90%, after 2 days, 80%, etc, for a maximum of 10 days.

Cheating and Inadvertent Disclosures

Since cheaters are mostly hurting themselves, we do not have the time or energy to hunt them down. We much prefer that you act collegially and help each other to learn the material and to solve problems than to have you live in fear of our wrath and not talk to each other. Nonetheless, if we detect blatant cheating, we will deal with the cheaters as per Northwestern guidelines.

As we note above, it is important that you control access to your GitHub repos.

Please do not place class materials from on any public site. If it's on the course web site, it's already public and will remain public. If it's from the discussion group or from the handout directory on the course servers, it should not be shared publicly.

Particularly sensitive materials, such as exams, may have the student owner's name steganographically and cryptographically embedded in them in several ways. If such materials become visible on some cheating support web site, they can easily be tracked back to the student. Please properly discard such materials after you are done with them.

Accessibility

Any student requesting accommodations related to a disability or other condition is required to register with ANU (accessiblenu@northwestern.edu; 847-467-5530) and provide professors with an accommodation notification from ANU, preferably within the first two weeks of class. All information will remain confidential.

Should you need them, additional campus resources are available, including, but not limited to:

- Accessible NU www.northwestern.edu/accessiblenu/
- CAPS www.northwestern.edu/counseling/index.html
- Student Enrichment Services www.northwestern.edu/enrichment/

If you have special concerns, please reach out.

Common sense

Be excellent to each other. The goal here is to learn something about operating systems. This is not a competition (recall there is no curve). If you have some background coming into the class, or this comes easy to you, help others. Also, while all of the course staff are quite happy to go off into the deep end, we also welcome a nudge to come back. This is an introductory course, after all.

Schedule

Lecture	Date	Topics	Readings	Labs
1	1/3 T ("North western Monday ")	Introduction, OS Structure, OS Models, HW/SW interface, History	Chapter 1, 8.1.2, 10.1, 10.2	Start lab out
2	1/4 W	Concurrency Sources: hw, interrupts, threads, processes, ...	2.1, 2.2, 5.1.5, 8.1.1	
1/9 is the last day for adding courses or changing sections.				
3	1/9 M	Concurrency Sources: continued	2.1, 2.2, 5.1.5, 8.1.1	Start lab in, PC lab out
4	1/11 W	Concurrency Challenges and Control: races, mutual exclusion, critical sections	2.3, Concurrency, Unix	
1/16 M is MLK Day – No Class				
5	1/18 W	Concurrency Challenges and Control: blocking, mutexes, spinlocks, semaphores, condvars, barriers, monitors, etc.	2.3 (cont.), 8.1.3, Therac	
6	1/23 M	Concurrency Challenges and Control: deadlocks, detection, avoidance, prevention, starvation, lockfree/waitfree data structures	6, 2.5	
7	1/25 W	Scheduling: classic treatment	2.4, 10.3, 8.1.4	PC lab in, Queue lab out

8	1/30 M	Scheduling: workload, queueing, and real-time perspectives	Workload, Queueing, Mars	
9	2/1 W	*Virtualization/Containers/Cloud, Special topic or Slack*	7 (if first option)	
Midterm Exam Review: TBD, probably in discussion section				
Midterm Exam: Around here, time+location TBD				
10	2/6 M	Devices and drivers: principles	5.1-5.3, 10.5	
11	2/8 W	Devices and drivers: examples	5.4-5.8	Queue Lab in, Driver lab out
2/10 is the last day to drop a class.				
12	2/13 M	OS design principles	Chapter 12	
13	2/15 W	Virtual memory with paging and segmentation	3.1, 3.2, 3.3, 3.7	
14	2/20 M	Paging and swapping policies and their effects, working set, allocation	3.4-3.6	
15	2/22 W	Paging on x64 and Linux	10.4	Driver lab in, Paging lab out
16	2/27 M	Security and Protection	9.1-9.6, Spectre	
17	3/1 W	File systems: principles and issues	4.1-4.4	
18	3/6 M	File systems: examples	4.5, 10.6	
19	3/8 W	*Research Topic, Special topic or Slack*		Paging lab in
Finals week – Exam is Monday, 3/13, 12-2 *Grades due on Monday, 3/20*				

Readings are from the textbook, with these exceptions:

Therac	THERAC-25 article
Mars	Mars Pathfinder article
Spectre	Meltdown/Spectre article
Unix	Unix Systems Programming in a Nutshell Handout
Workload	Workload Characterization Handout
Queueing	Queueing Theory Handout
Concurrency	Concurrency Handout