Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Back to School
Basics

• Instructor: Yipeng (Roger) Song
  • I go by Roger 😊

• Email
  • Instructor: songyip@oregonstate.edu

• Office Hours: TBD @ TBD

• Requirements: Laptop/PC

• Programming Language: C, Assembly
More Basics...

• Be respectful (Establishing a Positive Community)
• Have a growth mindset
  • Most abilities could be developed through dedication and hard work
• Don’t cheat (0 tolerance!!)
  • https://studentlife.oregonstate.edu/studentconduct/student-info
• Be Proactive
  • Take control and cause something to happen, rather than just adapt to a situation or wait for something to happen
Course Description

• Goal: Learn how modern operating systems work

• Lecture & Lab
  • Learn *high-level fundamental concepts* of OS in the lecture
  • Practice engineering details with Labs
  • You will **build your operating system** (JOS)
  • Lab sessions: TAs will help you

• Topics
  • Virtual memory, Segmentation, Paging
  • Process, Isolation, Kernel, User
  • Interrupt, Exceptions, Synchronization, Concurrency
  • Filesystem
  • etc.

Image from: https://www3.nd.edu/~pbui/teaching/cse.30341.fa17/
Course Description

Goal: Learn how modern operating systems work

• Lecture & Lab

• Learn high-level fundamental concepts of OS in the lecture

• Practice engineering details with Labs

• You will build your operating system (JOS)

• Lab sessions: TAs will help you

Topics

• Virtual memory, Segmentation, Paging

• Process, Isolation, Kernel, User

• Interrupt, Exceptions, Synchronization, Concurrency

• Filesystem

• etc.

Image from: https://www3.nd.edu/~pbui/teaching/cse.30341.fa17/OS Kernel (Ring 0)
Course Description

• Goal: Learn how modern operating systems work
• Lecture & Lab
• Learn high-level fundamental concepts of OS in the lecture
• Practice engineering details with Labs
• You will build your operating system (JOS)
• Lab sessions: TAs will help you

Topics
• Virtual memory, Segmentation, Paging
• Process, Isolation, Kernel, User
• Interrupt, Exceptions, Synchronization, Concurrency
• Filesystem
• etc.

Image from: https://www3.nd.edu/~pbui/teaching/cse.30341.fa17/
Course Objective

• Understand how modern computer systems work (in detail)

• Be able to answer the following questions:
  • What happens when we turn on the computer? How does it boot?
  • How an OS runs an application?
  • How an OS runs application that requires memory more than its physical memory?
  • How multiple applications can run on the system?
  • How an OS enforces privilege separation?
  • How an OS protects itself from malicious software?
  • How multiple programs synchronize each other? How can we implement a lock?
Important Links

• Course Material: Canvas site
• Instructor: Yipeng Song(songyip@oregonstate.edu)
  • I go by Roger 😊
• TAs:
  • Jonathan Keller, Zexian Li, Soon Song Cheok (graduate TAs)
  • Alvin Johns II, Adrian Baker, Fatima Moussaoui, Kimberly Truong, Walt Bringenberg (undergraduate TAs)
• GitHub classroom
• Discord: https://discord.gg/jaPETYZGnP
• Assignment server: os2.engr.oregonstate.edu
Course Structure

• 10 weeks schedule
  • Virtualization (Week 1-5)
  • Concurrency (Week 6-9)
  • Persistency and others (Week 10)

• Textbook
  • [http://pages.cs.wisc.edu/~remzi/OSTEP](http://pages.cs.wisc.edu/~remzi/OSTEP)

• **Read**: prep materials posted on “Reading Materials”
• **Watch**: Lab tutorials
• **Study**: study JOS labs (tutorial videos / lab instructions)
• **Engage**: Lectures in person and office hours on Discord, discuss with peers!
Meeting Time

• Lectures (in person)
  • Attendance: Optional, but strongly recommended
  • MW 4:00 – 5:20 pm at WNGR 151
    Recordings from Spring 2021 are posted

• My office hour
  • TBD

• Recitations: served as additional office hours 😊 (No recitations in Week 1!)
TA Office Hours

• Office hours starts from Week 2
• Available via Discord (and/or in person?)

• When? See Canvas → Office Hours page
Grading

• 70% JOS lab assignment
  • Lab 1 (10%), Lab 2 (15%), Lab 3 (20%), Lab 4 (25%)

• 30% Quizzes (mini-exam) (10% each)
  • Quiz 1 (4/22) : Virtual Memory
  • Quiz 2 (5/13): System calls, faults, and exceptions
  • Quiz 3 (6/3): Concurrency

Grading Scheme (tentative):

100 >= A >= 93 (96 for graduate students)
93 > A- >= 90 (93)
90 > B+ >= 86 (89)
86 > B >= 83 (86)
83 > B- >= 80 (83)
80 > C+ >= 76 (79)
76 > C >= 73 (76)
73 > C- >= 70 (73)
70 > D+ >= 66 (69)
66 > D >= 63 (66)
63 > D- >= 60 (63)
F < 60 (63)
The Lab (70%)

• Four labs
  • JOS Lab 1 (10%): Booting a PC (2 weeks, due Monday, 10/9)
    • Bootloader, protected mode, etc.
  
  • JOS Lab 2 (15%): Memory Management (2 weeks, due Monday, 10/23)
    • Virtual memory, paging, etc.
  
  • JOS Lab 3 (20%): User Environment (3 weeks, due Monday, 11/20)
    • Process, user, kernel, system call, etc.
  
  • JOS Lab 4 (25%): Preemptive Multitasking (3 weeks, due Monday, 12/11)
    • Implementing context switching, multi-core support, inter-process communication, etc.
How to Conduct Lab Assignments?

• Visit Lab Tutorial Webpage
  • Canvas → Labs

• Watch Lab Tutorial Video
  • I will explain necessary stuff for the lab assignments in the video (code/examples, etc.) and also give some tips...
An Exercise Example in Lab 1

Exercise 3. Take a look at the lab tools guide, especially the section on GDB commands. Even if you're familiar with GDB, this includes some esoteric GDB commands that are useful for OS work.

Set a breakpoint at address 0x7c00, which is where the boot sector will be loaded. Continue execution until that breakpoint. Trace through the code in `boot/boot.s`, using the source code and the disassembly file `obj/boot/boot.asm` to keep track of where you are. Also use the x/i command in GDB to disassemble sequences of instructions in the boot loader, and compare the original boot loader source code with both the disassembly in `obj/boot/boot.asm` and GDB.

Trace into `bootmain()` in `boot/main.c`, and then into `readsect()`. Identify the exact assembly instructions that correspond to each of the statements in `readsect()`. Trace through the rest of `readsect()` and back out into `bootmain()`, and identify the begin and end of the `for` loop that reads the remaining sectors of the kernel from the disk. Find out what code will run when the loop is finished, set a breakpoint there, and continue to that breakpoint. Then step through the remainder of the boot loader.
The Lab Could be Difficult

- Coding KERNEL code in C
  - Any memory error -> Triple fault...

- Use GDB for debugging OS Kernel
  - Get familiar to tools ASAP.

- Assembly Languages
  - Intel x86

- Control hardware specific data
  - Page table
  - Global descriptor table (GDT)
  - Interrupt descriptor table (IDT)
The Lab Could be Difficult

- Coding KERNEL code in C
  - Any memory error -> Triple fault...

- Use GDB for debugging OS Kernel
  - Get familiar to tools ASAP.

- Assembly Languages
  - Intel x86

- Control hardware specific data
  - Page table
  - Global descriptor table (GDT)
  - Interrupt descriptor table (IDT)
The Lab Could be Difficult

• Coding KERNEL code in C
  • Any memory error -> Triple fault...

• Use GDB for debugging OS Kernel
  • Get familiar to tools ASAP..

• Assembly Languages
  • Intel x86

• Control hardware specific data
  • Page table
  • Global descriptor table (GDT)
  • Interrupt descriptor table (IDT)
The Lab Could be Difficult

• Coding KERNEL code in C
  • Any memory error -> Triple fault...

• Use GDB for debugging OS Kernel
  • Get familiar to tools ASAP..

• Assembly Languages
  • Intel x86

• Control hardware specific data
  • Page table
  • Global descriptor table (GDT)
  • Interrupt descriptor table (IDT)
The Lab Could be Difficult

- Coding KERNEL code in C
  - Any memory error -> Triple fault...

- Use GDB for debugging OS Kernel
  - Get familiar to tools ASAP..

- Assembly Languages
  - Intel x86

- Control hardware specific data
  - Page table
  - Global descriptor table (GDT)
  - Interrupt descriptor table (IDT)

### Table 4-2. Paging Structures in the Different Paging Modes

<table>
<thead>
<tr>
<th>Paging Structure</th>
<th>Entry Name</th>
<th>Paging Mode</th>
<th>Physical Address of Structure</th>
<th>Bits Selecting Entry</th>
<th>Page Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>PML4 table</td>
<td>PML4E</td>
<td>32-bit PAE</td>
<td>N/A</td>
<td>N/A</td>
<td>1-Byte page if PS=01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IA-32e</td>
<td>CR3</td>
<td>47:39</td>
<td>N/A (PS must be 0)</td>
</tr>
<tr>
<td>Page-directory-pointer table</td>
<td>POPTPE</td>
<td>32-bit</td>
<td>N/A</td>
<td>N/A</td>
<td>1-Byte page if PS=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAE</td>
<td>CR3</td>
<td>31:30</td>
<td>N/A (PS must be 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IA-32e</td>
<td>PML4E</td>
<td>38:30</td>
<td>1-Byte page if PS=1</td>
</tr>
<tr>
<td>Page directory</td>
<td>PDE</td>
<td>32-bit</td>
<td>CR3</td>
<td>31:22</td>
<td>4-MByte page if PS=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAE, IA-32e</td>
<td>PDE</td>
<td>29:12</td>
<td>2-MByte page if PS=1</td>
</tr>
<tr>
<td>Page table</td>
<td>PTE</td>
<td>32-bit</td>
<td>PAE</td>
<td>21:12</td>
<td>4-KByte page</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAE, IA-32e</td>
<td>PDE</td>
<td>20:12</td>
<td>4-KByte page</td>
</tr>
</tbody>
</table>
The Lab Could be Difficult

- Coding KERNEL code in C
  - Any memory error -> Triple fault...
- Use GDB for debugging OS Kernel
  - Get familiar to tools ASAP
- Assembly Languages
  - Intel x86
  - Control hardware specific data
    - Page table
    - Global descriptor table (GDT)
    - Interrupt descriptor table (IDT)

Attend lectures and watch lab tutorial videos on time and ask TAs for help!
Lab Rules

• DO NOT SHARE YOUR CODE WITH OTHER STUDENTS
  • You are encouraged to discuss with others about the assignments but do not ask/give the code to the others
  • Do not copy other students’ code or code available online
  • Do not publish your code online

• You will be asked to submit a simple write-up for the assignment
  • Describe how you solve each exercise/questions
  • Mention your collaborators in the write-up
  • Do not copy other students’ write-up
  • Do not publish your write-up online
Lab Rules

• Plagiarism will be punished via the Office of Student Life..
  • E.g., getting F or zero point for the lab assignment that matters with plagiarism...

• Please refer the Code of Student Conduct
AI Tool Usage in this class

• You must be the author of all work
• You may use AI to:
  • generate abstract ideas
  • polish or edit text you have drafted
  • quiz yourself
  • explain new or confusing concepts
  • generate code snippets to solve unassigned example tasks
• You may NOT use AI to
  • generate code snippets to solve a problem presented in a quiz, or lab assignment
  • draft the code implementation for a lab assignment
• If used, add a citation just like you would when you copy language or code from human authors.
Due Dates on the Calendar

Lecture/Lab slides will be posted here as the term progresses. This schedule is subject to change and should be viewed as a proposed timeline.

<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture Topics(s)</th>
<th>Slides (Videos are from Spring 2021)</th>
<th>Lab Tutorial</th>
<th>Reading Materials</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Week #0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9/28 Thu</td>
<td>Course Intro</td>
<td>• Lecture 1</td>
<td>• Lab Setup Guide</td>
<td>• Intro to C5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recorded.lec</td>
<td>• Slides</td>
<td>• x86 Assembly</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Video</td>
<td>• GDB Tutorial 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• GDB Tutorial 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• GDB cheat sheet</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• bash cheat sheet</td>
<td></td>
</tr>
<tr>
<td><strong>Week #1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/3 Tue</td>
<td>BIOS, Booting, and CPU</td>
<td>• Lecture 2</td>
<td>• Lab1</td>
<td>• x86 Address Translation</td>
<td>Lab 1 Due Monday 10/9 11:59 pm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recorded.lec</td>
<td>• slides1, slides2</td>
<td>• Address Spaces</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• video1, video2</td>
<td>• Address Translation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Segmentation</td>
<td></td>
</tr>
<tr>
<td><strong>Week #2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/10 Tue</td>
<td>Paging and Virtual Memory Translation</td>
<td>• Lecture 4</td>
<td>• Page Table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recorded.lec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/12 Thu</td>
<td>Virtual Memory Layout</td>
<td>• Lecture 5</td>
<td>• Lab2</td>
<td>• Paging: Intro</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recorded.lec</td>
<td>• slides1, slides2</td>
<td>• Paging: TLBs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• video1, video2</td>
<td>• Paging: Smaller Tables</td>
<td></td>
</tr>
</tbody>
</table>
Lab Rules – Late Submissions

• If you submit your assignment before the due date, then
  • You will get 100% based on the grading result

• If you submit your assignment within one week after the due date, then
  • You will get 75% based on the grading result

• If you submit your assignment more than one week after due, but before 6/12 11:59 pm, then
  • You will get 50% based on the grading result (75% for lab 4)
CS 544 Students

• Will have higher grade bar than CS444 (+3 pts)
  • E.g., A – 93 and over for CS 444, and 96 and over for CS 544

• *Note: I do round 😊
  • i.e. 89.45 → 89.5 → 90

Grading Scheme (tentative):
100 >= A >= 93 (96 for graduate students)
93 > A- >= 90 (93)
90 > B+ >= 86 (89)
86 > B >= 83 (86)
83 > B- >= 80 (83)
80 > C+ >= 76 (79)
76 > C >= 73 (76)
73 > C- >= 70 (73)
70 > D+ >= 66 (69)
66 > D >= 63 (66)
63 > D >= 60 (63)
F < 60 (63)
Tips to the Lab

• Study in a group (discussions are highly encouraged!)
  • But please write the code individually!
• Follow tutorial video
• Ask questions (Discord)
• Understand your time budget (debugging will take lots of your time!)
  • Plan ahead to finish the labs on time
• Learn basic tools (e.g., C, gdb, assembly, editors, tmux, etc.) ASAP
  • This will help you earn more time on doing labs...
  • https://missing.csail.mit.edu/
  • Up to Debugging and Profiling would be helpful...
Help Hierarchy

• Reread assignment, lecture slides, labs, syllabus
• Google/Bing/Open a textbook
• Ask a friend
• Check Discord for relevant posts or create a new question
• Ask a TA
  • You can attend office hours
  • TAs will also be monitoring Discord
• Ask Roger
Others

• Be active on Discord
• Read pinned messages in each lab channel
Assignment

• Lab Setup, and Lab 1 will be posted by tomorrow
  • As well as the tutorial videos