CS 444/544
Operating Systems II

Lecture 1

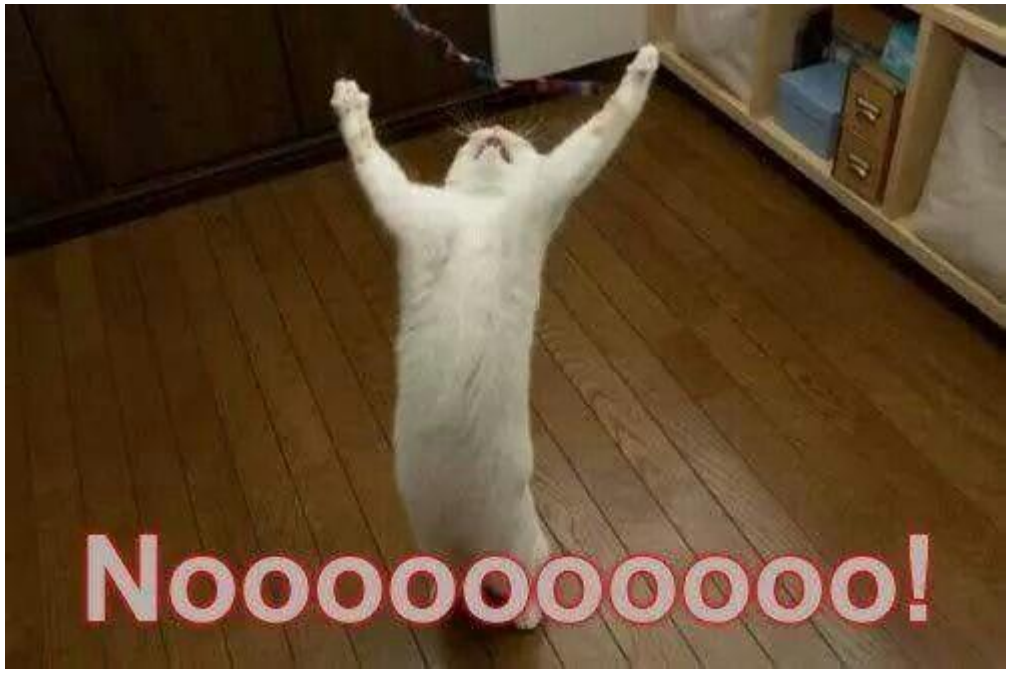
Course Intro

4/1/24

Acknowledgement: Slides drawn heavily from Yeongjin Jiang

Oregon State University

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.

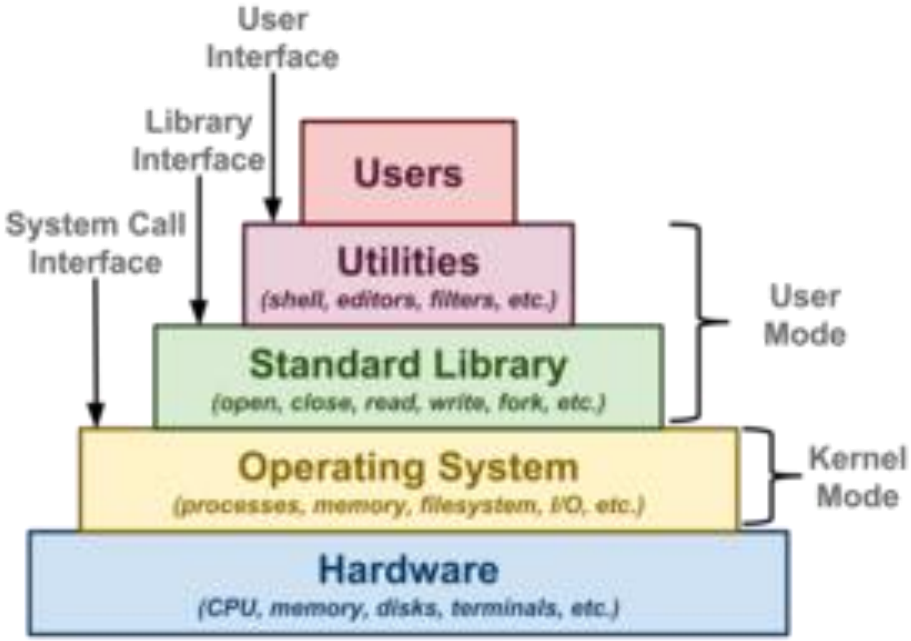


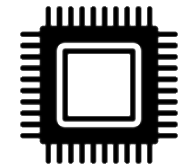
Image from: https://www3.nd.edu/~pbui/teaching/cse.30341.fa17/

Ring 3

ptio

n opera

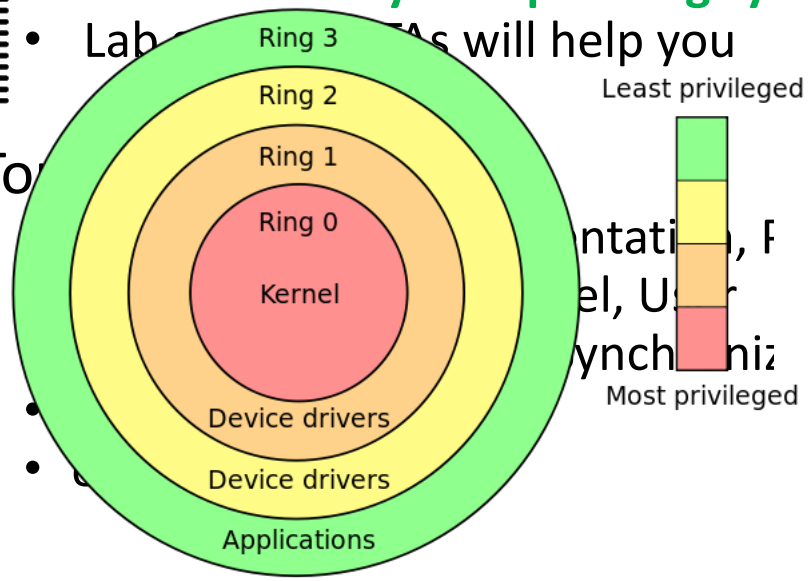
OS Kernel (Ring 0)



mental co

etails with

- You will **build your operating sy**
- Lab s will help you



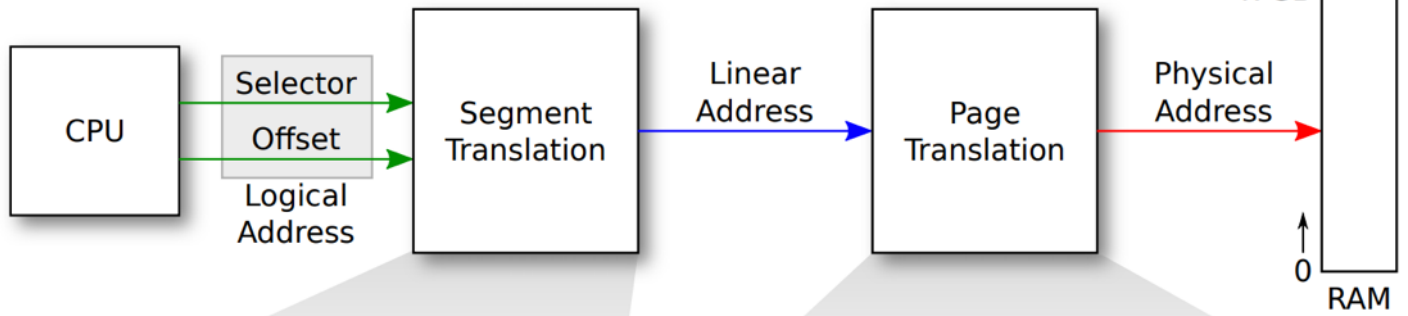
- To

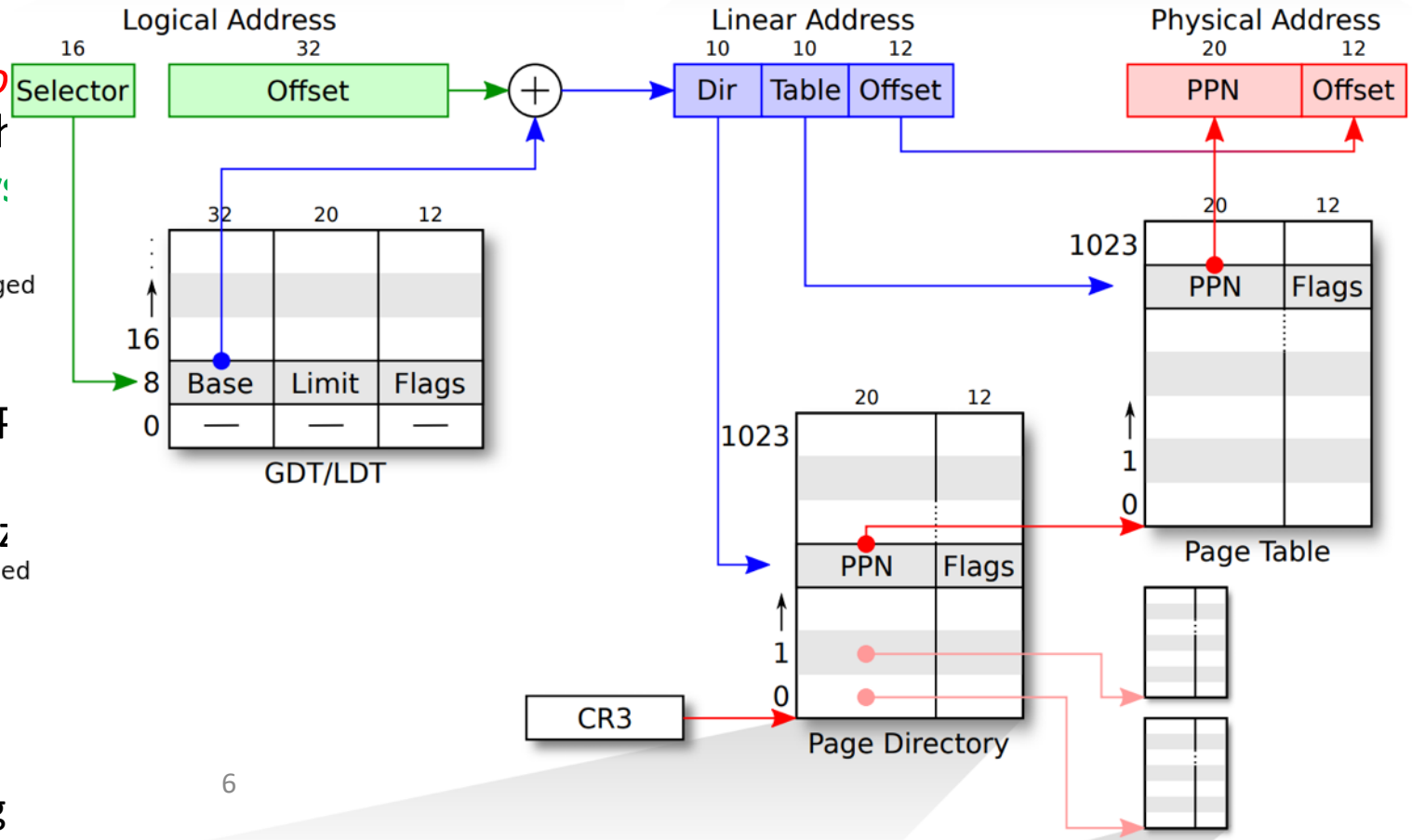
ntati , P

el, U

ynch niz

Image from: https://www3.nd.edu/~pbui/teaching





```
static void *
boot_alloc(uint32_t n)
{
    static char *nextfree;  // virtu
    char *result;

    // Initialize nextfree if this
    // 'end' is a magic symbol autor
    // which points to the end of tl
    // the first virtual address tha
    // to any kernel code or global
    if (!nextfree) {
        extern char end[];
        nextfree = ROUNDUP((char *)
    }
```

- Learn *high-level*

```
static inline physaddr_t
page2pa(struct PageInfo
{
    return (pp - pages)
}
```

- Virtual memory, S

```
// These variables are set in
pde_t *kern_pgdir;      // Ker
struct PageInfo *pages;     //
static struct PageInfo *page_f
```

- etc.

```
#define TRAPHANDLER_NOEC(name, num)                    \
    .globl name;                                       \
    .type name, @function;                             \
    .align 2;                                          \
    name:                                              \
    pushl $0;                                          \
    pushl $(num);                                      \
    jmp _alltraps
```

Labs


```
void
env_pop_tf(struct Trapframe *tf)
{
    asm volatile(
        "\tmovl %0,%%esp\n"
        "\tpopal\n"
        "\tpopl %%es\n"
        "\tpopl %%ds\n"
        "\taddl $0x8,%%esp\n" /* skip tf_trapno and tf_errcode */
        "\tiret\n"
        : : "g" (tf) : "memory");
    panic("iret failed");  /* mostly to placate the compiler */
}
```

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 app**lication?
  - 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/

Date	Lecture Topics(s)	Slides (Videos are from **Spring 2021**)	Lab Tutorial	Reading Materials	Due
Week #1					
4/4 Tue	Course Intro	• Lecture 1 ◦ recorded_lec	• Lab Setup ◦ Slides ◦ Video • Lab Setup Guide	• Intro to OS • x86 Assembly • GDB Tutorial 1 • GDB Tutorial 2 • GDB cheat sheet • tmux cheatsheet	
4/6 Thu	BIOS, Booting, and CPU	• Lecture 2 ◦ recorded_lec			
Week #2					
4/11 Tue	Virtual Memory	• Lecture 3 ◦ recorded_lec		• x86 Address Translation • Address Spaces • Address Translation • Segmentation	
4/13 Thu	Paging and Virtual Memory Translation	• Lecture 4 ◦ recorded_lec		• Page Table	Lab 1 Due Monday 4/17 11:59 pm

- **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

Note

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.

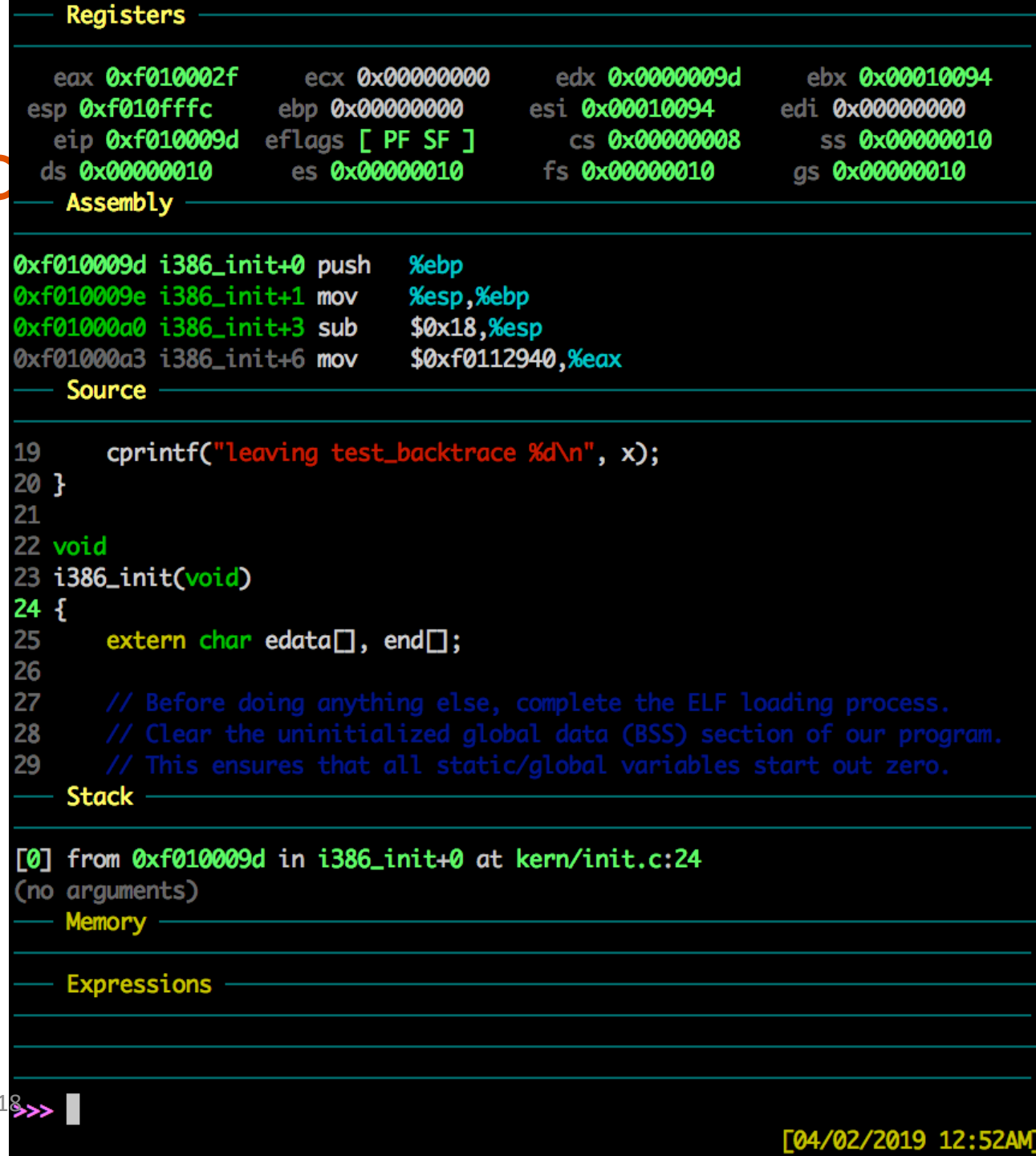
The Lab Could be Difficult

- 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)

```
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0
EAX=00000000 EBX=00010094 ECX=00000002 EDX=00000000
ESI=00010094 EDI=e0000000 EBP=f010ffd8 ESP=f010ffcc
EIP=f01014eb EFL=00000002 [-------] CPL=0 II=0 A20=1 SMM=0 HLT=0
ES =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
CS =0008 00000000 ffffffff 00cf9a00 DPL=0 CS32 [-R-]
SS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
DS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
FS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
GS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
LDT=0000 00000000 0000ffff 00008200 DPL=0 LDT
TR =0000 00000000 0000ffff 00008b00 DPL=0 TSS32-busy
GDT=     00007c4c 00000017
IDT=     00000000 000003ff
CR0=80010011 CR2=e0000000 CR3=00110000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
EFER=0000000000000000
Triple fault.  Halting for inspection via QEMU monitor.
```

The Lab Could be Diffic

- 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)

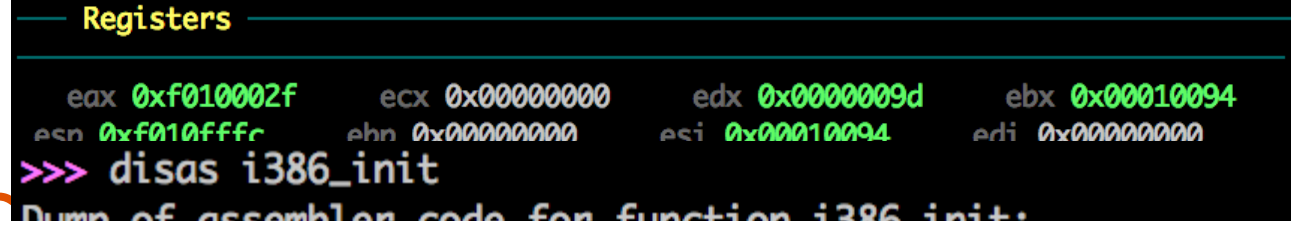


The Lab Could be Diffic

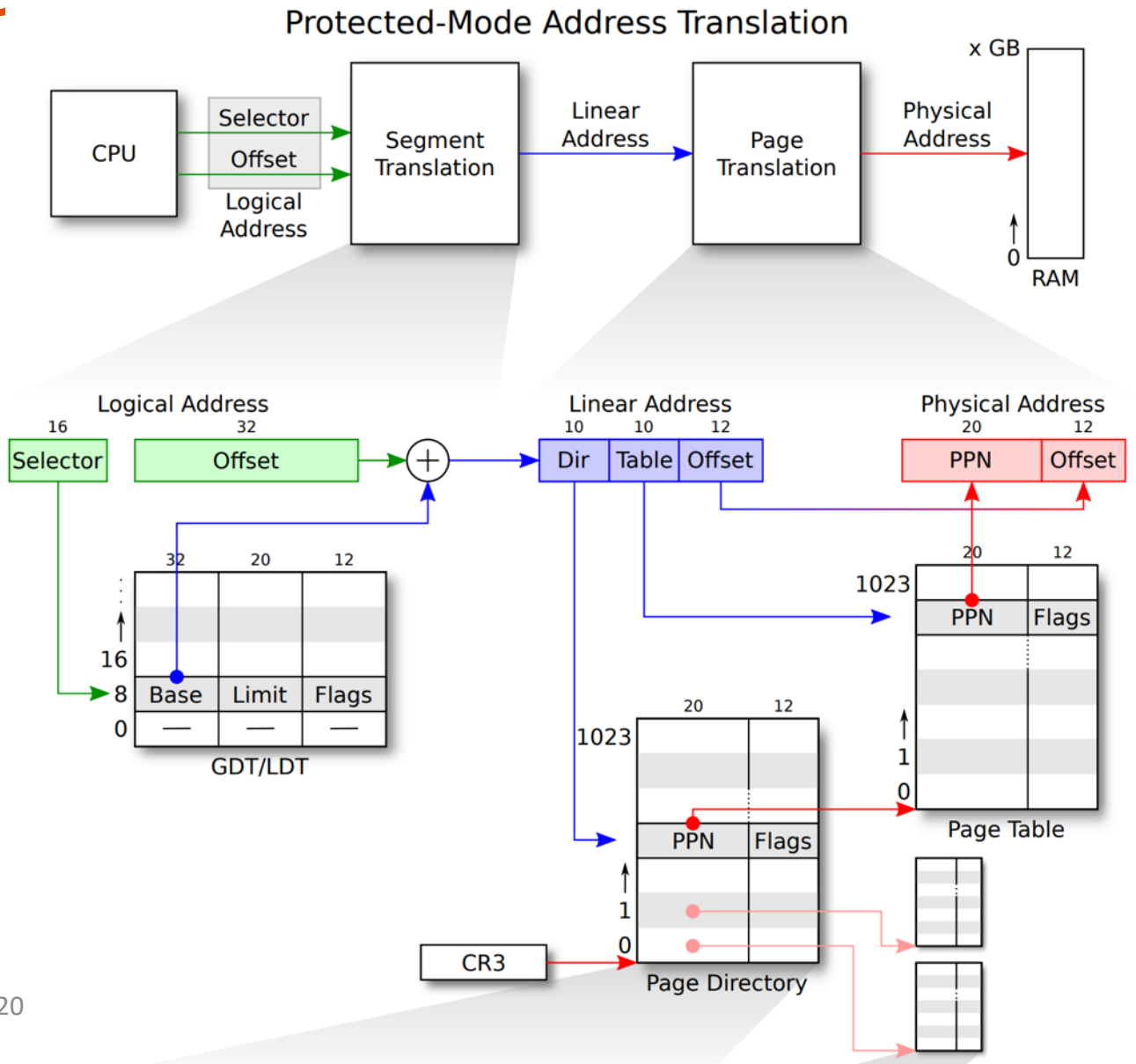
- Coding KERNEL code in C
  - Any memory error -> Triple fault…
- Use GDB for debugging OS Kernel
  - Get familiar to tools ASAP..
- Assembly Languages
  - Intel x86
- Control hardware specific data
  - Page table
  - Global descriptor table (GDT)
  - Interrupt descriptor table (IDT)

```
Registers
  eax 0xf010002f     ecx 0x00000000     edx 0x0000009d     ebx 0x00010094
>>> disas i386_init
Dump of assembler code for function i386_init:
=> 0xf010009d <+0>:     push   %ebp
   0xf010009e <+1>:     mov    %esp,%ebp
   0xf01000a0 <+3>:     sub    $0x18,%esp
   0xf01000a3 <+6>:     mov    $0xf0112940,%eax
   0xf01000a8 <+11>:    sub    $0xf0112300,%eax
   0xf01000ad <+16>:    mov    %eax,0x8(%esp)
   0xf01000b1 <+20>:    movl   $0x0,0x4(%esp)
   0xf01000b9 <+28>:    movl   $0xf0112300,(%esp)
   0xf01000c0 <+35>:    call   0xf01014a7 <memset>
   0xf01000c5 <+40>:    movl   $0x2,0x8(%esp)
   0xf01000cd <+48>:    movl   $0x0,0x4(%esp)
   0xf01000d5 <+56>:    movl   $0xe0000000,(%esp)
   0xf01000dc <+63>:    call   0xf01014a7 <memset>
   0xf01000e1 <+68>:    call   0xf010058f <cons_init>
   0xf01000e6 <+73>:    movl   $0x6c880,0x4(%esp)
   0xf01000ee <+81>:    movl   $0xf0101977,(%esp)
   0xf01000f5 <+88>:    call   0xf0100951 <cprintf>
   0xf01000fa <+93>:    movl   $0x5,(%esp)
   0xf0100101 <+100>:   call   0xf0100040 <test_backtrace>
   0xf0100106 <+105>:   movl   $0x0,(%esp)
   0xf010010d <+112>:   call   0xf01007c9 <monitor>
   0xf0100112 <+117>:   jmp    0xf0100106 <i386_init+105>
End of assembler dump.
```

The Lab Could be Diffic



- 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 Diffic

- 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)

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3 (3A, 3B, 3C & 3D): System Programming Guide

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

Paging Structure	Entry Name	Paging Mode	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML4 table	PML4E	32-bit, PAE	N/A		
		IA-32e	CR3	47:39	N/A (PS must be 0)
Page-directory-pointer table	PDPTE	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		IA-32e	PML4E	38:30	1-GByte page if PS=1[1]
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS=1[2]
		PAE, IA-32e	PDPTE	29:21	2-MByte page if PS=1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, IA-32e		20:12	4-KByte page

The Lab Could be Diffic

- Coding KERNEL code in C
  - Any memory error -> Triple fault
- Us
- As
- Co
  - Page table
  - Global descriptor table (GDT)
  - Interrupt descriptor table (IDT)

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

Intel® 64 and IA-32 Architectures
nual
& 3D):
Guide

		IA-32e	CR3	47:39	N/A (PS must be 0)
Page-directory-pointer table	PDPTE	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		IA-32e	PML4E	38:30	1-GByte page if PS=1[1]
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS=1[2]
		PAE, IA-32e	PDPTE	29:21	2-MByte page if PS=1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, IA-32e		20:12	4-KByte page

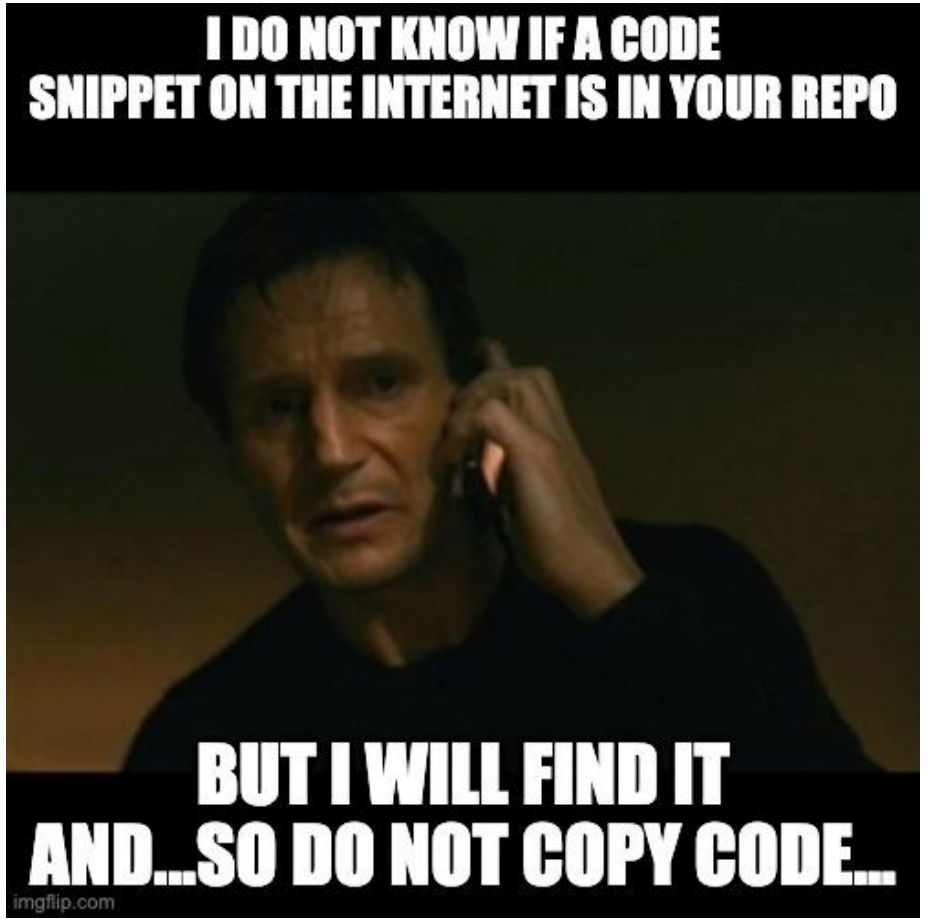
Lab Rules

- 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

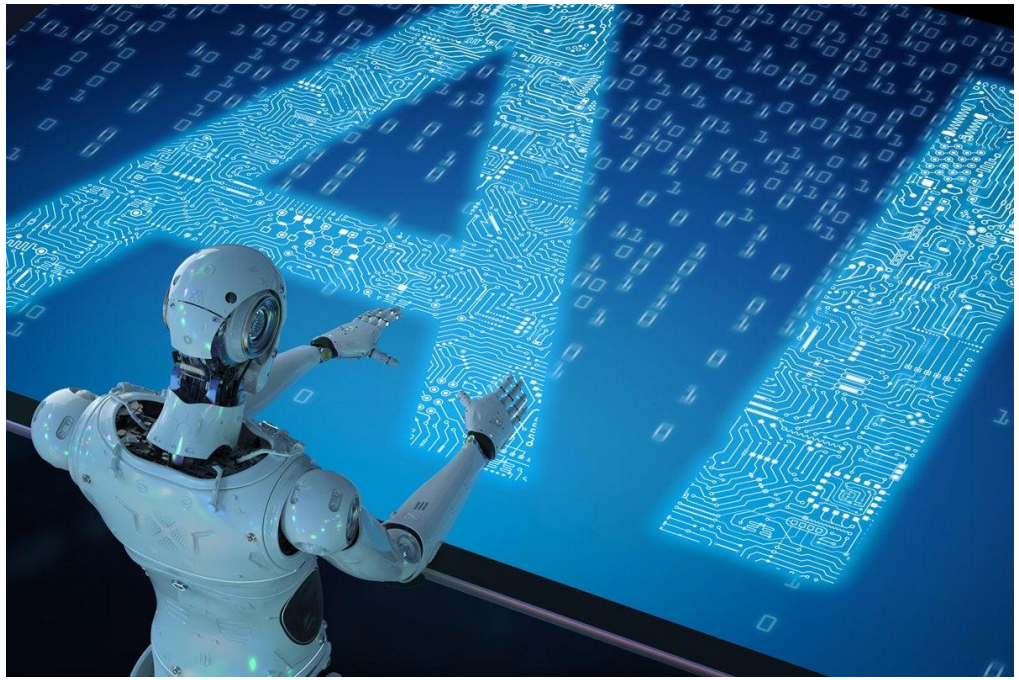
- You will be asked to submit a simple write-up for the assignment
  - Describe how you solve each exercise/questions
  - Mention your collaborators in the write-up
  - **Do not copy** other students' write-up
  - **Do not publish** your write-up online

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.

Date	Lecture Topics(s)	Slides (Videos are from **Spring 2021**)	Lab Tutorial	Reading Materials	Due
Week #0					
9/28 Thu	Course Intro	• Lecture 1 ◦ recorded_lec	• Lab Setup Guide ◦ Slides ◦ Video	• Intro to OS • x86 Assembly • GDB Tutorial 1 • GDB Tutorial 2 • GDB cheat sheet • tmux cheatsheet	
Week #1					
10/3 Tue	BIOS, Booting, and CPU	• Lecture 2 ◦ recorded_lec	• Lab1 ◦ slides1, slides2 ◦ video1, video2		
10/5 Thu	Virtual Memory	• Lecture 3 ◦ recorded_lec		• x86 Address Translation • Address Spaces • Address Translation • Segmentation	
Week #2					
10/10 Tue	Paging and Virtual Memory Translation	• Lecture 4 ◦ recorded_lec		• Page Table	Lab 1 Due Monday 10/9 11:59 pm
10/12 Thu	Virtual Memory Layout	• Lecture 5 ◦ recorded_lec	• Lab2 ◦ slides1, slides2 ◦ video1, video2	• Paging: Intro • Paging: TLBs • Paging: Smaller Tables	
Week #3					

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

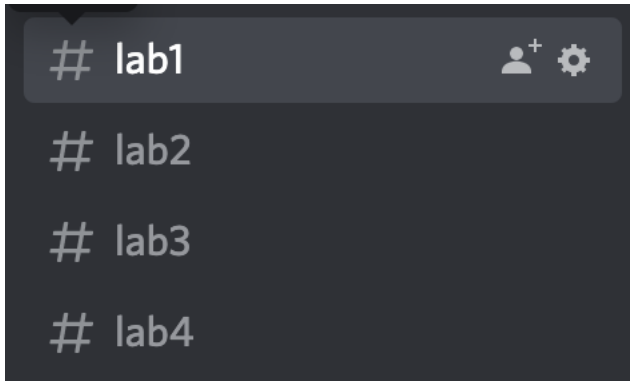
- Study in a group (discussions are highly encouraged!)
  - But please **write the code individually**!
- Follow tutorial video
- Ask questions (Discord)
- Understand your time budget (debugging will take **lots of your time**!)
  - **Plan ahead** to finish the labs on time
- Learn basic tools (e.g., 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