JOS Introduction

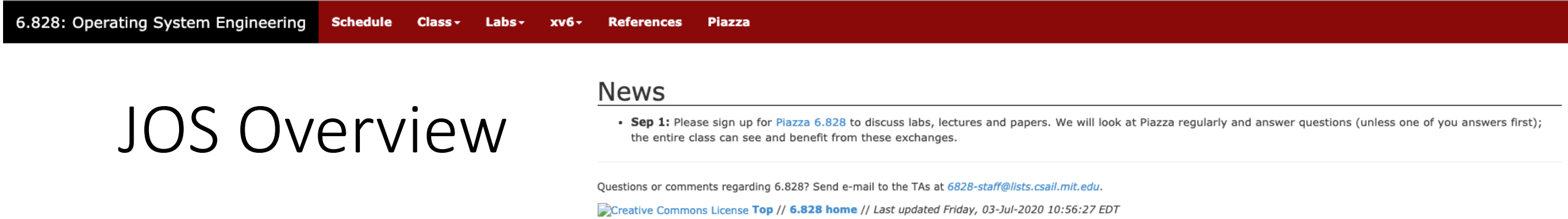
Ruizhe Huang（黄瑞哲）/ Qihang Xu（徐启航）

2024/09/11

Outline

- **JOS Overview**
- Course schedule & grading
- Some tips & tools
- Hands-on Lab 1: Bootloader

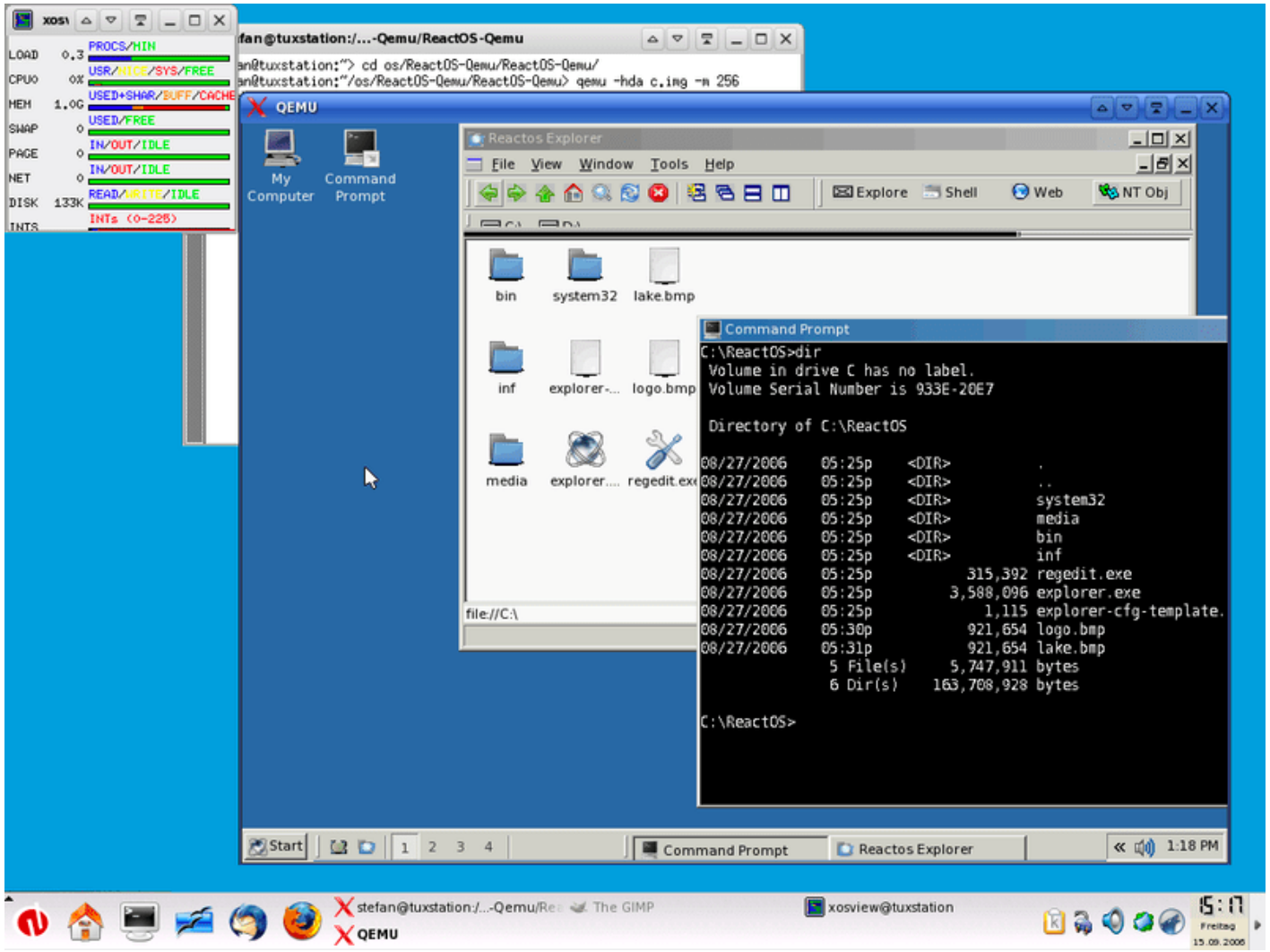
JOS Overview



- JOS is an x86-based OS designed by MIT for teaching
  - See website: https://pdos.csail.mit.edu/6.828/2018/
  - Note that we use course materials of **2018,** not 2019 or 2020
- To finish JOS, several labs are provided, in which we'll implement important OS components from scratch
  - Lab 1: bootloader
  - Lab 2: virtual memory management
  - Lab 3: user mode and system call
  - Lab 4: multitasking
  - Lab 5: file system
- We'll understand OS concepts better if we implement them

Lab environment

- JOS is a real OS that can run on real x86 hardware
- JOS will take over all hardware during runtime
- We cannot debug JOS as a common application
- Instead, we use **QEMU**, an emulator provides hardware emulation



- See https://pdos.csail.mit.edu/6.828/2018/tools.html and https://pdos.csail.mit.edu/6.828/2018/labguide.html for more details

Outline

- JOS Overview
- **Course schedule & grading**
- Some tips & tools
- Hands-on Lab 1: Bootloader

Lab Schedule

Lab	Start time	Report Submission	Weeks
Lab1	2024.9.9	2024.9.29	3 weeks
Lab2	2024.9.30	2024.10.20	3 weeks
Lab3	2024.10.21	2024.11.3	2 weeks
Lab4	2024.11.4	2024.11.24	3 weeks
Lab5	2024.11.25	2024.12.15	3 weeks

Quiz (Open Book)

- Objective: check the completion and the understanding of labs
- Schedule
  - There are one quiz in this semester
  - Quiz will be scheduled after lab5: 2024.12.23

OS Lab: Grading (30%)

- Complete the lab requirements: 60%
  - Complete the coding tasks
  - Submit the report
  - Due date is hard deadline!
    - Deduct 10% for each late day (3 days maximum for each lab)
- Quiz : 40%
- Bonuses: 10%
  - One or more challenge for each lab
  - Choose from a given list of challenges

Submission

- Archive both source code and report into **one zip file**
  - ID_name_lab#.zip, for example **2112345678_张三_lab1.zip**
- Report requirement
  - Recommended that the report should NOT exceed 3 pages of A4 paper
  - Describe any particularly noteworthy events you encountered during the lab
    - Architecture design of your code
    - Particularly sophisticated implementation
    - Memorable bugs
  - **State which challenge do you choose at the beginning of the report**
  - The report format is **PDF.** You can write with Latex, MS Word or Markdown but remember to **transfer the final format into PDF**.

Academic Integrity

- Plagiarizing is strictly prohibited
- YOU CAN NOT:
  - Share your code with others
  - Copy and paste directly from some blogs
  - Ask solutions from ChatGPT（你是一个学习操作系统的本科生，正在完成JOS相关实验，请完成lab1）
- YOU CAN:
  - Discuss with your classmates
  - Ask for help from the Internet / ChatGPT / TAs

Outline

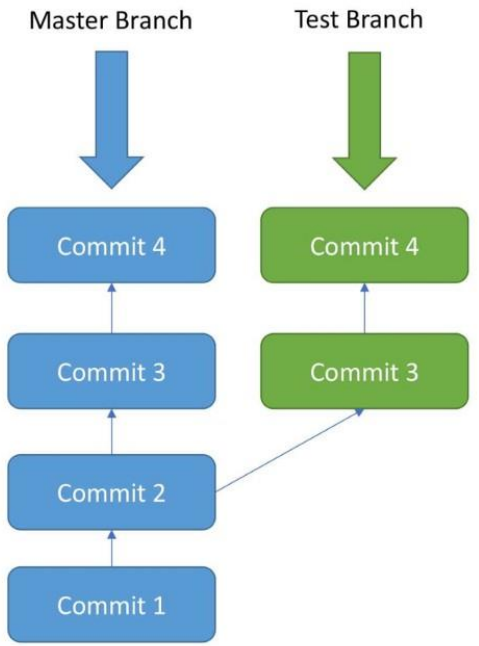
- JOS Overview
- Course schedule & grading
- **Some tips & tools**
- Hands-on Lab 1: Bootloader

Some tips

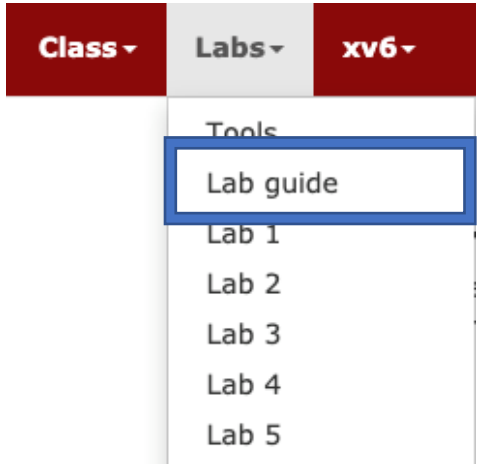
- Start early!
  - Debugging JOS can be time consuming
- **Read documents and understand existing code carefully before writing your own code**
  - JOS official lab guides
  - Intel Software Developer's Manual: https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4
  - OSDev Wiki: https://wiki.osdev.org/
- Maybe look for bugs in previous labs will help if you encounter strange bugs in current lab
  - JOS has an automated grader, but it only ensures basic correctness

Tools: git

- JOS is a large project, you need to manage your complex code
  - You may forget a previous small change
  - Backup with remote repository
- Frequently used commands
  - Git clone
  - Git add, git commit –m
  - Git status, git log
  - Git reset
  - Git branch / git checkout
  - Git push



Tools



- Makefile
  - Tells ``make'' how to compile and link a program
  - Make qemu
  - Make qemu-nox
  - Make qemu-gdb
  - Make qemu-nox-gdb

1:Makefile

```
helloworld : main.o print.o
    clang -o helloworld main.o print.o
main.o : main.c print.h
    clang -c main.c
print.o : print.c print.h
    clang -c print.c
clean :
    rm helloworld main.o print.o
~
```

1:print.c

```
#include"print.h"
void printhello(){
    printf("Hello, world\n");
}
~
```

1:main.c

```
#include "print.h"

int main(void){
    printhello();
    return 0;
}
```

1:print.h

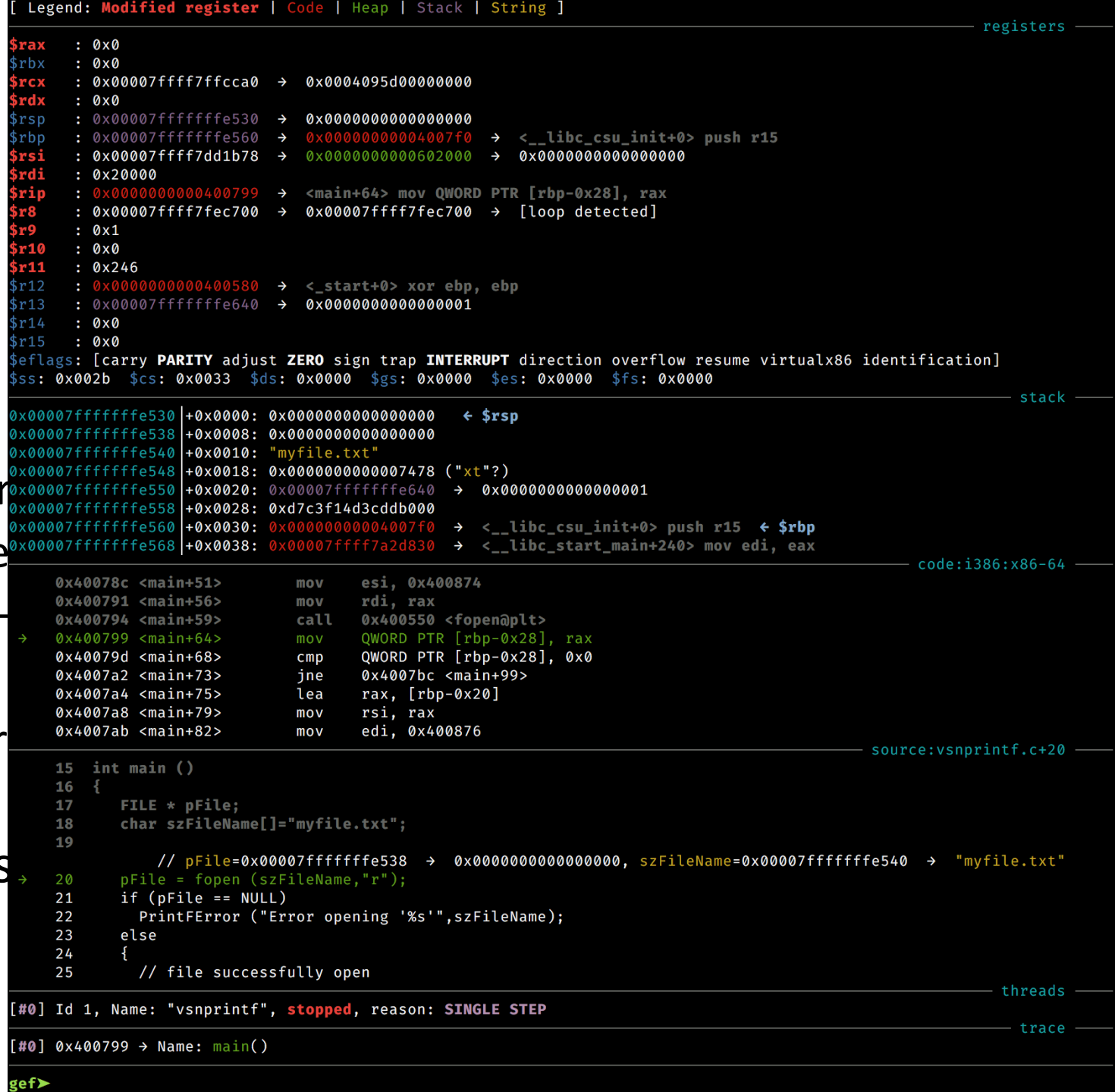
```
#include<stdio.h>

void printhello();
```

- Custom the Makefile to simplify your commands

Tools

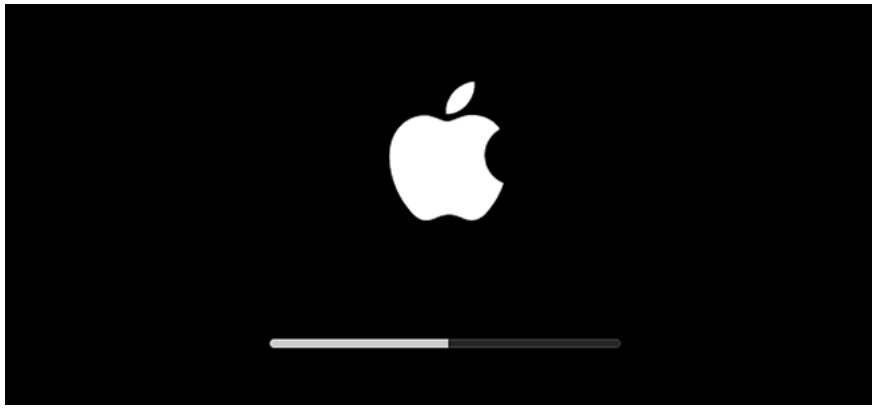
- GDB
- Basic usage:
  - run – r
  - continue – c
  - quit - q
  - next – n
  - step - s
  - breakpoints – b
  - info br
  - info re
  - print –
  - list – l
  - backtr
  - x
- plugin makes GDB easy to us
  - GEF, Pwndbg, peda

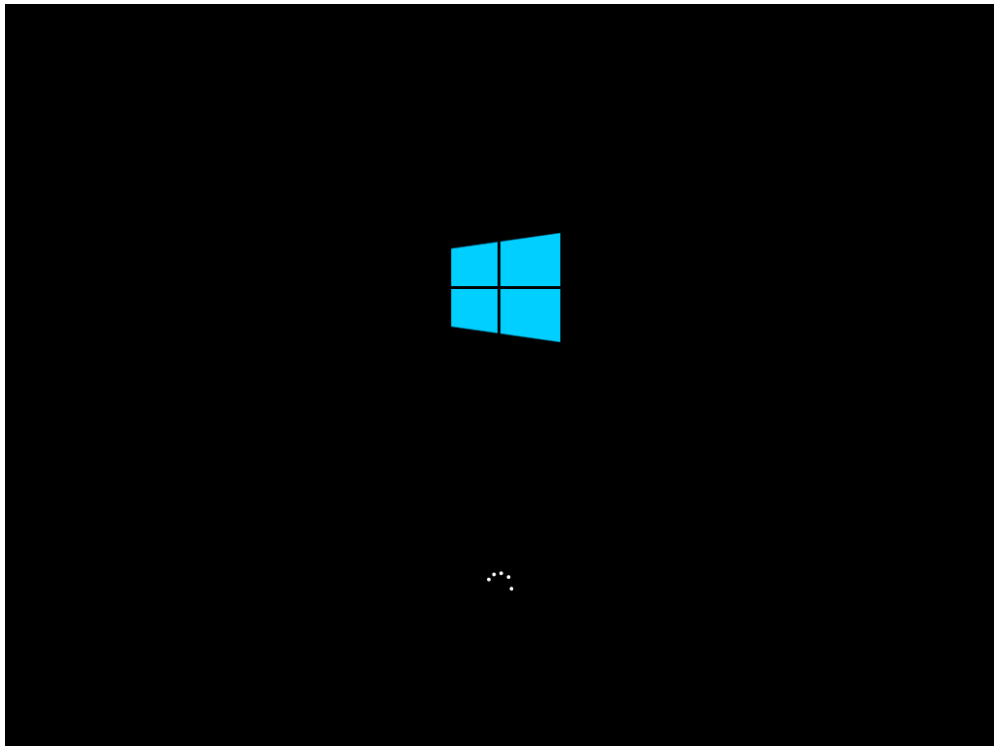


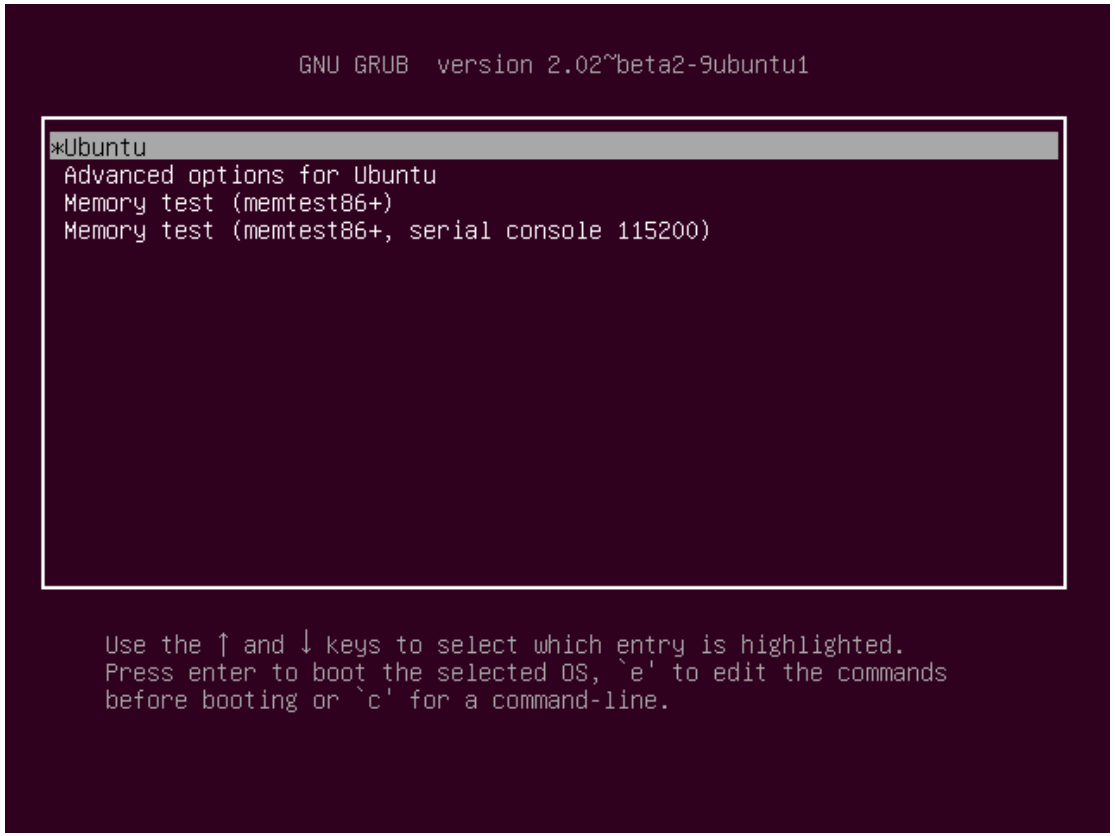
Outline

- JOS Overview
- Course schedule & grading
- Some tips
- **Hands-on Lab 1: Bootloader**

How does a PC boot?



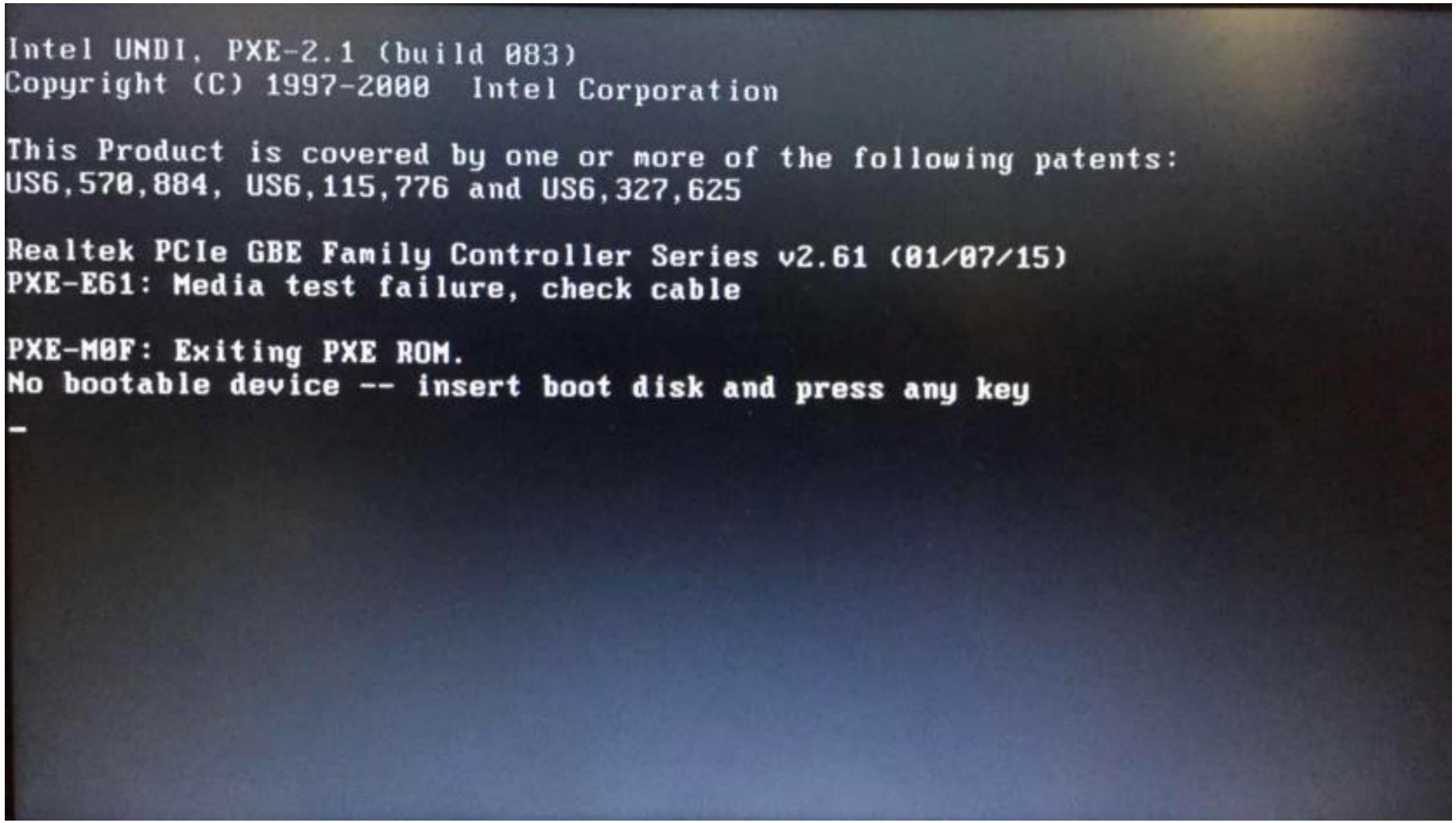




BIOS

- Recall what we've learned from ICS course
  - Once CPU is powered on, it fetches instructions from memory and modifies data in memory
  - The first instruction is stored in BIOS ROM
  - BIOS ROM is mapped to low-address space of RAM
  - The IBM PC starts executing at physical address `0xffff0`
    - how is the address calculated? CS:0xf000, PC:0xfff0, $CS << 4 + $PC
- CPU then executes pre-defined instructions in BIOS ROM and does some self-checking
- But what's next?

How your PC goes on strike…



Bootloader

- We need a **bootable device**
- Of course, not all devices (disks) are bootable
- Disks have sectors
  - Hard disks have 512-byte sectors
  - The first sector is used to flag whether the disk is bootable
  - MBR (master boot record) is the first sector ends with **0x55** and **0xAA.** This is manually set.
  - The disk is bootable ⇔ MBR exists
  - Bootloader are stored in MBR
- BIOS will recognize MBR if it exists on some disk, then load bootloader into RAM and finally jump to the first instruction of it
- cat *<MBR image>* | head –c 512 | hexdump [ndisasm –b 16]
- MBR+BIOS is outdated, but it is easy to understand for tutorial

OS loading

- In bootloader we need to load OS into RAM
- OS are stored in disk. How to load it?
  - x86 has special I/O instructions
  - Consult JOS lab 1 code for more details
- Finally, OS takes over the control flow
- Initializing and we enter the world of OS.

Lab 1 has begun

- Lab 1 due: 9/29
- Website: https://pdos.csail.mit.edu/6.828/2018/labs/lab1/
- What you need to do
  - Finish lab 1 according to instructions
  - Write lab 1 report
- What to submit
  - Lab 1 source code
  - Lab 1 report (in **PDF** format)
- Where to submit
  - http://course.pku.edu.cn/
- Don’t Panic and Start early

Linux/MacOS 配6.828-2018实验环境

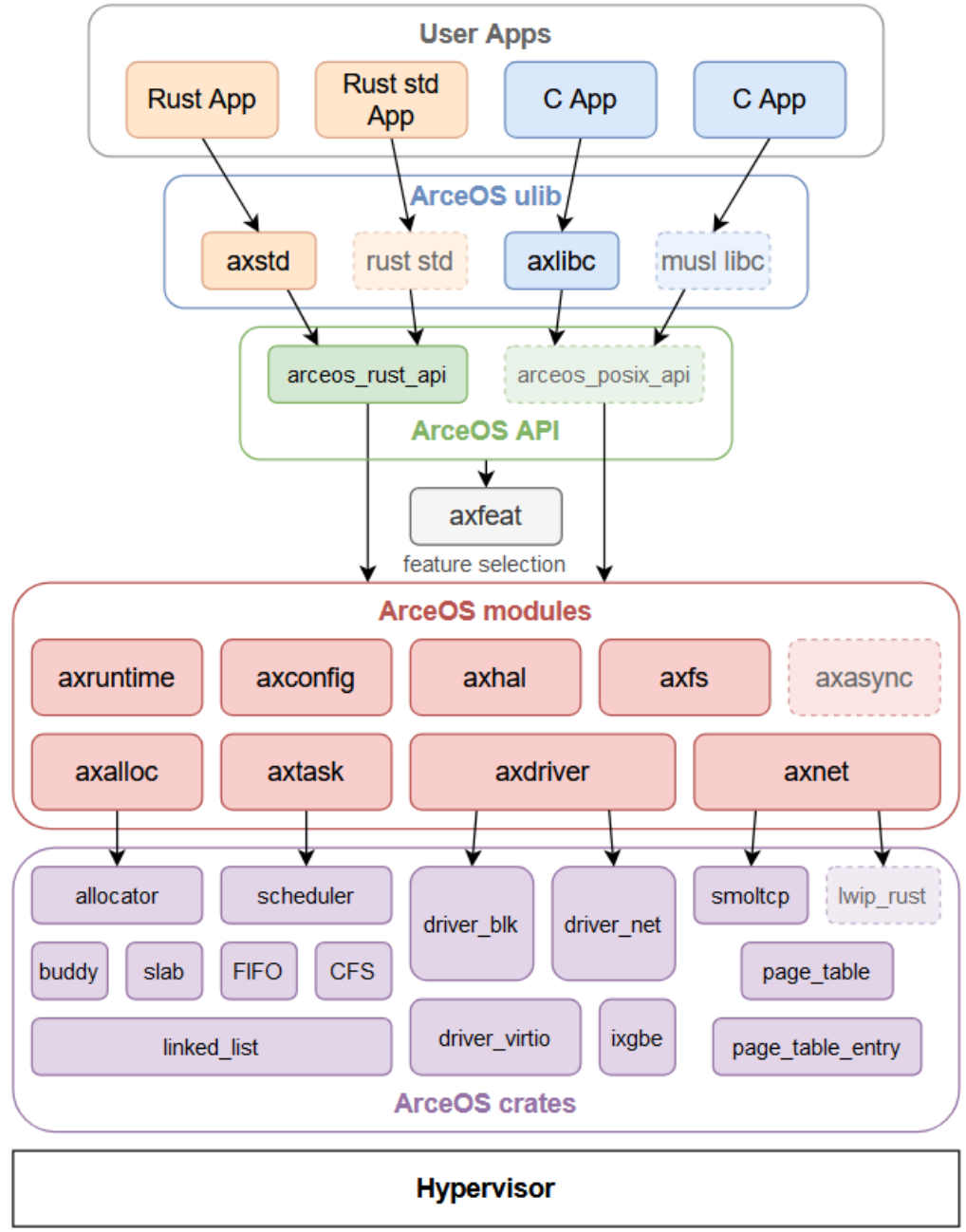
- https://zhuanlan.zhihu.com/p/383308283?utm_id=0

rCore：一个兼容Linux的Rust OS (by THU)

- 支持x86、RISC-V、ARM等多种架构
- Tutorial：使用Rust编写OS的入门级教程
- 新一代：
  - zCore：同时兼容Zircon (Fuchsia by Google)和Linux的OS内核
  - ArceOS：单一内核OS，具有组件化、模块化的特性
- 链接：
  - 项目主页：github.com/rcore-os
  - Tutorial：rcore-os.cn/rCore-Tutorial-Book-v3
  - rcore-os.cn/arceos-tutorial-book
- 任务指引：github.com/orgs/rcore-os/discussions

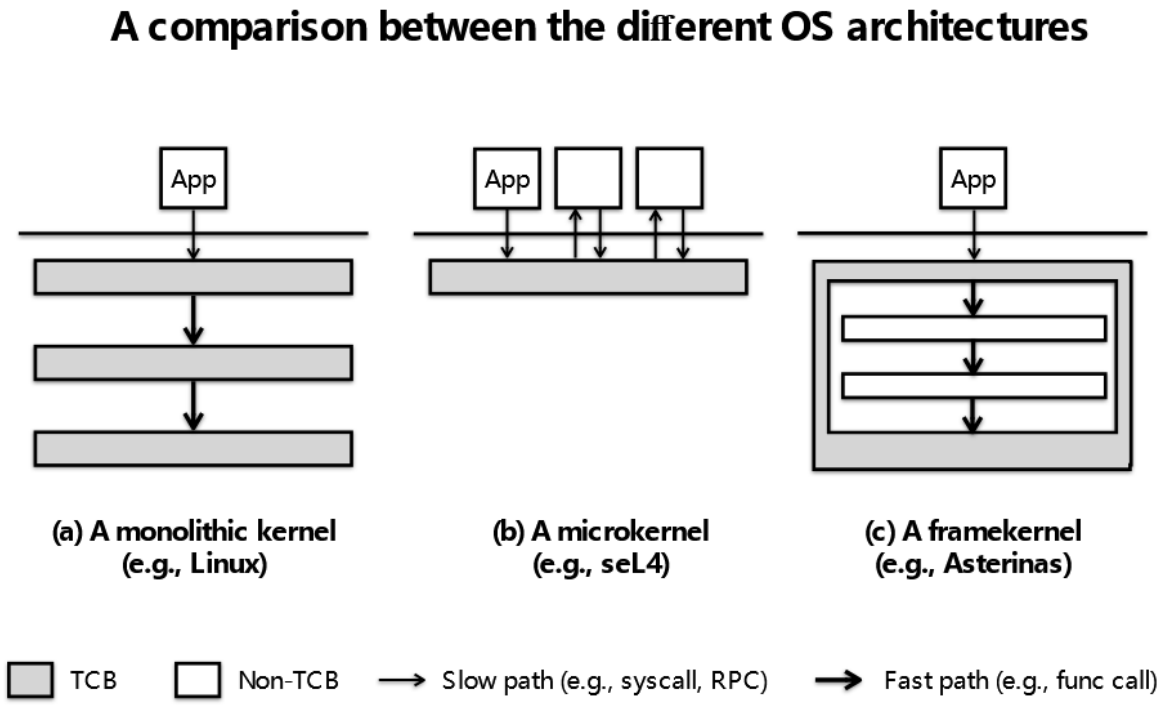
Discussions

↑ 7	实验小项目招新：基于arceos框架直接支持Linux App的宏内核 chyyuu started on Apr 18, 2023 in Ideas		106
↑ 1	实验小项目招新：Linux Device Driver in Rust chyyuu started on Apr 25, 2023 in Ideas		3
↑ 1	Arceos USB Host 驱动 ZR233 started on May 24 in General		0
↑ 1	2024 OS 比赛：基于 ArceOS 宏内核实现支持复杂 Linux APP Azure-stars started on Jul 1 in Ideas		5
↑ 1	实验小项目招新：比较具有同样对外接口但不同实现的内核模块差异性 chyyuu started on Aug 27, 2023 in Ideas		40
↑ 2	实验小项目招新：通过fuzzing&测例分析并查找Rust/C-based OS相关的bug chyyuu started on Aug 27, 2023 in Ideas		8
↑ 2	实验小项目招新：支持多种架构的 hal 层 crate yfblock started on Apr 3 in Ideas		12
↑ 2	实验小项目招新：微内核架构的arceos chyyuu started on Apr 18, 2023 in Ideas		4
↑ 2	实验小项目招新：loongarch --> x86的动态二进制翻译 chyyuu started on Sep 11, 2023 in Ideas		7
↑ 1	rCore-Tutorial-Book-v3 第一章 lovetreee asked on Apr 19 in Q&A · Unanswered		0
↑ 1	项目：基于 Intel TDX 的精简的可信计算运行环境 dramforever started on Jan 4 in General		6
↑ 4	实验小项目招新：物理网卡驱动(华山派、荔枝派、U740、星光二代等) chyyuu started on Aug 9, 2023 in Ideas		31



星绽Asterinas：框内核OS (by Ant/PKU)

- 隔离内核的核心代码和外围服务
- 核心代码成为可信单元，外围服务从语言层面禁止unsafe
- 将核心代码构造成标准库模式，优化OS的开发体验



- github.com/asterinas/asterinas
- asterinas.github.io/book
- 任务指引：
  - github.com/asterinas/asterinas/issues
  - 浏览项目代码，寻找感兴趣的点