CS 318 Principles of Operating Systems

Fall 2022

Lecture 1: Introduction

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Prof. Ryan Huang

Course Instructor

Prof. Ryan Huang

- Assistant Professor, joined Hopkins in 2017
  - https://cs.jhu.edu/~huang
- Lead the Ordered Systems Lab: https://orderlab.io
  - research on OS, Cloud and Mobile Computing, Systems Reliability
- Office: Malone 231

Office Hours

- Mon 4:30pm-5:30pm, Wed 9:30 am-10:30 am (or by appointment)

Lecture 1 Overview

COURSE OVERVIEW

ADMINISTRATIVE

WHAT IS AN OS?

WALK-THROUGH OF OS BASICS

Staff: Teaching Assistants

Yuzhuo Jing (TA)

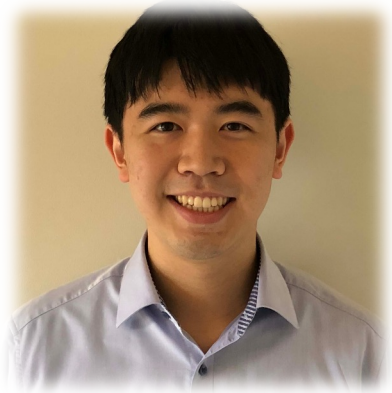
- Office Hours:

  Wed 4:30 - 5:30 PM
  Fri 9:30 - 10:30 AM
  Malone 122

Tony Pan (CA)

- Office Hours: TBA

Brian Choi (CA)



- Office Hours: TBA

Course Overview

An introductory course to operating systems

- Classic OS concepts and principles
- Prepare you for advanced OS and distributed system course
- OS concepts often asked in tech interview questions

A practice course for hands-on experience with OS

- Four large programming assignments on a small but real OS
- Reinforce your understandings about the theories

Bad News…

This is a TOUGH course

Requires proficiency in systems programming

- *"Low level (C) programming absolutely necessary."*
- *"Need to be fearless about breaking code (and then fixing it later)."*
- *"Need to be confident in touching and modifying large systems of code"*

Requires significant time commitment

- *"The projects are insanely time consuming"*
- *"The workload is much much heavier than your average CS course…Be prepared to spend entire weeks working on nothing but the material for this course."*

Good News

There aren't many such hardcore courses in CS curriculum ☺

- Typically the final checkmark for a *solid* CS degree
- You don't have to take it if you are not interested in it

It's hard, but rewarding in the end

- *"The project are very hard. But completing them is very rewarding."*
- *"You learn a lot about operating systems and computers in general."*

A highly valued skill after graduation

We will try our best to help you

Why Study Operating Systems?

Technology trends

CPU: 1.85 GHz dual-core

memory: 2 GB

price: $329

size: 9.4 in × 6.6 in

iPad (2017)

Why Study Operating Systems?

Technology trends

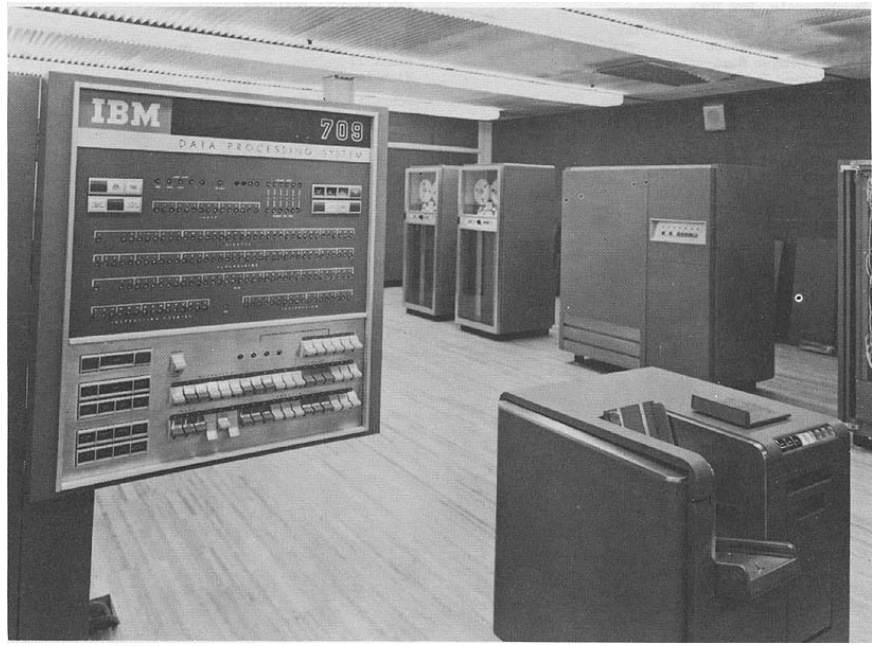
CPU: 1.85 GHz dual-core — **???** mult/div per sec.

memory: 2 GB — **???**

price: $329 — **???**

size: 9.4 in × 6.6 in — **???**

iPad (2017)



IBM 709 (c. late 1950~)

World's most powerful computer then

Quizzes

How many multi/div per second can an IBM 709 execute?

A: 100s

B: 1,000s

C: 10,000s

D: 1,000,000s

How much does an IBM 709 cost?

A: $100s

B: $1,000s

C: $10,000s

D: $1,000,000s

Why Study Operating Systems?

Technology trends

CPU: 1.85 GHz dual-core

memory: 2 GB

price: $329

size: 9.4 in × 6.6 in

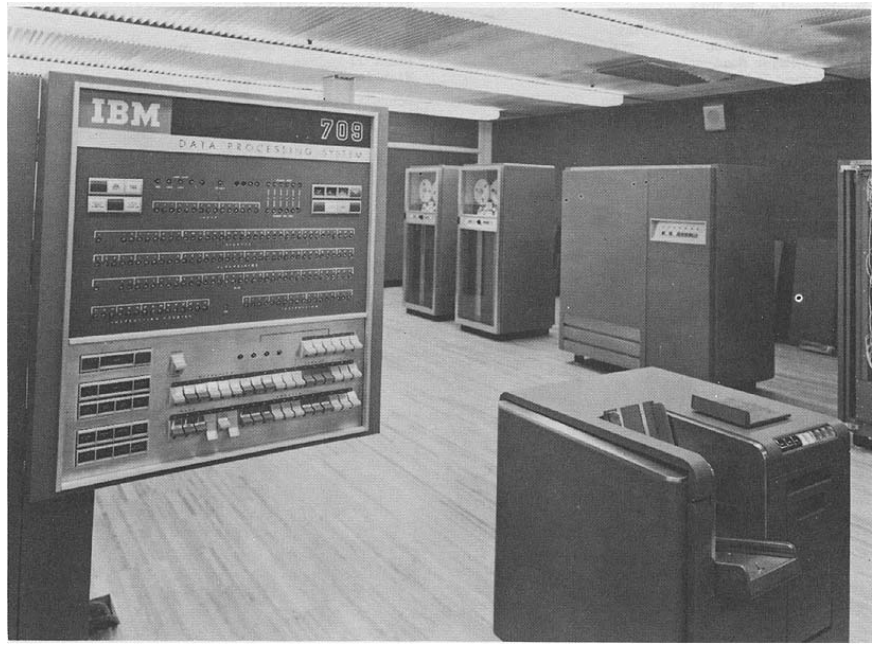
iPad (2017)

~4000 mult/div per sec.

32K 36-bit words

$2,630,000+

half room



IBM 709 (c. late 1950~)

World's most powerful computer then

Why Study Operating Systems?

Technology trends

manycore

3D stacked chip

persistent memory

accelerators

Tensor Processing Unit

They all need OS support to be useful!

smartphones

IoT device

self-driving cars

robots

data centers

...

Why Study Operating Systems?

An exciting time for OS designs

- New hardware, smart devices, self-driving cars, data centers, etc.
- Existing OSes face issues in performance, battery life, security, isolation

some of you

Pervasive principles for systems in general

- Caching, concurrency, memory management, I/O, protection

many of you

Complex software systems

- Many of you will go on to work on large software projects
- OSes serve as examples of an evolution of complex systems

many of you

Understand what you use

- System software tends to be mysterious
- Understanding OS makes you a more effective programmer

all of you

Course Materials

Course materials

- Lectures are the primary references
- Textbooks are supplementary readings
- Occasionally non-required papers

Topics Covered

Virtualization

Processes

Scheduling

Virtual Memory

Concurrency

Threads

Synchronization

Semaphores and Monitors

Persistence

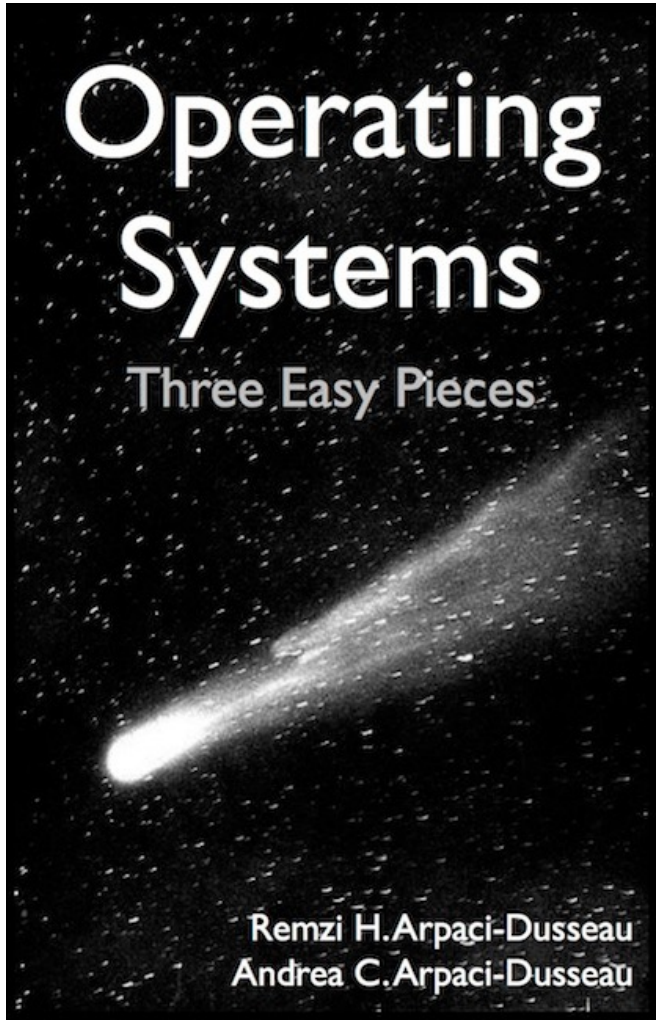
I/O

Disks

File Systems

Three Fundamental Pieces

Textbook



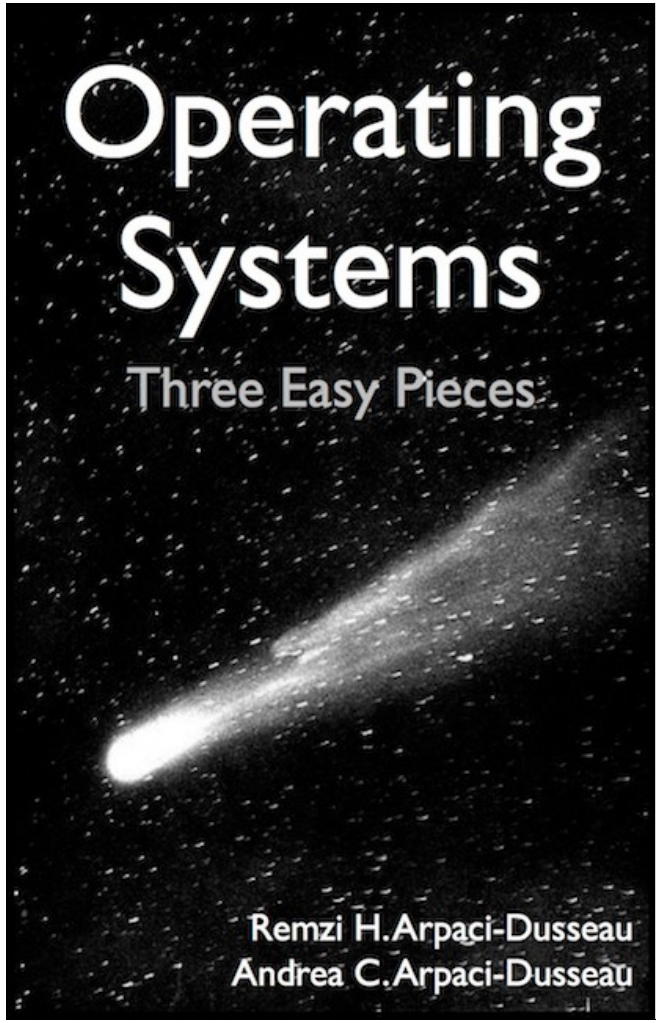
Operating Systems: Three Easy Pieces, Version 0.91

By *Remzi Arpaci-Dusseau* and *Andrea Arpaci-Dusseau*

http://from-a-to-remzi.blogspot.com/2014/01/the-case-for-free-online-books-fobs.html

Textbook

FREE



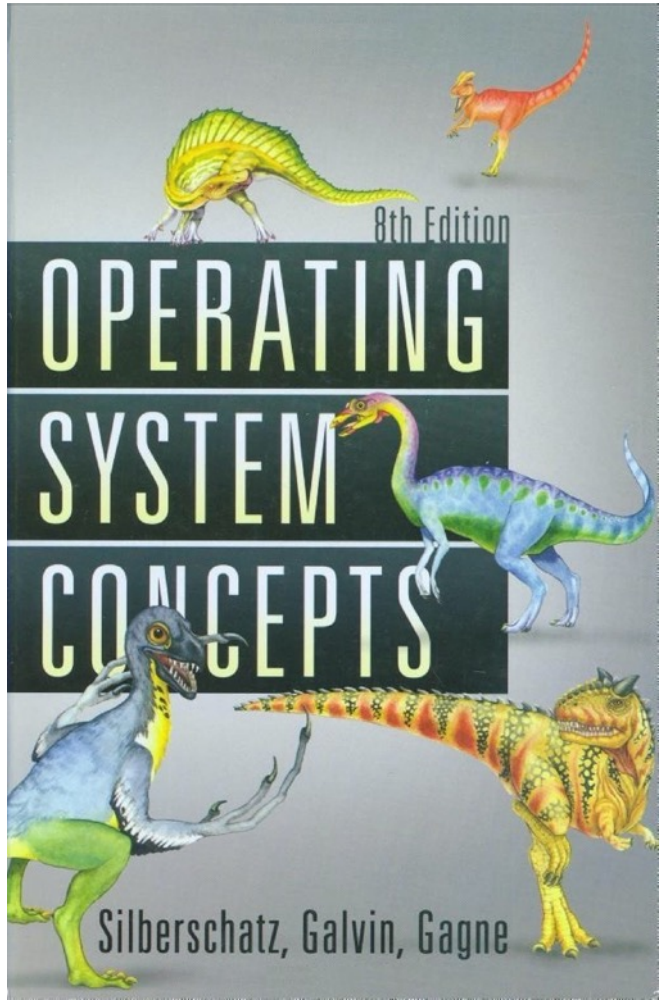
Operating Systems: Three Easy Pieces, Version 0.91

By *Remzi Arpaci-Dusseau* and *Andrea Arpaci-Dusseau*

http://from-a-to-remzi.blogspot.com/2014/01/the-case-for-free-online-books-fobs.html

Textbook

What killed the dinosaur?

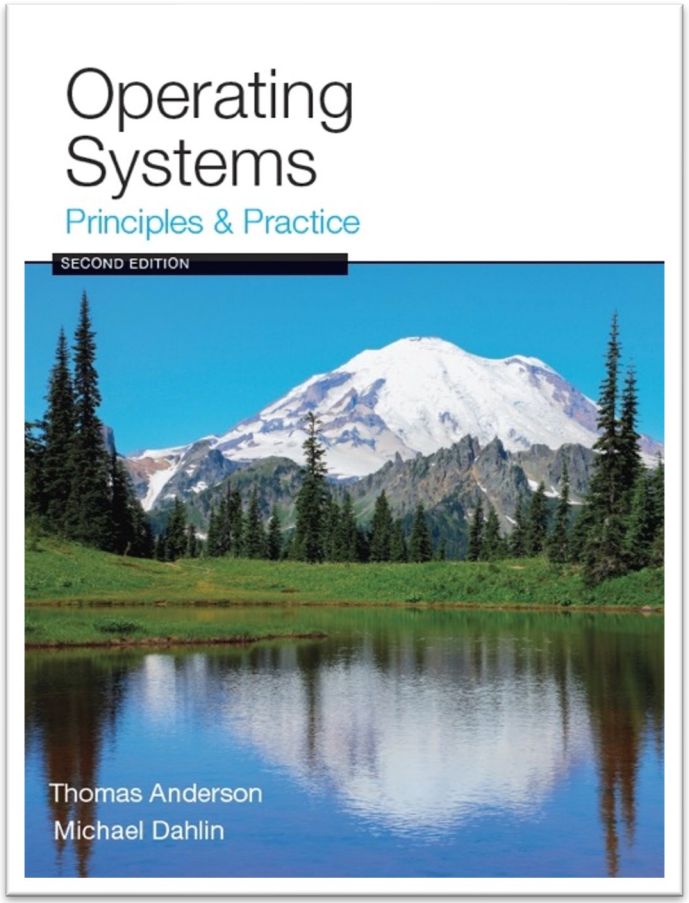


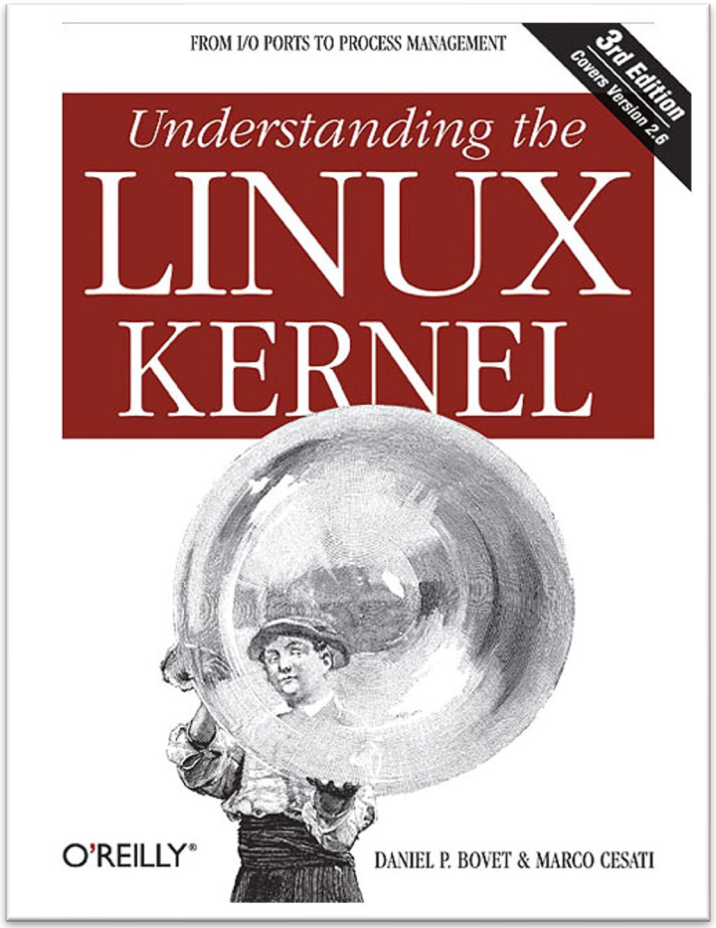
Operating Systems Concepts

By *Silberschatz*, *Galvin* and *Gagne*

Textbook

Other Recommended Textbooks





Important Links (1)

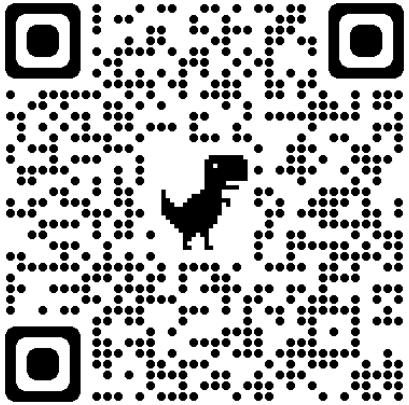
Course Website (check it often)

- https://www.cs.jhu.edu/~huang/cs318/fall22
- Course syllabus and schedule
- Lecture slides
- Homework handouts
- Project descriptions and references

Important Links (2)

Discussion Forum: CampusWire

- https://campuswire.com/p/GF03D61F4
- Access code: 2738
- Questions about project, lecture, exams



Staff mail list:

- cs318-staff@cs.jhu.edu
- administrative requests, sensitive questions, etc.

Homework

Several homework assignments throughout the semester

- help you check understanding about the lectures
- prepare you for the exams

The homework assignments will *not* be graded

- amount learned from doing homework is proportional to effort
- your choice on how much effort

Project Assignments

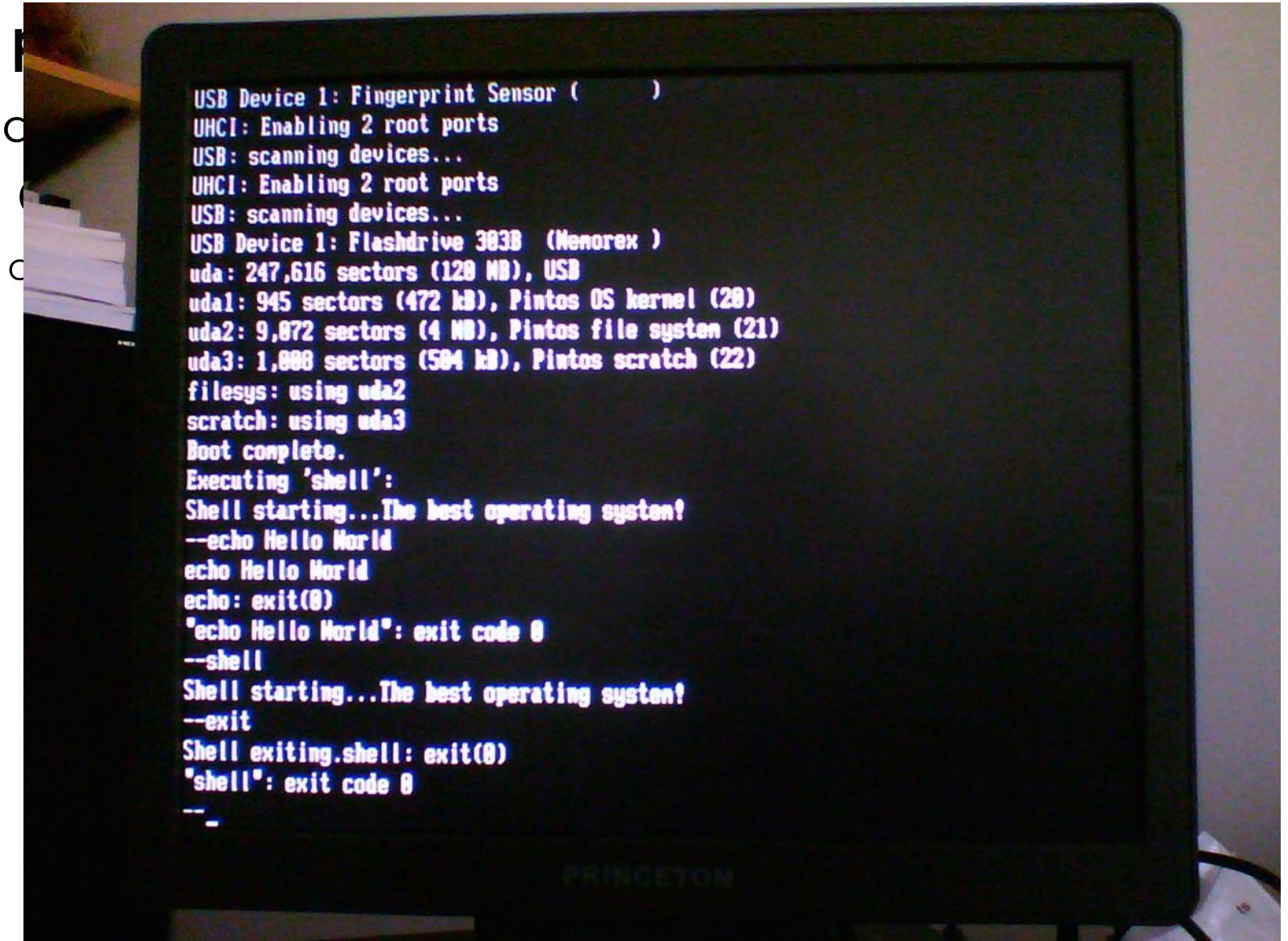
Implement parts of Pintos operating system

- Developed in 2005 for Stanford's CS 140 OS class
- Written in C, built for x86 hardware
  - can run on a real machine!

Project Assignments

Implement p

- Developed
- Written in C
  - can run c



Project Assignments

Implement parts of Pintos operating system

- Developed in 2005 for Stanford's CS 140 OS class
- Written in C, built for x86 hardware
  - can run on a real machine!
- Use hardware emulator (QEMU/Bochs) during development


QEMU
SeaBIOS (version rel-1.10.2-0-g5f4c7b1-prebuilt.qemu-project.org)
Booting from Hard Disk...
PiLo hda1
Loading.....
Kernel command line: -q run shell
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 523,468,800 loops/s.
hda: 1,008 sectors (504 kB), model "QM00001", serial "QEMU HARDDISK"
hda1: 218 sectors (109 kB), Pintos OS kernel (20)
hdb: 9,072 sectors (4 MB), model "QM00002", serial "QEMU HARDDISK"
hdb1: 8,192 sectors (4 MB), Pintos file system (21)
filesys: using hdb1
no swap device--swap disabled
Boot complete.
Executing 'shell':
Shell starting...
--echo "hello cs318"
echo "hello cs318"
echo: exit(0)
"echo "hello cs318"": exit code 0
--ls /
/:
echo
ls
cat
mkdir
rm
shell
ls: exit(0)
"ls /": exit code 0
--mkdir home
mkdir: exit(0)
"mkdir home": exit code 0
--ls /
/:
echo
ls
cat
mkdir
rm
shell
home
ls: exit(0)
"ls /": exit code 0
--_

Project Assignments (2)

One setup lab (lab 0)

- due next Thursday (done individually)

Four substantial labs:

- Required: Threads, User processes, Virtual memory
- Optional: File system

Implement projects in groups of up to 3 people

- Start picking your partners today

Warning: each project requires significant time to complete

- Don't wait until the last minute to start!!

Project Assignments (3)

Automated tests

- All tests are given so you immediately know how well your solution performs
- You either pass a test case or fail, there is *no* partial credit

Design document

- Answer important questions related to your design for a lab

Coding style

- Can your group member and TAs understand your code easily?

Project Design and Style

Must turn in a design document along with code

- Large software systems not just about producing working code
- We supply you with templates for each project's design doc

TAs will manually inspect code

- e.g., must actually implement the design
- must handle corner cases (e.g., handle `malloc` failure)
- will deduct points for error-prone code

Code must be easy to read

- Indent code, keep lines and functions short
- Use a consistent coding style
- Comment important structure members, globals, functions

Project Lab Environment

The CS department ugrad and grad lab machines

- Running Linux on x86
- The toolchain already setup

You may also use your own machine

- We have written detailed instructions for setting up the environment
  - https://cs.jhu.edu/~huang/cs318/fall22/project/setup.html
- Unix and Mac OS preferred. Windows needs VMs
- Pre-built VM image provided

Exams

Two Midterm Exams

- Exam 1 covers materials in the first half of class
- Exam 2 covers materials in the second half of class
- Include project questions

Grading

Midterm Exam 1: 15%

Midterm Exam 2: 20%

Project: 65%

- 1 warm-up lab + 3 major labs
  - Lab 3b is optional for 318-section students
- For each project
  - 70% based on passing test cases
  - 30% based on design document and style

Late Policies

Late submissions receive penalties as follows

- 1 day late, 15% deduction
- 2 days late, 30% deduction
- 3 days late, 60% deduction
- after 4 days, no credit

Each team will have a total of 6-day grace period

- can spread into 4 projects
- for interview, attending conference, errands, etc., no questions asked
- use it wisely, strongly suggest to reserve it for later labs (lab3)

Collaboration and Cheating Policies (A)

Collaboration

- Explaining a concept to someone in another group
- Discussing algorithms/testing strategies with other groups
- Helping someone else (in another group) debug

Collaboration and Cheating Policies (B)

Do not look at other people's solutions

- Including solutions online
  - This means copying code from GitHub will get you into big trouble
- We will run comprehensive tools to check for potential cheating

Do not publish your own solutions

- online (e.g., on GitHub) or share with other teams

Cite any code that inspired your code

- If you cite what you used, it won't be treated as cheating
  - in worst case, we deduct points if it undermines the assignment

Do Not Cheat

It *will* be caught

The consequence is very high

Truth: you can always get better outcome by not cheating

How *Not* to Pass CS 318?

Do not come to lecture

- The slides are online and the material is in the book anyway
- Lecture walks you through difficult materials and tells you the context

Do not do the homework

- It's not part of the grade
- Concepts seem straightforward...until you apply them
- Excellent practice for the exams, and project

How *Not* to Pass CS 318?

Do not ask questions in lecture, office hours or online

- It's scary, I don't want to embarrass myself
- Asking questions is the best way to clarify lecture material
- Office hours and email will help with homework, projects

Wait until the last couple of days to start a project

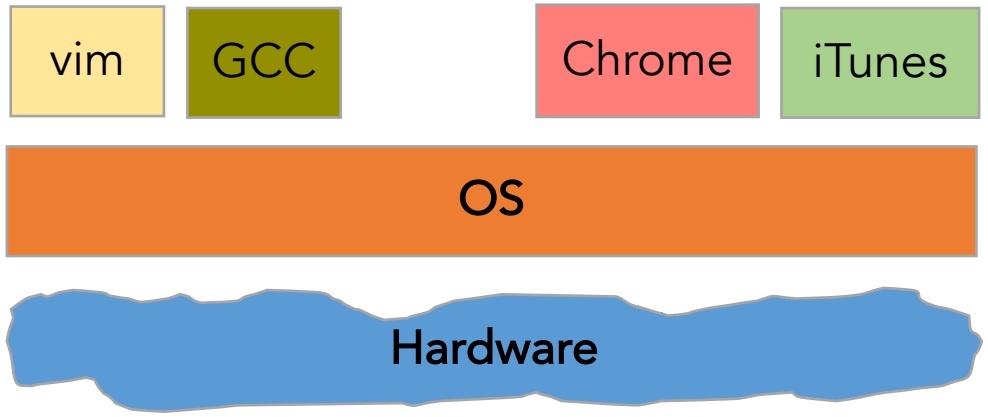
- We'll have to do the crunch anyways, why do it early?
- The projects cannot be done in the last few days
- Repeat: **The projects cannot be done in the last few days**
- (p.s. The projects cannot be done in the last few days)

Questions

Before we start, any questions?

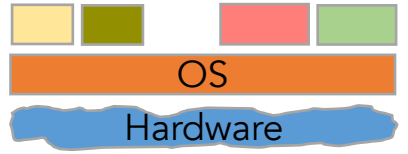
What Is An Operating System?

Layer between applications and hardware

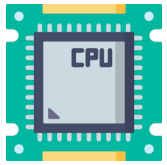


All the code that you didn't have to write to implement your app :)

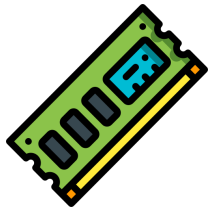
OS and Hardware



Manage hardware resources

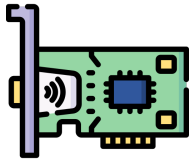


Computation

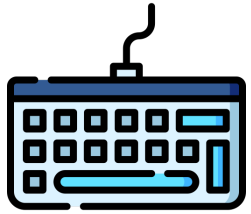


Volatile storage

Persistent storage



Communication

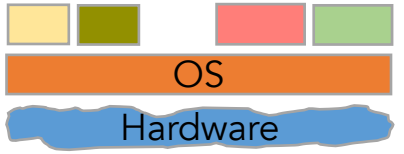


I/O

Provides abstractions to hide details of hardware from applications

- Processes, threads
- Virtual memory
- File systems
- …

OS and Hardware (2)



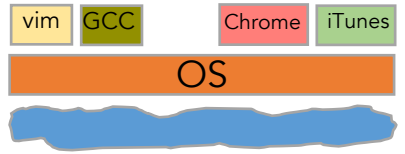
Mediate accesses from different applications

- Who has access at what point for how much/long

Why? Benefits to applications:

- Simpler (no tweaking device registers)
- Device independent (all network cards look the same)
- Portable (across Win95/98/ME/NT/2000/XP/Vista/7/8/10)

OS and Applications



Virtual machine interface

- Each program *thinks* it owns the computer

Provides protection

- Prevents one process/user from clobbering another

Provides sharing

- Concurrent execution of multiple programs (time slicing)
- Communication among multiple programs (pipes, cut & paste)
- Shared implementations of common facilities, e.g., file system

Questions to Ponder

What is part of an OS? What is not?

- Is the windowing system part of an OS?
- Is the Web browser part of an OS?
- This very question leads to different OS designs

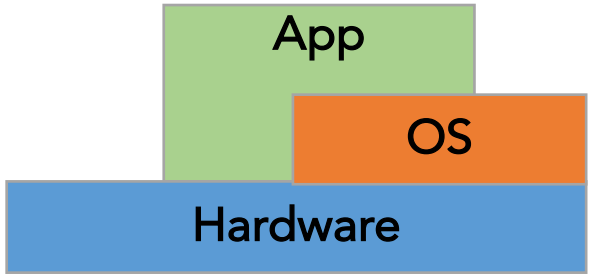
How different are popular OSes today?




Walk-through of OS basics

A Primitive Operating System

Just a library of standard services



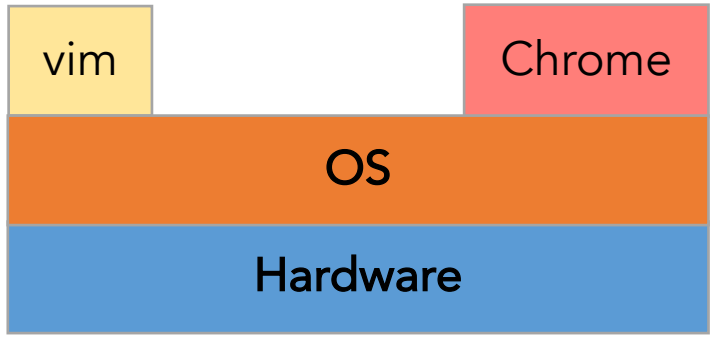
Simplifying assumptions

- System runs one program at a time
- No bad users or programs

Problems: poor utilization

- ...of hardware (e.g., CPU idle while waiting for disk)
- ...of human user (must wait for each program to finish)

Multitasking



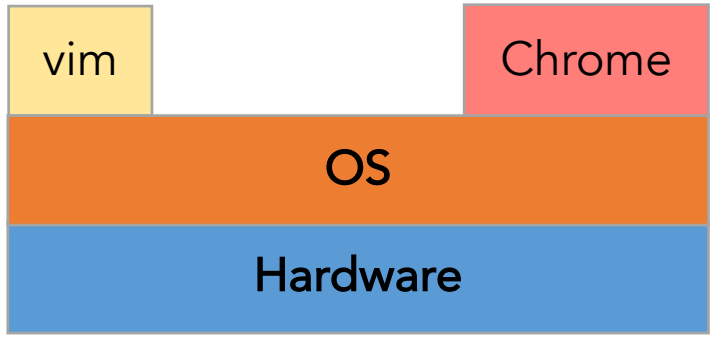
Idea: more than one process can be running at once

- When one process blocks (waiting for disk, network, input, etc.) run another process

How? mechanism: context-switch

- When one process resumes, it can continue from last execution point

Multitasking



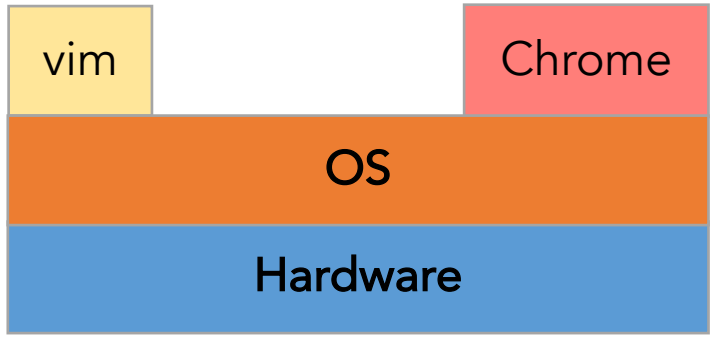
Idea: more than one process can be running at once

Mechanism: context-switch

Problems: ill-behaved process

- go into infinite loop and never relinquish CPU
- scribble over other processes' memory to make them fail

Multitasking



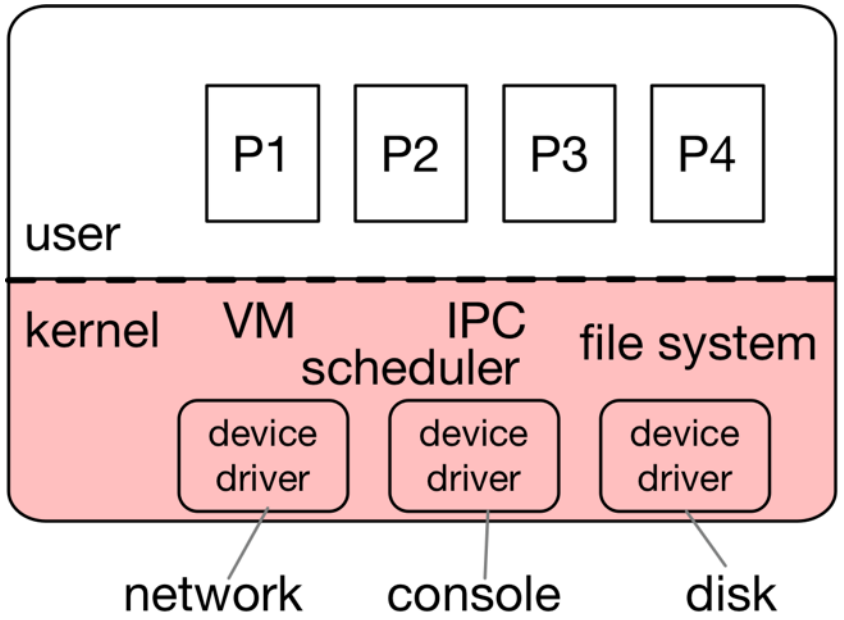
Problems: ill-behaved process

- go into infinite loop and never relinquish CPU
- scribble over other processes' memory to make them fail

Solutions:

- scheduling: fair sharing, take CPU away from looping process
- virtual memory: protect process's memory from one another

Typical OS Structure

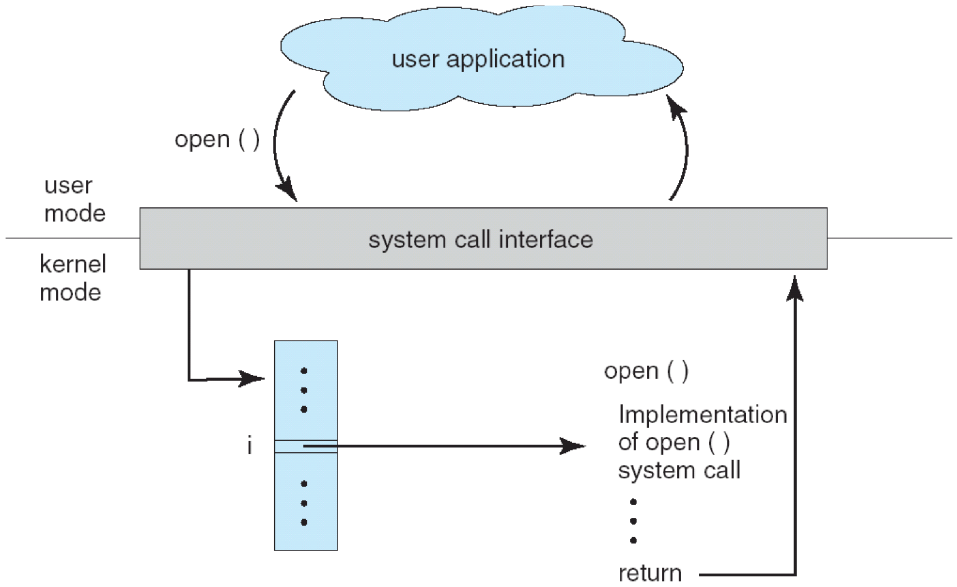


Most software runs as user-level processes (P[1-4])

OS kernel runs in privileged mode (shaded)

System Calls

```c
#include <fcntl.h>
#include <unistd.h>
int main()
{
  int fd = open("cs318.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0) {
    write(2, "Failed to open cs318.txt\n", 25);
    _exit(1);
  }
  write(fd, "Hello, OS!\n", 11);
  close(fd);
  return 0;
}
```



Applications can invoke kernel through system calls

- Special instruction transfers control to kernel
- ...which dispatches to one of few hundred syscall handlers

System Calls (continued)

The *only* way for an application to invoke OS services

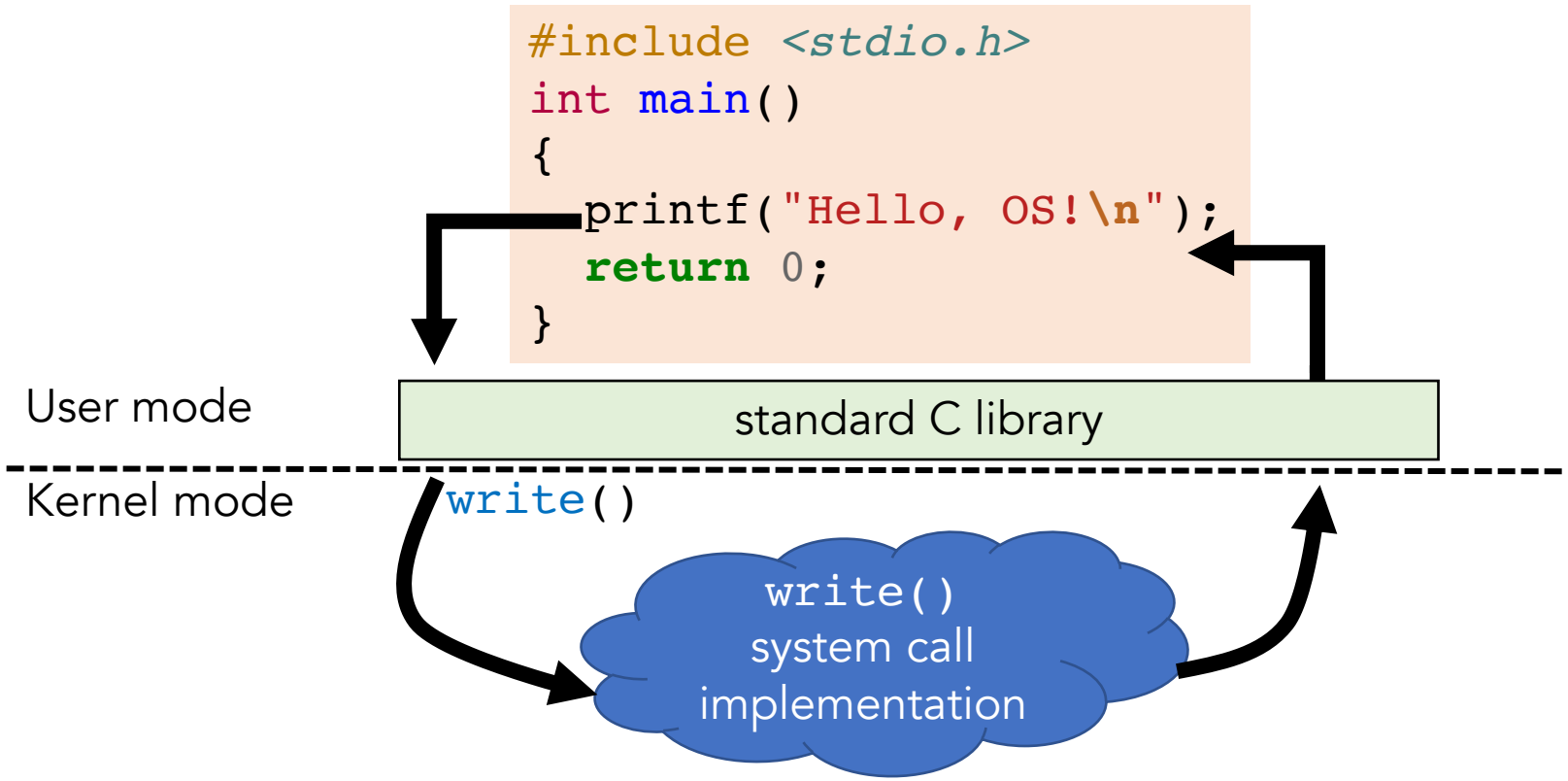
Goal: Do things application can't do in unprivileged mode

- Like a library call, but into more privileged kernel code

Kernel supplies well-defined system call interface

- Applications set up syscall arguments and trap to kernel
- Kernel performs operation and returns result

System Calls (continued)



Standard library calls are built on syscalls

For Next Class...

Browse the course web

- https://cs.jhu.edu/~huang/cs318/fall22

Sign up on Campuswire

Read Chapters 1 and 2

Setup Pintos and read its documentation

- Work on Lab 0, attend lab 0 overview session this Thursday

Looking for project partners

For Next Class...

Browse the course web

- https://cs.jhu.edu/~huang/cs318/fall22

Sign up on Cam

Read Chapters

Setup Pintos an

- Work on Lab 0,

Looking for pro