

Advanced Operating Systems (CS 202)

Instructor: Heng Yin

Professor

Department of Computer Science & Engineering

heng@cs.ucr.edu

<https://www.cs.ucr.edu/~heng>

Who I am and My Expectations



- I do research in system and software security
 - I know some aspects about modern operating systems very well (Windows/Linux/Android, etc.)

- Expect to discuss and learn with you altogether
 - Read papers carefully
 - Actively participate in class discussions
 - Your participation counts 10% of your final grade!

Today



- › Course organization and mechanics
- › Introduction to OS

What is this course about?



- How has the role of the operating system evolved over time?
 - How does the past inform the present?
- What are the principles that underlie Operating Systems?
- What are current and future trends in OS?
- Make it real: lab assignments to get some experience with OS development
- *Get you ready to do Systems research*

Some topics we will cover



- Operating Systems models and how they evolved
 - Monolithic kernels, micro-kernels, ...
 - extensibility, protection, performance
- Concurrency:
 - Synchronization and Scheduling
 - Multicore OS
- File systems:
 - Local, networked, distributed, internet scale
- Other advanced topics (if we have time)

Pre-requisites



- CS 202 is a tier-1 course
 - Will recap basics of OS
 - But you are expected to read textbook chapters for background
- To do well, you must understand all lecture materials and read required materials
- Architecture, networks, distributed systems courses are also a plus.

Class format

- For every topic:
 - Some overview
 - Read related book chapters before the class
 - Discuss research papers
- Research papers:
 - Critique for some required papers (1 paper most weeks)
 - You are responsible for required papers and material discussed in class

Questions while reading papers



- › What are the primary goals (hypothesis)?
 - › 2 sentence elevator pitch
- › Why did the authors do what they did the way they did?
 - › Understand motivation for design
- › What were the driving forces for the paper at the time it was written?
- › What parts still hold? What parts don't?
- › How did they experiment to support their ideas?
- › **Write the review by yourself, do NOT leverage online tools such as chatgpt!**

Reading Research Papers



- Guidelines for reading papers
 - Make sure to identify authors' goals and assumptions. Not always directly stated.
 - Look for high-level takeaways.
 - **Simulate the whole process in your head**
 - Follow a multi-pass reading strategy
 - Pass1: Get overview. Pass2: Read details and make notes. Pass3: Re-read details to evaluate.
 - Think how techniques used are applicable today. Identify extensions.

Projects



- › 3 Lab Assignments
 - › Modifications on xv6 (<https://github.com/mit-pdos/xv6-riscv>)
 - › Related to the topics discussed
- › All labs are done in groups (up to 2 students)
 - › You can change groups for each lab
- › Submission
 - › Short report & code per group on eLearn
 - › Brief summaries of individual contributions
 - › Short demo
- › Deadlines are **hard!**
- › Late policy
 - › 10% penalty per day; up to 3 late days permitted
- › Exceptions can be granted on a case-by-case basis

Course Material



- Textbook: OS, 3 easy pieces:
<http://pages.cs.wisc.edu/~remzi/OSTEP/>
 - Its free!
 - Its excellent!
- Lecture notes (uploaded in eLearn)
- Published research papers

Grading Policy



- Lab Assignments ← 40%
 - Lab 1: 10%, Lab 2: 15%, Lab 3: 15%
- Reading and critiquing papers ← 10%
- Attendance & Class Participation ← 10%
- Mid-term (closed book) ← 10%
- Final (cumulative, closed book) ← 30%
 - A set of CEP questions included

Class Logistics



- › Class website:

- › <https://www.cs.ucr.edu/~heng/teaching/cs202-fall24/index.html>

- › Slack:

- › UCR slack workspace: cs_202_001_24F

- › Office Hours:

- › Monday 10:30-12, Wednesday 14:00-15:30

- › Submissions via eLearn

Why is this class challenging?



- From the instructor's perspective
 - The “comprehensive” way of teaching AOS yet in a QUARTER system!
 - Tier 1 course: need to consider students who have not taken an undergrad OS
- From students' perspective
 - Digesting the comprehensiveness in 9-10 weeks
 - Some (parts) of the lectures could be boring for those who have taken OS classes before

Cheating and Plagiarism



- Your code and critiques may be compared against each other

- Academic integrity policies and procedures
 - <https://conduct.ucr.edu/policies/academic-integrity-policies-and-procedures>

- What consequences can happen?
 - Will be reported to the Academic Integrity Committee (AIC)
 - Grade penalty
 - Course failure
 - Academic sanctions: probation, suspension, dismissal

Questions?



- › Visit my office during office hours
- › Email: heng@cs.ucr.edu
- › Slack

Situation



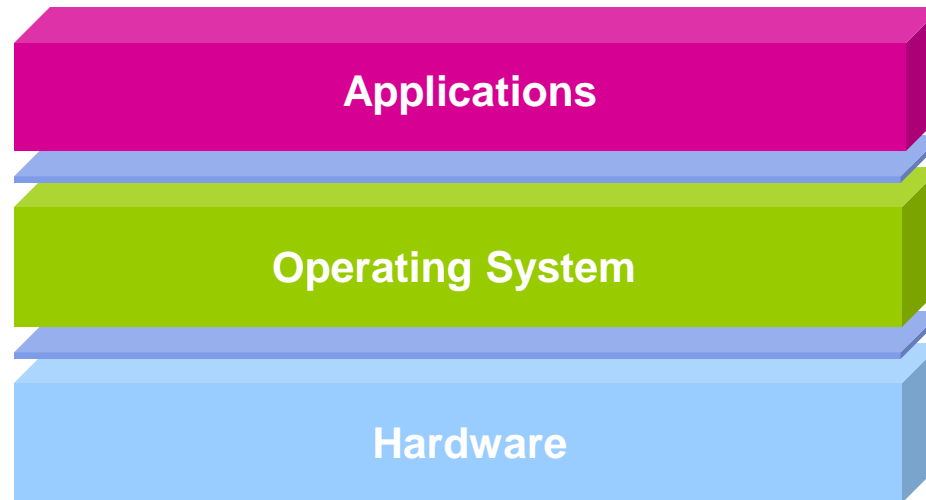
- We all have multiple applications running on our smart phone or computer
 - Written by programmers that don't know each other
 - They all just magically work – how??
- Goal today: get you ready to discuss OS structure, our first topic

Computing Systems – a hierarchy of abstractions



- Computing systems are a series of abstractions
 - Impossible to think about a system from electrons to application in one shot
 - What are some abstraction layers we have from transistors to applications?
- This class: OS level abstractions

What is an OS?



- › Directly has access to underlying hardware
- › Hides hardware complexity
 - › Offers nice abstractions to the applications through system calls
- › Manage the hardware on behalf of one or more applications
- › Ensures that applications are isolated and protected from each other

Getting more technical

- What is an OS?
 - A piece of software that *abstracts* and *arbitrates* a computing system
- A manager in a shop
 - Directs resources
 - Controls CPUs, memory, devices...
 - Enforces working policies
 - Fairness, resource limits, ...
 - Simplifies complex tasks
 - Abstracts hardware; offers system calls

Abstraction and Arbitration



- OS offers abstractions and arbitration
- Example of arbitration?
 - Allocate memory or CPU time
 - Arbitrate access to a shared device
- Examples of abstractions?
 - Files, sockets, process, thread, ...

Abstractions, mechanisms, policies

- › Memory management example
- › Abstraction: memory page
- › Mechanisms: allocate, map to a process, deallocate
- › Policies: page replacement, LRU, LFU, ...

Design principles



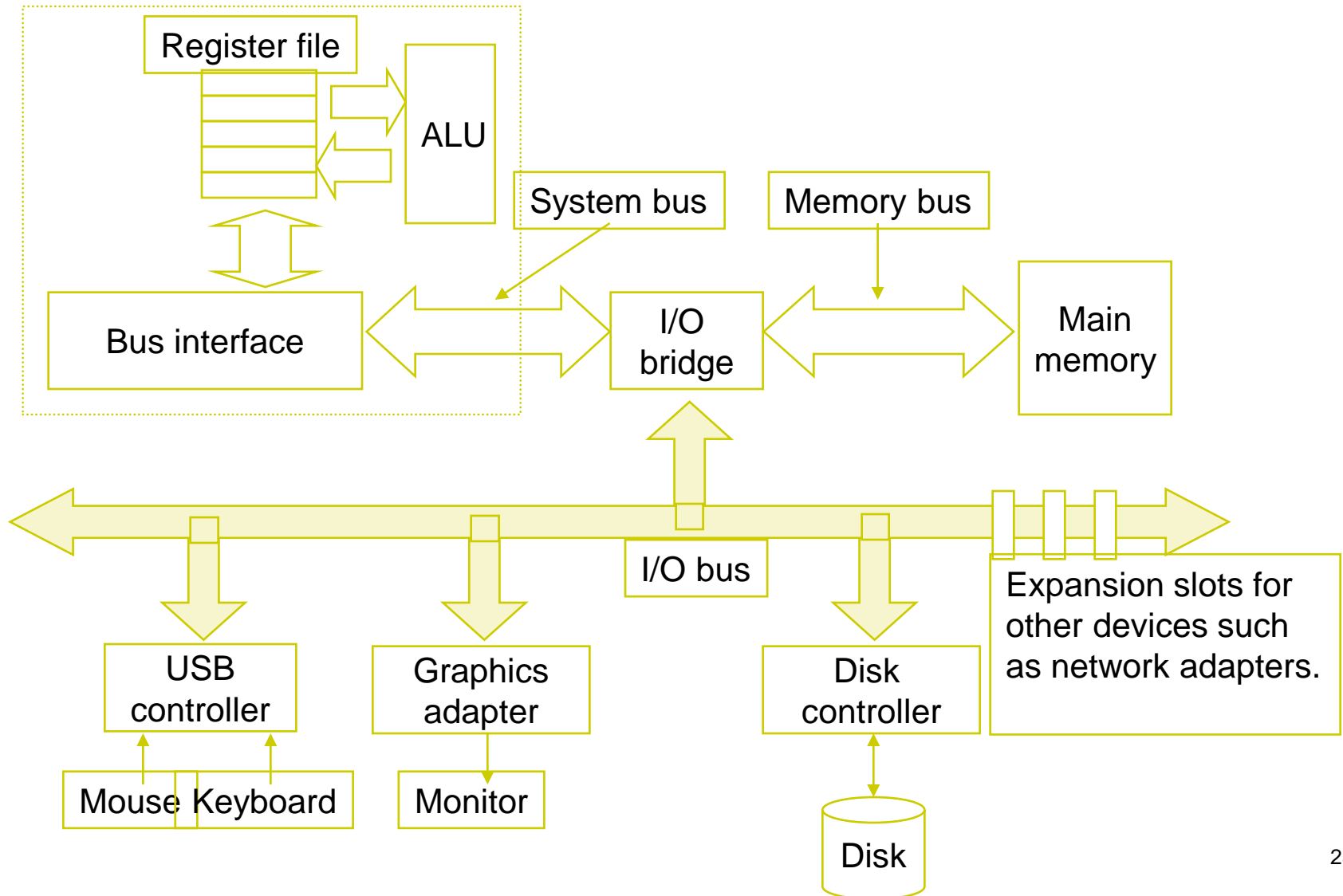
- Separation of mechanism and policy
 - Implement flexible mechanisms to support many policies
- Policies must optimize for the common case
 - Where will the OS be used?
 - What will the user want to execute?

Hardware and Resources



- › Good understanding of the hardware is essential to understanding OS
- › What hardware?
 - › Smart phone/tablets?
 - › Desktops?
 - › Servers?
 - › Computing clusters?
 - › Cloud?
- › How different are these?

They are not that different!



How does the OS interact with the hardware?



➤ OS

- Has protected access to hardware resources
- Arbitrates among competing requests
- Receives and handles events from the hardware

What support does the hardware provide to allow that?



- › **Manipulating privileged machine state**
 - › Protected instructions
 - › Manipulate device registers, TLB entries, etc.

- › **Generating and handling “events”**
 - › Interrupts, exceptions, system calls, etc.
 - › Respond to external events
 - › CPU requires software intervention to handle fault or trap

- › **Mechanisms to handle concurrency**
 - › Interrupts, atomic instructions

Catering to Applications



- › Provide resource needs of an application
 - › CPU, memory, device access
- › When applications launch, the OS loads the program from file into memory
 - › Allocates memory for code, data, heap and stack
 - › Can the application ask for more resources?
 - › Yes, it receives additional requests and provides resources as needed
- › OS also reacts to events in the system
- › Gets out of the way as fast as possible

CPU management

➤ Abstractions

- Program: static entity
- Process: program in execution
 - Unit of resource allocation and one thread of execution

Memory management



- ▶ Abstractions:
 - ▶ Address space for each processor
- ▶ OS implements these abstractions using the available hardware support
 - ▶ Paging, segmentation, TLBs, caches...

Storage/file system

- Abstraction: Files and directories
 - Others possible: e.g., object store
- Implemented in a variety of different ways
 - Traditional file system: mapping of files to storage
 - Network file system
 - Distributed FS
 - Internet scale FS

Conclusions



- Today was a quick overview of the role of an OS
- Goal is to get you ready to discuss OS organization and evolution, our first topic
 - First reading assignment out this evening.
- We did not discuss any implementation details
 - You should know from undergraduate OS
 - But please read on your own if you do not remember