Introduction
CS 111
Summer 2025
Operating System Principles
Peter Reiher

Outline

- Administrative materials
- Introduction to the course
  - Why study operating systems?
  - Basics of operating systems

Administrative Issues

- Instructor and TAs
- Load and prerequisites
- Web site, syllabus, reading, and lectures
- Exams, homework, projects
- Grading
- Academic honesty

Instructor: Peter Reiher

- UCLA Computer Science department faculty member
- Long history of research in operating systems
- Email: reiher@cs.ucla.edu
- Office: No in-person office hours this quarter
  - Office hours: By appointment, via Zoom
    - Zoom ID for office hours to be announced later

TAs

- Sara Khosravi
  - Email: srkhsrv21@gmail.com
- Office hours to be announced later

Instructor/TA Division of Responsibilities

- Instructor handles all lectures, readings, and tests
  - Ask me about issues related to these
- TA handles projects
  - Ask her about issues related to these
- Generally, instructor won't be involved with project issues
  - So direct those questions to the TA

Web Site

- We'll primarily use a web site set up for this class
  - Schedules for reading, lectures, exams, projects
  - Project materials
    - And uploads of completed projects
  - Copies of lecture slides and taped lectures
  - Also Zoom IDs for live lectures and office hours
  - Announcements
  - Sample midterm and final problems

Prerequisite Subject Knowledge

- CS 32 programming
  - Objects, data structures, queues, stacks, tables, trees
- CS 33 systems programming
  - Assembly language, registers, memory
  - Linkage conventions, stack frames, register saving
- CS 35L Software Construction Laboratory
  - Useful software tools for systems programming
- If you haven't taken these classes, expect to have a hard time in 111

Course Format

- Two weekly reading assignments
  - Mostly from the primary text
  - Some supplementary materials available on web
- Two weekly lectures
  - Posted as recorded lectures
  - No live/Zoom lecture sessions
- Four (10-25 hour) individual projects
  - Exploring and exploiting OS features
  - Plus one warm-up project
- A midterm and a final exam
  - Taken online

Course Load

- Reputation: THE hardest undergrad CS class
  – Fast pace through much non-trivial material
- Expectations you should have

– lectures	4-6 hours/week
– reading	3-6 hours/week
– projects	3-20 hours/week
– exam study	5-15 hours (twice)

- Keeping up (week by week) is critical
  – Catching up is extremely difficult

Primary Text for Course

- Remzi and Andrea Arpaci-Dusseau: *Operating Systems: Three Easy Pieces*
  - Freely available on line at http://pages.cs.wisc.edu/~remzi/OSTEP/
- Supplemented with web-based materials

Course Grading

- Basis for grading:
  - Class evaluation 1%
  - 1 midterm exam 20%
  - Final exam 26%
  - Lab 0 9%
  - Other labs 11% each

There are no extra credit options for this class. This will be the basis for grading.

- I do look at distribution for final grades
  - But don't use a formal curve

There is no "grade guarantee"

- All scores available on MyUCLA
  - Please check them for accuracy
  - Scores on BruinLearn are not authoritative

Midterm Examination

- When: 5th week (Friday, July 25)
  - Replacing that day's recitation section
  - Can be taken any time during that day
- Scope: <u>All</u> material up to the exam date
  - Approximately 60% lecture, 40% text
  - No questions on purely project materials
- Format:
  - On line, multiple choice, open book/notes
- Goals:
  - Test understanding of key concepts
  - Test ability to apply principles to practical problems

Final Exam

- When: Friday, August 29
  - Replacing final recitation section
  - Can be taken any time during that day
- Scope: Entire course
  - More emphasis on material covered after the midterm
- Format:
  - On line, multiple choice, open book/notes
- Goals:
  - Determining if you have mastered the full range of material presented in the class

Lab Projects

- Format:
  - 1 warm-up project
  - 4 regular projects
  - Done individually
- Goals:
  - Develop ability to exploit OS features
  - Develop programming/problem solving ability
  - Practice software project skills

Late Assignments & Make-ups

- Labs
  - Due dates set by the TA
  - ***NOTE: They may change from the dates listed on the syllabus***
  - TA also sets policy on late assignments
  - The TA will handle all issues related to labs
    - Ask her, not me
    - Don’t expect me to overrule their decisions
- Exams
  - Alternate times or make-ups only possible with prior consent of the instructor
  - If you miss a test, too bad

Academic Honesty

- It is OK to study with friends
  – Discussing problems helps you to understand them
- It is OK to do independent research on a subject
  – There are many excellent treatments out there
- But all work you submit must be your own
  – Do not <u>write</u> your lab answers with a friend
  – Do not <u>copy</u> another student's work
  – Do not turn in solutions <u>from off the web</u>
  – If you do research on a problem, <u>cite your sources</u>
- I decide when two assignments are too similar
  – And I forward them immediately to the Dean
- If you need help, ask the instructor

Academic Honesty – Projects

- Do your own projects
  - If you need additional help, ask your TA
- You must design and write <u>all</u> your own code
  - Do not ask others how they solved the problem
  - Do not copy solutions from the web, files or listings
- Protect yourself
  - Do not show other people your solutions
- It IS NOT OK to copy the answers from other people's old assignments
  - People who tried that have been caught and referred to the Office of the Dean of Students

Academic Honesty - Tests

- It shouldn't be necessary to say this, but . . .
- You must take the test yourself
- The complete rules for the tests will be stated before the test
- You must follow general UCLA academic honesty principles

Introduction to the Course

- Purpose of course and relationships to other courses
- Why study operating systems?
- What is an operating system?

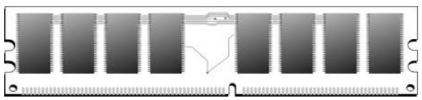
What Will CS 111 Do?

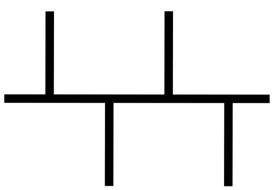
- Build on concepts from other courses
  - Data structures, programming languages, assembly language programming, computer architectures, ...
- Prepare you for advanced courses
  - Data bases, data mining, and distributed computing
  - Security, fault-tolerance, high availability
  - Network protocols, computer system modelling
- Provide you with foundation concepts
  - Processes, threads, virtual address space, files
  - Capabilities, synchronization, leases, deadlock

Why Study Operating Systems?

- Why do we have them, in the first place?
- Why are they important?
- What do they do for us?

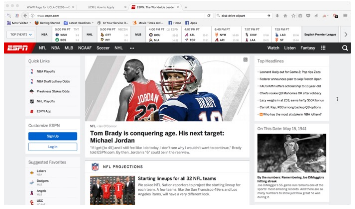
Starting From the Bottom

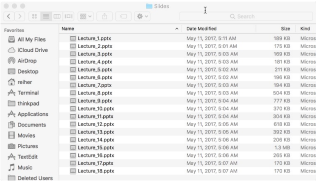


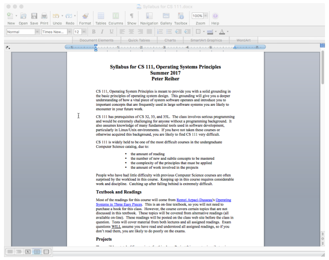


Here's what you've got

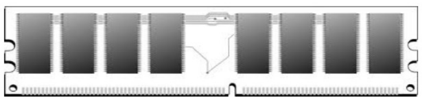
Here's what you want







What Can You Do With What You've Got?

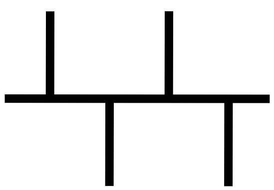


Read or write some binary words

MOV
ADD
JMP
SQRTPD

Report X and Y movements

Write to groups of pixels



READ
REQUEST SENSE

Read or write a block of data

And You Want This?

You're Going to Need Some Help

- And that's what the operating system is about
- Helping you perform complex operations
  - That interact
  - Using various hardware
  - And probably various bits of software
- While hiding the complexity
- And making sure nothing gets in the way of anything else

What Is An Operating System, Anyway?

- System software intended to provide support for higher level applications
  - Including higher level system software applications
  - But primarily for user processes
- The software that sits between the hardware and everything else
- The software that hides nasty details
  - Of hardware, software, and common tasks
- On a good day, the OS is your best computing friend

But Why Are You Studying Them?

- High probability none of you will ever write an operating system
  - Or even fix an operating system bug
- Not very many different operating systems are in use
  - So the number of developers for them is small
- So why should you care about them?

Everybody Has One

- Practically every computing device you will ever use has an operating system
  - Servers, laptops, desktop machines, tablets, smart phones, game consoles, set-top boxes
- Many things you don't think of as computers have CPUs inside
  - Usually with an operating system
  - Internet of Things devices
- So you <u>will</u> work with operating systems

How Do You Work With OSes?

- You configure them
- You use their features when you write programs
- You rely on *services* that they offer
  - Memory management
  - Persistent storage
  - Scheduling and synchronization
  - Interprocess communications
  - Security

Another Good Reason

- Many hard problems have been tackled in the context of operating systems
  - How to coordinate separate computations
  - How to manage shared resources
  - How to virtualize hardware and software
  - How to organize communications
  - How to protect your computing resources
- The operating system solutions are often applicable to programs and systems you write

One More Reason

- You are (mostly) computer scientists
- A computer scientist should have a reasonable understanding of how computers work
  - That’s why you take classes in architecture
- How operating systems work is also a major component of how computers work
- If you don’t understand an OS’ operations, you don’t understand computers

What Is An Operating System?

- Many possible definitions
- One is:
  - It is low level software . . .
  - That provides better, more usable abstractions of the hardware below it
  - To allow easy, safe, fair use and sharing of those resources

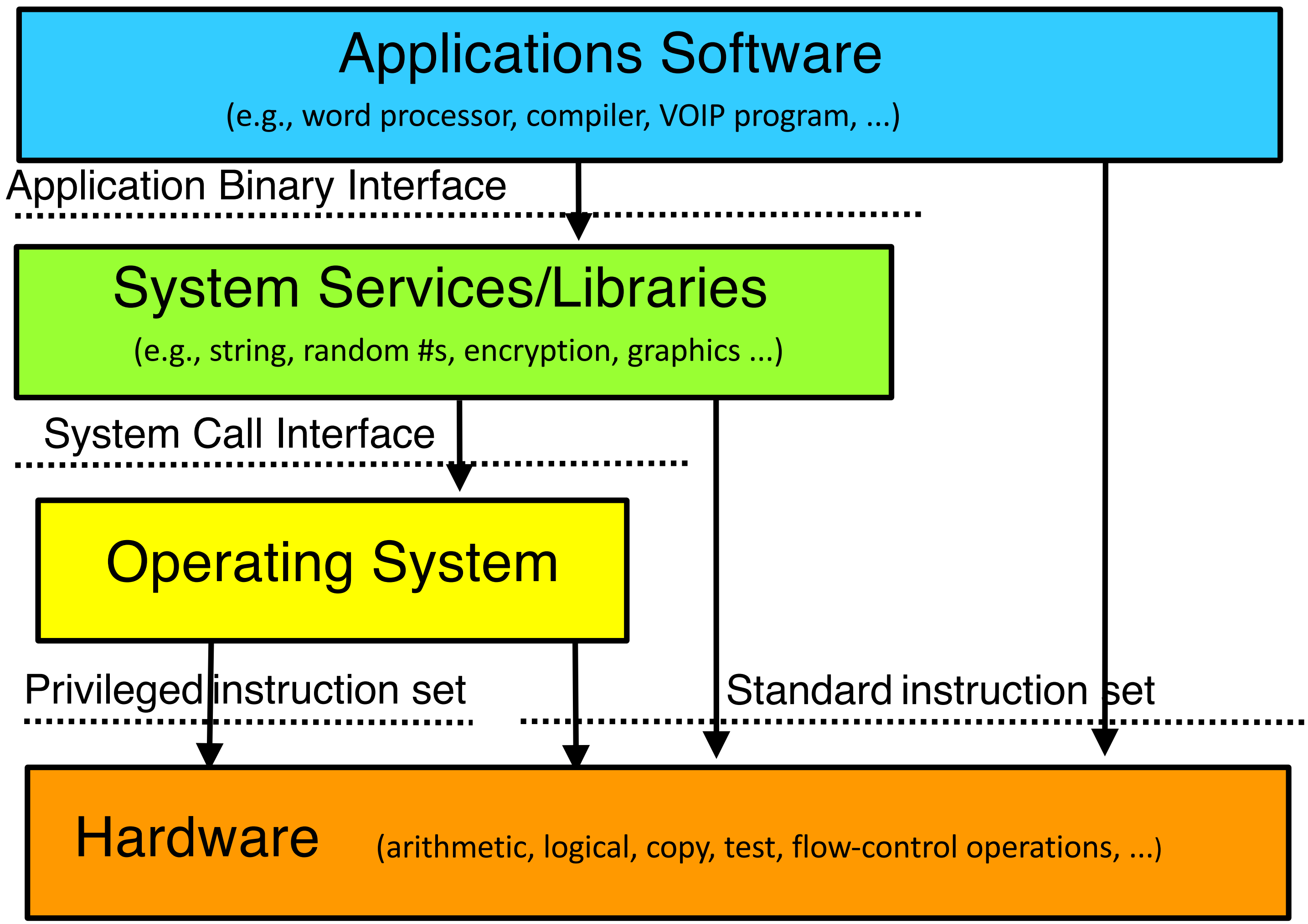
What Does an OS Do?

- It manages hardware for programs
  - Allocates hardware and manages its use
  - Enforces controlled sharing (and privacy)
  - Oversees execution and handles problems
- It abstracts the hardware
  - Makes it easier to use and improves SW portability
  - Optimizes performance
- It provides new abstractions for applications
  - Powerful features beyond the bare hardware

What Does An OS Look Like?

- A set of management & abstraction services
  - Invisible, they happen behind the scenes
- Applications see objects and their services
  - CPU supports data-types and operations
    - bytes, shorts, longs, floats, pointers, ...
    - add, subtract, copy, compare, indirection, ...
  - So does an operating system, but at a higher level
    - files, processes, threads, devices, ports, ...
    - create, destroy, read, write, signal, ...
- An OS extends a computer
  - Creating a much richer virtual computing platform
    - Supporting richer objects, more powerful operations

Where Does the OS Fit In?



What's Special About the OS?

- It is always in control of the hardware
  - Automatically loaded when the machine boots
  - First software to have access to hardware
  - Continues running while apps come & go
- It alone has <u>complete access</u> to hardware
  - Privileged instruction set, all of memory & I/O
- It mediates applications' access to hardware
  - Block, permit, or modify application requests
- It is trusted
  - To store and manage critical data
  - To always act in good faith
- If the OS crashes, it takes everything else with it
  - So it better not crash . . .

Instruction Set Architectures (ISAs)

- The set of instructions supported by a computer
  - Which bit patterns correspond to what operations
- There are many different ISAs (all incompatible)
  - Different word/bus widths (8, 16, 32, 64 bit)
  - Different features (low power, DSPs, floating point)
  - Competitive reasons (x86, PowerPC, Apple Silicon)
- They usually come in families
  - Newer models add features (e.g., Pentium vs. 386)
  - But remain upwards-compatible with older models
    - A program written for an ISA will run on any compliant CPU

Non-matching ISA means
your program can't run

Privileged vs. General Instructions

- Most modern ISAs divide the instruction set into *privileged* vs. *general*
- Any code running on the machine can execute general instructions
- Processor must be put into a special mode to execute privileged instructions
  - Usually only in that mode when the OS is running
  - Privileged instructions do things that are "dangerous"

Platforms

- ISA doesn't completely define a computer
  - Functionality beyond user mode instructions
    - Interrupt controllers, DMA controllers
    - Memory management unit, I/O busses
    - BIOS, configuration, diagnostic features
    - Multi-processor & interconnect support
  - I/O devices
    - Display, disk, network, serial device controllers
- These variations are called "platforms"
  - The platform on which the OS must run
  - There are lots of them

Portability to Multiple ISAs

- A successful OS will run on many ISAs
  - Some customers cannot choose their ISA
  - If you don't support it, you can't sell to them
- Which implies that the OS will abstract the ISA
- Minimal assumptions about specific HW
  - General frameworks are HW independent
    - File systems, protocols, processes, etc.
  - HW assumptions isolated to specific modules
    - Context switching, I/O, memory management
  - Careful use of types
    - Word length, sign extension, byte order, alignment
- How can an OS manufacturer distribute to all these different ISAs and platforms?

Binary Distribution Model

- Binary is a derivative of source
  - The OS is written in source
  - But only a binary distribution is ready to run
  - So a source distribution must be compiled
- OSes usually distributed in binary
- One (or more) binary distributions per ISA
- Binary model for platform support
  - Device drivers can be added, after-market
    - Can be written and distributed by 3^{rd} parties
    - Same driver works with many versions of OS

Binary Configuration Model

- Good to eliminate manual/static configuration
  - Enable one distribution to serve all users
  - Improve both ease of use and performance
- Automatic hardware discovery
  - Self-identifying busses
    - PCI, USB, PCMCIA, EISA, etc.
  - Automatically find and load required drivers
- Automatic resource allocation
  - Eliminate fixed sized resource pools
  - Dynamically (re)allocate resources on demand

What Functionality Is In the OS?

- As much as necessary, as little as possible
  - OS code is very expensive to develop and maintain
- Functionality must be in the OS if it ...
  - Requires the use of privileged instructions
  - Requires the manipulation of OS data structures
  - Must maintain security, trust, or resource integrity
- Functions should be in libraries if they ...
  - Are a service commonly needed by applications
  - Do not actually have to be implemented inside OS
- But there is also the performance excuse
  - Some things may be faster if done in the OS

Which OSes are Popular?

- Windows
  - The most popular choice for personal computers
  - Laptops, desktops, etc.
  - Some use in servers and small devices
- MacOS
  - Exclusively in Apple products
  - But in all Apple products (Macbooks, iPhones, Apple Watches, etc.)
- Linux
  - The choice in industrial servers (e.g., cloud computing)
  - And the choice of CS nerds and embedded systems

Conclusion

- Understanding operating systems is critical to understanding how computers work
- Operating systems interact directly with the hardware
- Operating systems rely on stable interfaces