CSE 120
Principles of Operating Systems

Spring 2020

Lecture 1: Course Introduction

Yiying Zhang
Lecture 1 Overview

• Class overview
• Administrative info
• Introduction to operating systems
Personnel

- Instructor
  - Yiying Zhang

- TAs and Tutors
  - Lihaol He (TA)
  - Xiao Liu (TA)
  - Jinmou Li (Tutor)
  - Priyal Rakesh Suneja (Tutor)

- Lectures: T/Th 5-6:20pm
- Discussion session: Wed 2-2:50pm
About Me (https://cseweb.ucsd.edu/~yiying/)

Vincent C. Rideout (MS 1940, Caltech)
-- Gerald Estrin (PhD 1951, University of Wisconsin)
---- David Martin (PhD 1966, University of California at Los Angeles)
------ David Patterson (PhD 1976, University of California at Los Angeles)
-------- Remzi and Andrea Arpaci-Dusseau (PhD 1999, University of California Berkeley)
--------- Yiying Zhang (PhD 2013, University of Wisconsin)

• Research interests:
  ◆ Operating systems
  ◆ Distributed systems
  ◆ Computer architecture
  ◆ Systems+networking, systems+security, systems+PL, systems+ML
  ◆ Undergraduate research opportunities at my lab (WukLab)!
CSE 120 Class Overview

• Course material taught through class lectures, textbook readings, and handouts
• Zoom lectures (link available on Canvas)
  ♦ Lectures will be recorded and available on Canvas
  ♦ No class attendance recorded, but strongly encourage all of you to attend Zoom live => actively thinking and asking questions right away is the best and easiest way to learn
• Lecture slides
  ♦ I will post slides in the morning of class day
• Course assignments are
  ♦ Homework questions
  ♦ Two large programming projects in groups
  ♦ Midterm and final exams
CSE 120 Class Overview

• Discussion sections
  ♦ TAs will go over projects and answer questions related to projects, homework, and course materials (mainly projects)

• Discussion forum
  ♦ Piazza

• My office hour hours **Wed/Fri 9:30-10:30am**
  ♦ Main forum for asking questions related to lecture materials
  ♦ Zoom links available on Canvas
  ♦ When you join the Zoom meeting, you will be in the waiting room. Post your questions through Zoom chat. I will answer them and accept students in the waiting room in FIFO order

• TA and tutor lab hours: more soon
  ♦ Main forum for asking questions related to projects
Textbook

Homeworks

• There will be 4 homeworks throughout the quarter
  ♦ Reinforce lecture material

• Homeworks provide practice learning the material
  ♦ You get full credit for a technical answer related to the homework question
  ♦ Amount learned from doing homework is proportional to effort
  ♦ Your choice on how much effort
Nachos Project

DOCTOR FUN

"This is the planet where nachos rule."
Nachos

• Nachos is an instructional operating system
  ♦ It is a user-level operating system and a machine simulator
    » Not unlike the Java runtime environment
    » Will become more clear very soon
  ♦ Programming environment will be Java on Unix (Linux)
  ♦ The projects will require serious time commitments
    » Waiting until the last minute is not a viable option!

• You will do two+ projects using Nachos
  ♦ Concurrency and synchronization
  ♦ System calls, processes, multiprogramming
  ♦ (Virtual memory used to be the third project, removed for COVID-19. We can release it for your own summer fun project.)

• You will work in groups of 1-3 on the projects
  ♦ Start thinking about partners
Labs

• You will need to use your home machine (laptop/desktop) to connect to ieng6
  ✤ You can also try to set up local environments and run locally (we will only provide limited support/help for that)
  ✤ Note: We will test and grade on ieng6 machines
  ✤ Be sure to test your projects there
    » You will be able to test before the deadline

• Lab hours: main forum to ask questions about projects
  ✤ Find time and Zoom link on course website
  ✤ We share lab hours (and all project materials) with Section-a
  ✤ TAs and tutors will use autograder to order questions
  ✤ They will not debug for you!
  ✤ Everyone gets only up to 10 min => need to serve a big class
Exams

• Midterm
  ♦ Tuesday April 30th (tentatively 5pm-7pm, reply Piazza poll for your availability by the end of this Friday!)
  ♦ Covers first half of class

• Final
  ♦ Thursday June 11th (7pm-9pm)
  ♦ Covers second half of class

• Both exams will be conducted online through Canvas
  ♦ Open book, you can use any resources
  ♦ Questions will be randomized, you have to finish the current one before you see the next one => no way for you to exchange answers with other students

• No makeup exams
  ♦ Everyone must be able to attend these exam dates
    » Unless absolute dire circumstances
Grading

- Homeworks: 6%
- Midterm: 21%
- Final: 23%
- Projects: 50%
How Not To Pass CSE 120

• Do not watch Zoom lecture videos
  ♦ Lecture videos are too long, the slides are online, and the material is in the book anyway
  ♦ Lecture material is the basis for exams and directly relates to the projects
  ♦ Batch many lectures and watch once, or worse, watch all lectures right before exams!!
  ♦ No class attendance recorded, but strongly encourage all of you to attend Zoom live => actively thinking and asking questions right away are the best and easiest way to learn

• Do not do the homework
  ♦ It’s only 6% of the grade, get full credit for turning anything in
  ♦ Concepts seem straightforward…until you apply them
  ♦ Excellent practice for the exams, and some homework problems are exercises for helping with the project
How Not To Pass Even More

• Violate academic integrity
  ♦ It is much better to get a 0 for an assignment than to fail the course for academic integrity violations
• Do not ask questions in lecture, office/lab hours, or online
  ♦ It’s scary, I don’t want to embarrass myself
  ♦ Asking questions is the best way to clarify lecture material at the time it is being presented
  ♦ 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
Project 1 Scores

SCORE

START DATE

DAY.0-2  DAY.2-4  DAY.4-6  DAY.6-8  DAY.8-10  DAY.10-12  DAY.12-14  DAY.14-16

mean  median
Class Web Page

http://cseweb.ucsd.edu/classes/sp20/cse120-b/

• Serves many roles…
  ♦ Course syllabus and schedule (updated over quarter)
  ♦ Lecture slides
  ♦ Homework handouts
  ♦ Project handouts

• Optional material and supplemental readings
  ♦ Entirely for your interest only
Academic Integrity

• Exams
  ✤ Work them on your own!

• Projects
  ✤ Each team must write their own solution
  ✤ No discussion of or sharing of specific code or written answers is allowed
  ✤ Any sources used outside of textbook/handouts/lectures must be explicitly acknowledged
  ✤ Your responsibility to protect your files from
    » e-copying using UNIX file protection
    » public access, including disposal

• We take cheating very seriously, with a zero tolerance policy
  ✤ We will run tools to catch that, do not even attempt!
Questions

• Before we start the material, any questions about the class structure, contents, etc.?
Why?

YOU HAVE A QUESTION, CALVIN?

YES! WHAT ASSURANCE DO I HAVE THAT THIS EDUCATION IS ADEQUATELY PREPARING ME FOR THE 21ST CENTURY?

AM I GETTING THE SKILLS I’LL NEED TO EFFECTIVELY COMPETE IN A TOUGH, GLOBAL ECONOMY? I WANT A HIGH-PAYING JOB WHEN I GET OUT OF HERE! I WANT OPPORTUNITY!
Why Operating Systems?

- Why are we making you sit here today, having to suffer through a core course in operating systems?
  - It’s not like everyone will become OS developers, after all
- Understand what you use
  - Understanding how an OS works helps you develop apps
  - OS is the foundation of virtualization, what cloud runs on
  - System functionality, performance, efficiency, etc.
- Pervasive abstractions
  - Concurrency: Threads and synchronization are common modern programming abstractions (Java, C#, C++, Go, etc.)
- Complex software systems
  - Many of you will go on to work on large software projects
  - OSes serve as examples of complex systems
This course addresses classic OS concepts

- Services provided by the OS
- OS implementation on modern hardware
- Interaction of hardware and software
- Techniques for implementing software systems that are
  - Large and complex
  - Long-lived and evolving
  - Concurrent
  - Performance-critical

System software tends to be mysterious

- Can your program allocate more memory than what your machine has physically? Why?

Our goal is to explain those mysteries
What this course is not about

- How to use an OS
- Graphic user interfaces of Oses
- A particular OS (although we use UNIX/Linux to explain many concepts)
- Different OS kernel architectures (take grad OS for that)
The system is running with 275 days uptime. The CPU usage is as follows: 0.1% in the user mode, 0.1% in the system mode, and 0% for both nice and idle modes. There are 171 tasks running, with 1 task currently running and 0 tasks stopped or zombie. The memory usage is as follows: 16467276k total, 141596k used, and 171168k buffers. The swap usage is 0k total and 0k used.

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>14677</td>
<td>voelker</td>
<td>20</td>
<td>0</td>
<td>55548</td>
<td>3232</td>
<td>2364</td>
<td>R</td>
<td>0.00</td>
<td>0.07</td>
<td>top</td>
</tr>
<tr>
<td>24637</td>
<td>voelker</td>
<td>20</td>
<td>0</td>
<td>86300</td>
<td>6364</td>
<td>1024</td>
<td>S</td>
<td>32:06</td>
<td>70</td>
<td>mosh-server</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>57812</td>
<td>1636</td>
<td>584</td>
<td>S</td>
<td>1:26</td>
<td>73</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.03</td>
<td>13</td>
<td>kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.04</td>
<td>38</td>
<td>migration/0</td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>9.54</td>
<td>94</td>
<td>ksoftirqd/0</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.00</td>
<td>1</td>
<td>watchdog/0</td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.04</td>
<td>39</td>
<td>migration/1</td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>11.22</td>
<td>89</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.00</td>
<td>1</td>
<td>watchdog/1</td>
</tr>
<tr>
<td>9</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.18</td>
<td>05</td>
<td>migration/2</td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>9.44</td>
<td>37</td>
<td>ksoftirqd/2</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.00</td>
<td>1</td>
<td>watchdog/2</td>
</tr>
<tr>
<td>12</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.18</td>
<td>06</td>
<td>migration/3</td>
</tr>
<tr>
<td>13</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>9.01</td>
<td>67</td>
<td>ksoftirqd/3</td>
</tr>
<tr>
<td>14</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.00</td>
<td>1</td>
<td>watchdog/3</td>
</tr>
<tr>
<td>15</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>2.30</td>
<td>99</td>
<td>events/0</td>
</tr>
</tbody>
</table>
What is an Operating System?

• How would you answer?
  ♦ (Yes, I know that’s why you’re taking the course…)
  ♦ (Note: There are many answers…)
A Typical Computer from a Hardware Point of View

CPU

Chipset

Memory

... I/O bus

CPU

Network
Computer System Components

User 1 → Compiler
User 2 → Assembler
User 3 → Text Editor
... → Database System

System and Application Programs

Computer Hardware

?
Computer System Components

- user 1
- user 2
- user 3
- ... user n

- compiler
- assembler
- text editor
- ...
- database system

system and application programs

- operating system
- computer hardware
“Code” that *sits between*:

- programs & hardware
- different programs
- different users

But what does it do/achieve?
What is an OS?

- Resource manager
- Extended (abstract) machine

Makes computers efficient, easy, and safe to use

- (will have a 3rd def based on pragmatics next time)
OS Manages Hardware Resources (answer 1)

• The OS controls/mediates/manages access to hardware resources
  ♦ Computation (CPUs)
  ♦ Volatile storage (memory) and persistent storage (disk, etc.)
  ♦ Communication (network, modem, etc.)
  ♦ Input/output devices (keyboard, display, printer, camera, etc.)
OS Manages Hardware Resources (answer 1)

- Allocation
- Reclamation
- Protection

Finite resources
Competing demands
OS Manages Hardware Resources (answer 1)

- Allocation
- Reclamation
- Protection

“The OS giveth
The OS taketh away”

Implied at termination
Involuntary at run time
Cooperative (yield cpu)
OS Manages Hardware Resources (answer 1)

- Allocation
- Reclamation
- Protection

“You can’t hurt me I can’t hurt you”

Implies some degree of safety & security
Abstraction for Applications (Answer 2)

- The OS defines a set of logical resources *(objects)* and a set of well-defined operations on those objects *(interfaces)*
  - Physical resources (CPU and memory)
  - Logical resources (files, programs, names)
  - Sounds like OO…

- The **logical, well-defined abstraction** OS provides is much more ideal than the hardware interface
  - Ease to use (no need to deal with low-level interface, hardware registers, different device models etc.)
  - Fair (well-behaved)
  - Supporting backward-compatibility
  - Reliable
  - Secure
Abstraction for Applications (Answer 2)

• Users and programs can safely coexist, cooperate, and share resources

• with the illusion of infinite, private (reliable, secure) resources
  ♦ Concurrent execution of multiple programs (timeslicing)
  ♦ Communication among multiple programs (pipes, cut & paste)
  ♦ Shared implementations of common facilities
    » No need to implement the file system more than once
  ♦ Mechanisms and policies to manage/share/protect resources
    » File permissions (mechanism) and groups (policies)
How to design an OS?
Is there a perfect OS?

- Efficiency
- Fairness
- Portability
- Interfaces
- Security
- Robustness

- Conflicting goals
  - Fairness vs efficiency
  - Efficiency vs portability
  - ...

- Furthermore, …
Hardware is evolving...

- 60’s-70’s - Mainframes
  - Rise of IBM

- 70’s - 80’s – Minicomputers
  - Rise of Digital Equipment

- 80’s - 90’s – PCs
  - Rise of Intel, Microsoft

- 90’s - 00’s – handheld/portable systems (laptops)

- 2007 - today -- mobile systems (smartphones), Internet of Things, specialized hardware in the cloud
  - Rise of iPhone, Android
## Implications on OS Design Goals: Historical Comparison

<table>
<thead>
<tr>
<th></th>
<th>Mainframe</th>
<th>Mini</th>
<th>Micro/Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System $/worker</strong></td>
<td>10:1 – 100:1</td>
<td>10:1 – 1:1</td>
<td>1:10-1:100</td>
</tr>
<tr>
<td><strong>Performance goal</strong></td>
<td>System utilization</td>
<td>Overall cost</td>
<td>Worker productivity</td>
</tr>
<tr>
<td><strong>Functionality goal</strong></td>
<td>Maximize utilization</td>
<td>Features</td>
<td>Ease of Use</td>
</tr>
</tbody>
</table>

CSE 120 – Lecture 1 – Course Intro
Hardware is evolving (cont) ...

- (once) New architectures
  - Multiprocessors
  - 32-bit vs. 64-bit
  - Multi-core

- New memory, storage, network devices
  - SSD, NVM, RDMA, SmartNIC

- New processors
  - GPU, TPU, FPGA
May You Live in Interesting Times...

- Processor speed doubles in 18 months
  - Number of cores per chip doubles in 24 months
  - But meeting its limit!
- Disk capacity doubles every 12 months
- Global bandwidth doubles every 6 months

→ Performance/cost “sweet spot” constantly decaying

* Does human productivity ever double?
Applications are also evolving...

- New applications
  - Computer games, networked games
  - Virtual reality
  - Web 2.0 (search, youtube, social network, …)
  - Video streaming
  - Mobile apps (> 2.8 million iPhone, Android apps)
  - Big data
  - Machine learning, deep learning, reinforcement learning
  - Autonomous vehicles
  - …
Implications to OS Design

• Constant evolution of hardware and applications continuously reshape
  ♦ OS design goals (performance vs. functionality)
  ♦ OS design performance/cost tradeoffs

• Any magic bullet to good OS design?
no magic in OS design

This is Engineering

• Imperfection
• Tradeoffs (perf/func/security)
• Different Goals
• Constraints
  ◆ hardware, cost, time, power
• Optimizations

Nothing’s Permanent

• High rate of change
  ◆ Hardware
  ◆ Applications
• Cost / benefit analyses
• One good news:
  ◆ Inertia of a few design principles
Separating Policies from Mechanisms

A fundamental design principle in Computer Science

**Mechanism** – tool/implementation to achieve some effect

**Policy** – decisions on what effect should be achieved

Example – CPU scheduling:
- All users treated equally
- All program instances treated equally
- Preferred users treated better

Separation leads to flexibility!
About this course...

Principles of OS design
• Some theory
• Some rational
• Lots of practice

Goals
• Understand OS design decisions
• Last piece of the “puzzle”
• Basis for future learning

To achieve the goals:
• Learn concepts in class
• Get hands “dirty” in labs
Topics we’ll cover

- Process management
- Memory management
- I/O management (file system)
- Intro to virtualization
- A touch of advanced topics if we have time
The fundamental issues/questions in this course are:

- **Structure**: how is an operating system organized?
- **Sharing**: how are resources shared among users?
- **Naming**: how are resources named (by users and programs)?
- **Protection**: how are users/programs protected from each other?
- **Reliability and fault tolerance**: how to mask failures?
- **Security**: how can information access/flow be restricted?
- **Communication**: how to exchange data?
Fundamental OS Issues (2)

- **Concurrency**: how to control parallel activities?
- **Performance**: how to make efficient use of resources, reduce OS overhead?
- **Scale and growth**: how to handle increased demand?
- **Compatibility**: can we ever do anything new?
- **Distribution**: how to coordinate remote operations?
- **Accountability**: how to charge for/restrict use of resources?

- And the **principles** in this course are the design methods, approaches, and solutions to these issues
Expect (some) pain

Somewhat fast pace

Lots of programming and debugging

Some difficult (abstract) concepts
For next class...

• Browse the course web
   [http://cseweb.ucsd.edu/classes/sp20/cse120-b/](http://cseweb.ucsd.edu/classes/sp20/cse120-b/)
• Sign up on Piazza!
• Read Chapters 1, 2, and 6
• Start thinking about partners for project groups
• Let the fun begin!
Brief History of OS design

In the beginning…
• OSes were runtime libraries
  ♦ The OS was just code you linked with your program and loaded into the computer
  ♦ First computer interface was switches and lights, then punched tape and cards
• Batch systems were next
  ♦ OS was permanently stored in primary memory
  ♦ It loaded a single job (card reader, mag tape) into memory
  ♦ Executed job, created output (line printer)
  ♦ Loaded the next job, repeat…
  ♦ Card readers, line printers were slow, and CPU was idle while they were being used
Spooling

- Disks provided a much faster I/O device than card readers, mag tape, and line printers
- Motivated development of spooling (Simultaneous Peripheral Operation On-Line)
  - Use disk to overlap I/O of one job with computation of others
  - Move program/data from card reader onto disk while another job computes
  - When done, next job ready to be loaded from disk
  - Can spool multiple programs onto disk, OS can choose which job to run next (job scheduling)
  - But, CPU still idle when job reads/writes to disk
Multiprogramming

- Multiprogramming increased system utilization
  - Keeps multiple runnable jobs loaded in memory
  - Overlaps I/O processing of a job with computation of another
  - Benefits from I/O devices that can operate asynchronously
  - Requires the use of interrupts and DMA
  - Optimizes system throughput (number of jobs finished in a given amount of time) at the cost of response time (time until a particular job finishes)
Timesharing

• Timesharing supports interactive use of computer by multiple users
  ♦ Terminals give the illusion that each user has own machine
  ♦ Optimizes response time (time to respond to an event like a keystroke) at the cost of throughput
  ♦ Based on timeslicing – dividing CPU time among the users
  ♦ Enabled new class of applications – interactive!
  ♦ Users now interact with viewers, editors, debuggers

• The MIT Multics system (mid-late 60s) was an early, aggressive timesharing system

• Unix and Windows are also timesharing systems…
Distributed Operating Systems

- Distributed systems facilitate use of geographically distributed resources
  - Machine connected by wires
- Supports communication between parts of a job or different jobs on different machines
  - Interprocess communication
- Sharing of distributed resources, hardware, and software
  - Exploit remote resources
- Enables parallelism, but speedup is not the goal
  - Goal is communication
Parallel Operating Systems

- Support parallel applications trying to get speedup of computationally complex tasks across multiple CPUs
- Requires basic primitives for dividing single task into multiple parallel activities
- Supports efficient communication among activities
- Supports synchronization of activities to coordinate data sharing
- Early parallel systems used dedicated networks and custom CPUs, now common to use networks of high-performance PC/workstations
Embedded Operating Systems

- Decreased cost of processing makes computers ubiquitous
  - Your car has dozens of computers in it
  - Think of everything that has electric motor in it, and now imagine that it also has a computer
- Each embedded application needs its own OS
  - Cell phones
  - PDAs (PalmPilot, etc.)
- Very soon
  - Your house will have 100s of embedded computers in it
  - Your electrical lines and airwaves will serve as the network
  - All devices will interact as a distributed system