Lecture 2: Historical perspective
Teaching Staff

- Abbas Mazloumi
  - Office hours
    - Available by appointment (email amazl001@ucr.edu)

- TA: Nathan Brennan
  - Office hours:
    - Wednesdays & Fridays from 3:30pm - 4:30pm
      - After the labs with a 10 minutes break
Grading breakdown

- projects (45% total)
  - Xv6 Operating system
  - Book uses examples from it
  - 3 projects
- Quizzes/homeworks (15% total)
  - 3 take-home quizzes (homeworks) each 5%
- Mid-term (20%)
- Final (really, second midterm) (20%)
- Engagement/extra credit (at least 2-4%)
  - Includes attendance in lab. and lecture
  - You learn much better if you are interested and engaged
Questions we started considering last time

- Why do we need operating systems course?
- Why do we need operating systems?
- What does an operating system need to do?
- Looking back, looking forward
Soap box – why you should care

- Student surveys show low interest coming in

- Computers are an amazing feat of engineering
  - Perhaps the greatest human achievement

- You get to understand how they work
  - OS, Architecture, Compilers, PL, … are the magic that makes computers possible

- Ours is a young field
  - Our Newtons, Einsteins, LaPlace’s, … happened in the last century
  - Many of our giants are still alive
  - So much innovation at an unbelievable pace
  - You can help write the next chapter
Why an OS class?

- Why are we making you sit here today, having to suffer through a course in operating systems?
  - After all, most of you will not become OS developers
- Understand what you use (and build!)
  - Understanding how an OS works helps you develop apps
  - System functionality, debugging, performance, security, etc.
- Learn some pervasive abstractions
  - Concurrency: Threads and synchronization are common modern programming abstractions (Java, .NET, etc.)
- Learn about 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
Questions we started considering last time

- Why do we need operating systems course?
- Why do we need operating systems?
- What does an operating system need to do?
- Looking back, looking forward
Why have an OS?

- What if applications ran directly on hardware?

- Problems:
  - Portability
  - Resource sharing
What is an OS?

- The operating system is the software layer between user applications and the hardware

- The OS is “all the code that you didn’t have to write” to implement your application
Questions we started considering last time

- Why do we need operating systems course?
- Why do we need operating systems?
- What does an operating system need to do?
- Looking back, looking forward.
Roles an OS plays

- **Beautician** that hides all the ugly low level details so that anyone can use a machine (e.g., smartphone!)
- **Wizard** that makes it appear to each program that it owns the machine and shares resources while making them seem better than they are
- **Referee** that arbitrates the available resources between the running programs efficiently, safely, fairly, and securely
  - Managing a million crazy things happening at the same time is part of that – **concurrency**
- **Elephant** that remembers all your data and makes it accessible to you -- persistence
More technically...

- **Abstraction**: defines a set of logical resources (objects) and well-defined operations on them (interfaces)

- **Virtualization**: Isolates and multiplexes physical resources via spatial and temporal sharing

- **Access Control**: who, when, how
  - Scheduling (when): efficiency and fairness
  - Permissions (how): security and privacy

- **Persistence**: how to keep and share data
Some Questions to Ponder

- What is part of an OS? What is not?
  - Is the windowing system part of an OS? Java? Apache server? Compiler? Firmware?

- Popular OS’s today include Windows, Linux, and OS X
  - How different/similar do you think these OSes are?

- Somewhat surprisingly, OSes change all of the time
  - Consider the series of releases of Windows, Linux, OS X…
  - **What are the drivers of OS change?**
  - What are the most compelling issues facing OSes today?
How many lines of code in an OS?

- Windows 10: 50M
- Vista (2006): 50M (XP + 10M)
- OS X (2006): 86M
- Linux: 25 million (grew 250K in 2018!)

What is largest kernel component?

- OSes are useful for learning about software complexity
  - The mythical man month
  - KDE (X11): 4M
  - Browser: 2M+, ...

If you become a developer, you will face complexity
  - Including lots of legacy code
Questions for today

- Why do we need operating systems course?
- Why do we need operating systems?
- What does an operating system need to do?
- Looking back, looking forward.
A brief history—Phase 0

- In the beginning, OS is just runtime libraries
  - A piece of code used/sharable by many programs
  - Abstraction: reuse magic to talk to physical devices
  - Avoid bugs

- User scheduled an exclusive time where they would use the machine

- User interface was switches and lights, eventually punched cards and tape
  - An interesting side effect: less bugs
Phase 1: Batch systems (1955-1970)

- Computers expensive; people cheap
  - Use computers efficiently – move people away from machine

- OS in this period became a program loader
  - Loads a job, runs it, then moves on to next
  - More efficient use of hardware but increasingly difficult to debug
    - Still less bugs 😊
 Advances in OS in this period

- SPOOLING/Multiprogramming
  - Simultaneous Peripheral Operation On-Line (SPOOL)
    - Non-blocking tasks
    - Copy document to printer buffer so printer can work while CPU moves on to something else
  - Hardware provided memory support (protection and relocation)
  - Scheduling: let short jobs run first
  - OS must manage interactions between concurrent things

- OS/360 from IBM first OS designed to run on a family of machines from small to large
Phase 1, problems

- Utilization is low (one job at a time)
- No protection between jobs
  - But one job at a time, so?
- Short jobs wait behind long jobs
- Coordinating concurrent activities
- People time is still being wasted
- Operating Systems didn’t really work
  - Birth of software engineering
Phase 2: 1970s – Time sharing, Unix, Persistence

- Computers and people are expensive
  - Help people be more productive

- Interactive time sharing: let many people use the same machine at the same time
  - CTSS/Multics projects at MIT
  - Corbato got Turing award for this idea

- Emergence of minicomputers
  - Terminals are cheap

- Persistence: Keep data online on fancy file systems
Unix appears

- Ken Thompson, who worked on MULTICS, wanted to use an old PDP-7 laying around in Bell labs
  - He and Dennis Richie built a system designed by programmers for programmers

- Originally in assembly. Rewritten in C
  - In their paper describing unix, they defend this decision!
  - However, this is a new and important advance: portable operating systems!

- Shared code with everyone (particularly universities)
  - Start of open source?
Unix (cont’d)

- Berkeley added support for virtual memory for the VAX
  - Unix BSD

- DARPA selected Unix as its networking platform in arpanet

- Unix became commercial
  - …which eventually lead Linus Torvald to develop Linux
Phase 3: 1980s -- PCs

- Computers are cheap, people expensive
  - Put a computer in each terminal
  - CP/M from DEC first personal computer OS (for 8080/85) processors
  - IBM needed software for their PCs, but CP/M was behind schedule
  - Approached Bill Gates to see if he can build one
  - Gates approached Seattle computer products, bought 86-DOS and created MS-DOS
  - Goal: finish quickly and run existing CP/M software
  - OS becomes subroutine library and command executive
Phase 4: Networked/distributed systems--1990s to now?

- Its all about connectivity
- Enables parallelism but performance is not goal
- Goal is communication/sharing
  - Requires high speed communication
  - We want to share data not hardware
- Networked applications drive everything
  - Web, email, messaging, social networks, …
New problems

- Large scale
  - Google file system, mapreduce, …
- Parallelism on the desktop (multicores)
- Heterogeneous systems, IoT
  - Real-time; energy efficiency
- Security and Privacy
Phase 5

- Computing evolving beyond networked systems
  - Cloud computing, IoT, Drones, Cyber-physical systems, computing everywhere
- Hardware accelerators, heterogeneous systems, end of Moore’s Law, Hardware democratization/Open source HW
- New workloads: AI, Blockchain, …
- New generation?
  - But what is it?
    - …and what problems will it bring?
Where are we headed next?

- How is the OS structured? Is it a special program? Or something else?
  - How do other programs interact with it?

- How does it protect the system?
  - What does the architecture/hardware need to do to support it?