Lecture 1: Introduction/Historical development

Instructor: Nael Abu-Ghazaleh
Slide contributions from
Chengyu Song, Harsha Madhyvasta and Zhiyun Qian
Teaching Staff

- Nael Abu-Ghazaleh
  - I am a Professor in CSE and ECE
  - Office hours Monday 1-2pm, Wednesday 2-3pm
    » Hope to meet many of you during office hours

- TA: Pavan Kumar Ananthula
  - MS student in Computer Science
    » TA’d CS153 with me before
  - Office hours TBA
  - Pavan is lead for Labs
Class Resources

- Check class webpage for information

- Lecture slides, homework, and projects will be posted on class webpage

- Assignment turn-in through Gradescope
  - Digital preferred, but if not please make sure legible

- Piazza for discussion forums; link on website
  - Stay on top of things – falling behind can snowball
Textbook

- Apraci-Dessau and Apraci-Dessau, OS, 3 easy pieces (required + free!)
  - Really well written book, rare in academic textbooks
  - Read! (especially if you can before class)

- Other pretty good books:
  - Anderson and Dahlin, *Operating Systems: Principles and Practice*
Class Mechanics Overview

- Grading breakdown
  - projects (40% total)
    » Xv6 Operating system
    » Book uses examples from it
    » 4 projects (used to be 2, splitting into halves)
      ■ To keep the TA load under control, they will grade each two together
  - 4 homeworks (20% total)
  - Exam 1 (20%)
  - Final (really, second exam) (20%)
  - Engagement额外 credit (2%+)
    » Includes attendance in lab. and lecture
    » Participation on piazza
    » You learn much better if you are interested and engaged
Class recordings

- Default policy: not available
  - From past experience, they harm (many) student learning and engagement
    » Attendance drops; students don’t end up watching videos

- Exceptions:
  - If you miss class due to an emergency, I can share them
  - If you attend classes regularly, and want to re-watch the lectures later, also let me know
    » Will check attendance through zoom logs
Projects

- Project framework: xv6
  - Projects are in C
  - Good debugging support
  - Used in OS class at several other universities

- Start to get familiar immediately
  - We will start labs. next week
  - Go over the xv6 documentation (on the course web page)
  - Optional Lab 0 to help get familiar with what xv6 is
Projects are difficult!

- Reputation as a hard class in the CS curriculum because of projects (IMO)
  - You *must learn gdb* if you want to preserve your sanity! 😊
  - Hopefully you won't think it's that hard by the time we are done!

- Working on the projects will take a lot of time

- Biggest reason the projects are hard: *legacy code*
  - You have to understand existing code before you add more code
  - Preparation for main challenge you will face at any real job
Project Recommendations

- Easier if you are engaged/excited
- Find a partner that you like/trust
- **Do not start working on projects at last minute!**
  - A lot of the time will be spend understanding the code
  - Debugging is integral process of development

- Make good use of help available
  - Post questions on piazza
  - Take advantage of TA office hours
  - Come prepared to Labs
  - Again, learning to debug
Project logistics

- Projects *can* be done in groups of two or individually
  - When you have chosen groups, send your group info to your TA
  - Use the find a partner feature in piazza
    - email if unable to find partner and we’ll try to connect
  - Option to switch partners after project two

- First step is to conceptually understand the project
  - Then come up with implementation plan
    - Fail and fail again
    - Debug, debug, debug (systems are unforgiving)
      - gdb is your friend
    - →success!!
Homeworks and Exams

- Four homeworks
  - Can expect similar questions on the exams

- Midterm (tentatively end of July)
  - In class

- Final
  - Covers second half of class
    » But you will need to know some basics from first half
    » Wont test you directly on first half

- Makeup exams
  - Only if there are documented serious circumstances
Submission Policies

- Homeworks due on ilearn by the end of the day (will be specified on ilearn)

- Code and design documents for projects (if applicable) due by the end of the day (similarly will be specified on ilearn)

- Late policy (also on course webpage):
  - 4 slack days across all deliverables
    - Will use the ilearn submission timestamp to determine the days
    - 2% bonus to HW and Labs if you don't use any of the slack days
  - 10% penalty for every late day beyond slack days
Recipe for success in CS153

- Start early on projects
- Attend labs and office hours
  - Take advantage of available help
- Be engaged, interested, curious
- Make sure to attend lectures
  - Going over slides is not the same
- Try to read textbook material before class
- Ask questions when something is unclear
  - 2%+ participation and extra credit – may bump up your grade if on borderline. Face recognition 😊
How Not To Pass CS 153

- Do not attend lecture
  - It’s too early! 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
  - I often see capable students hurt themselves badly (fail, or get miserable grades) by not attending

- Do not ask questions in lecture, office hours, or piazza
  - Asking questions is the best way to clarify lecture material at the time it is being presented
  - Office hours, piazza, and email will help with projects
How *Not* To Pass (2)

- 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

- Each quarter groups learn that starting early meant finishing all of the projects on time… and some do not
Objectives of this class

- In this course, we will study **problems** and **solutions** that go into design of an OS to address these issues
  - Focus on concepts rather than particular OS
  - Specific OS for examples

- Develop an understanding of how OS and hardware impacts application performance and reliability

- Examples:
  - What causes your code to crash when you access NULL?
  - What happens behind a printf()?
  - Why can multi-threaded code be slower than single-threaded 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
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 Euclids, Newtons, Darwins, … lived in the last half 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 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
What if applications ran directly on hardware?

Problems:
- Portability
- Resource sharing
We refer to the Operating System as "all the code that you didn’t have to write" to implement your application.
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.
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?
Pondering Cont’d

- 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 does this mean (for you)?
  - 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?

- It's 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?
For next class...

- Browse the course web (especially xv6 docs)
  http://www.cs.ucr.edu/~nael/cs153

- Read module 2 in textbook

- Start …
  - … tinkering with xv6
  - … attempting lab 0
  - … finding a partner for project group