Lecture 1: Course Introduction
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
    - Fourth year at UCR, but many more elsewhere
  - Office hours Tuesday 9-10am, Thursday 4:30-5:30, or by appointment
    - Hope to meet many of you during office hours

- Two TAs
  - Hadi Zamani (2nd time TA’ing cs153) and Ali Mohammadkhan
    - PhD students in Computer Science
  - Office hours TBA
  - Leads for Labs
Class Overview

- Check class webpage for information
  - Will send out link to webpage

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

- Assignment turn-in through iLearn
  - Digital only, no paper copy
  - Announcements through iLearn and posted on class webpage

- Piazza for discussion forums; link on website
  - Use these please
  - Stay on top of things – falling behind can snowball quickly into trouble
Textbook

● Apraci-Dessau and Apraci-Dessau, OS, 3 easy pieces (required + free!)

● Other good books:
  ◆ Anderson and Dahlin, *Operating Systems: Principles and Practice* (recommended)
Class 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)
  - Mid-term (16%)
  - Final (24%)
  - Engagement/extra credit (4%)
    » Includes attendance in lab. and lecture
    » You learn much better if you are interested and engaged
Projects

● Project framework this time: xv6
  ◆ Projects are in C
  ◆ Very good debugging support
  ◆ Used in OS class at several other universities

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

- Probably the hardest class you will take at UCR in terms of development effort
  - You must learn gdb if you want to preserve your sanity! 😊

- Working on the projects will take most of your time in this class

- Biggest reason the projects are hard: legacy code
  - You have to understand existing code before you can 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 to be done in groups of two
  ◆ 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 form groups
  ◆ 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)
    » → success!!
Homeworks and Exams

- Four homeworks
  - Can expect similar questions on the exams

- Midterm (tentatively Feb. 12)
  - In class

- Final (March 23, 8-10am)
  - Covers second half of class + selected material from first part
    » I will be explicit about the material covered
    » Because first midterm is short (50 minutes)

- No makeup exams
  - Unless dire circumstances
Submission Policies

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

- Code and design documents for projects 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 point 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
  ◆ 4% participation and extra credit – may bump up your grade if on borderline. Face recognition 😊
How *Not* To Pass CS 153

- Do not come to lecture
  - It’s nice out, 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 email
  - 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 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 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 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
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 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
The OS and Applications

- The OS defines a **logical, well-defined environment**…
  - Virtual machine (each program thinks it owns the computer)
- …for users and programs to **safely coexist, cooperate, share resources**

- **Benefits to applications**
  - Simpler (no tweaking device registers)
  - Device independent (all network cards look the same)
  - Portable (across Windows95/98/ME/NT/2000/XP/Vista/…)}
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?
- **Security**: how can information access/flow be restricted?
- **Communication**: how to exchange data?
- **Reliability and fault tolerance**: how to mask failures?
- **Extensibility**: how to add new features?
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