Introduction

Today
● Welcome to OS
● Administrivia
● OS overview and history

Next time
● Architectural support
Course overview ...

- Everything you need to know
  http://www.aqualab.cs.northwestern.edu/classes/eecs-343-f13/

- Course staff
  - Fabián Bustamante
  - John Rula and Zach Bischof (TAs)

- Overall structure
  - Lectures – Read the text before class
  - TA Sessions – Try them, a great way to bootstrap projects
  - Homework – Mainly a reading enforcer
  - Readings – Extra reading for extra points
  - Projects – Key to the course, learn-by-doing nature
  - Exams
Course overview ...

- Homework (4+1)
  - Part of their role is reading enforcers – textbook + papers

- Readings
  - Optional set of papers; outline in the webpage
  - Basis for extra-credit questions in exams

- Projects (4)
  - To do in groups of 2-3
  - *Must run in the VM image we will make available*

- Exams (2)
  - Final (not cumulative) on first day of final weeks (Dec. 11)

*Whatever you have heard, it’s not that bad!*
On reading

- Optional papers

- Mandatory
Your TODO list

- Homework
  - First one will be posted Thursday; due on Oct. 10
- Project
  - First one will be posted on Thursday, due Oct. 9
- Get and install the VM image
- Check you have access to T-Lab and/or Wilkinson Lab
  - Email help@eecs otherwise
- Start reading
- Start looking for partners for your project!

- And now a short quiz
FEATURE PRESENTATION
A computer system - Where's the OS?

- Hardware provides basic computing resources
- Applications define ways in which resources are used to solve users' problems
- **OS controls & coordinates use of hardware by users’ applications**
- A few vantage points
  - End user
  - Programmer
  - OS Designer

![Diagram of computer system components](image)

- compiler
- text editor
- Web browser

- systems and application programs
- operating system
- machine language
- microarchitecture
- physical devices
What is an operating system?

- Extended machine – top-down/user-view
  - The *illusionist*
  - Hiding the messy details, presenting a virtual machine that's easier to program than the hardware
    - *How much memory do you have?*
    - *How is your disk formatted?*
    - *How many processors do you have?*
What is an operating system?

- Resource manager – bottom-up/system-view
  - The referee
  - Everybody gets a fair-share of time/space from a resource (multiplexing in space/time)
  - A control program – to prevent errors & improper use (CP/M?)

CP/M Control Program/Monitor

Created for Intel 8080/85 by Gary Kildall of Digital Research, Inc. IBM tried to license it for their “new” IBM PC. Failing to get a non-disclosure agreement they used instead Microsoft to provide an OS.
What is an operating system?

- A bundle of helpful, commonly used things

Goals and evaluation criteria

- Reliability – the system does exactly what is designed to do
- Security and privacy – the system operation cannot be compromised by a malicious attacker and the data stored in the computer is only accessible to authorized users
- Convenience – make solving user problems easier
- Efficiency – use hardware in an efficient manner (machines demand efficient use)
- Easy to modify/evolve/port
What's part of the OS?

- Trickier than you think: file system, device drivers, shells, window systems, browser, ...
- Everything a vendor ships with your order?
- The one program running at all times, or running in kernel mode
  - Everything else is either a system program (ships with the OS) or an application program
  - Can the user change it?
- Why does it matter? In 1998 the US Department of Justice filed suit against MS claiming its OS was too big
Why having one?

- For applications
  - Programming simplicity
    - High-level abstractions instead of low-level hardware details
    - Abstractions are reusable across many programs
  - Portability (to != machines configurations/architectures)
Why having one?

- ... 

- For user
  - Safety
    - Program works within its own virtual machine
    - OS protects programs from each other
    - OS fairly multiplexes resources across programs
  - Efficiency (cost and speed)
    - Share one computer across many users
    - Concurrent execution of multiple programs
Why study operating systems?

- **Tangible reasons**
  - Build/modify one - OSs are everywhere
  - Administer and use them well
  - Tune your favorite application performance
  - Great capstone course
  - Common source for job interview questions

- **Intangible reasons**
  - Curiosity
  - Use/gain knowledge from other areas
  - Challenge of designing large, complex systems
Major OS issues

- Structure: how is the OS organized?
- Sharing: how are resources shared?
- Naming: how are resources named?
- Security: how is integrity of the OS and its resources ensured?
- Protection: how is one user/program protected from another?
- Performance: how do we make it all go fast?
- Reliability: what happens if something goes wrong?
- Extensibility: can we add new features?
- Communication: how do programs exchange information, including across a network?
Other OS issues

- Concurrency: how are parallel activities created and controlled?
- Scale and growth: what happens as demands or resources increase?
- Persistence: how do you make data last longer than program executions?
- Distribution: how do multiple computers interact with each other? how do we make distribution invisible?
- Accounting: how do we keep track of resource usage, and perhaps charge for it?

There are a huge number of engineering tradeoffs in dealing with these issues!
And now a short break …
Hardware/Software evolution/change

- 1960s Mainframes
- 1970s Minicomputers
- 1980s Microprocessors, workstations, LANs, Internet
- 1990s PC, Web
- 2000s Internet services, cloud computing, mobile/ubiquitous/embedded computing
- 2010s Sensor networks, data-intensive computing
- *It’s up to you*
The evolution of operating systems

- Early attempts – Babbage's (1702-1871)
  - Analytical Engine (Ada Lovelace – World's first programmer)

- 1945-55 – Vacuum tubes & plugboards ~ Open shop
  - ABC, MARK 1, ENIAC, later IBM 701 (1952)
  - No programming languages, no OS
  - A big problem
    - Scheduling – signup sheet on the wall
Evolution ... Batch systems (1955)

- Transistors → machs. reliable enough to sell
  - Separation of builders & programmers

- Getting your program to run
  - Write it in paper (maybe in FORTRAN)
  - Punch it on cards & drop cards in input room
  - Operator may have to mount/dismount tapes, setting up card decks, ... setup time!

- Batch systems ...
Evolution ... Batch systems (1955)

- **Batch systems**
  - Collect a tray of full jobs, read them all into tape with a cheap computer
  - Bring them to the main computer where the “OS” will go over each jobs one at a time
  - Print output offline

---

Evolution ... Multiprogramming (1965)

- To increase system utilization
  - Keeps multiple runnable jobs loaded in memory at once
  - Overlap I/O of a job with computing of another
  - Needs asynchronous I/O devices
    - Some way to know when devices are done
      - Interrupt or polling
  - Goal- optimize system throughput
    - Maybe at the cost of response time

- IBM OS/360 & the tar pit
Evolution … Spooling (1965)

- Disks much faster than card readers & printers
- Spool (Simultaneous Peripheral Operations On-Line)
  - While one job is executing, spool next one from card reader onto disk
    - Slow card reader I/O overlapped with CPU
  - With multiple programs onto disk, scheduling
  - But CPU still idle when program interact with a peripheral during execution
Evolution ... Timesharing (1961)

- To support interactive use
  - Multiple terminals into one machine
  - Each user given the illusion of owing the entire machine
  - Optimize response time maybe at the cost of throughput

- Time-slicing
  - Dividing CPU equally among users
  - If jobs are truly interactive, CPU can jump between them without users noticing it
  - Recovers interactivity for the user (why do you care?)

- CTSS (Compatible Time Sharing System), MULTICS (second-system effect) and UNIX
Evolution ... Parallel systems (1962)

- Some applications can be written as multiple tasks
  - Speed up when running on several CPUs
  - Need OS and language primitives for dividing/organizing the multiple tasks
  - Need OS primitives for fast communication between tasks
    • Degree of speedup dictated by communication/computation ratio
  - Many flavors of parallel computers today
    • SMPs (symmetric multi-processors, multi-core)
    • SMT (simultaneous multithreading [“hyperthreading”])
    • MPPs (massively parallel processors)
    • NOWs (networks of workstations) [clusters]
    • Computational grid (SETI @home)
Evolution ... PCs (197x)

- Large-scale integration >> small & cheap machines
- 1974 – Intel's 8080 & Gary Kildall's CP/M
- Early 1980s – IBM PC, BASIC, CP/M & MS-DOS
- User interfaces, XEROX Altos, MACs and Windows

![IBM PC circa 1981](image)
![Xerox Alto 1973](image)
Evolution ... Distributed and pervasive

- Facilitate use of geographically distributed resources
  - Workstations on a LAN or across the Internet
- Support communication between programs
- Speed up is not always the issue, but access to resources (including information)

Architectures
- Client/servers
  - Mail server, print server, web server
- Peer-to-peer
  - (Most) everybody is both, server and client
Evolution ... Embedded and pervasive

- **Pervasive computing**
  - Cheap processors embedded everywhere
  - How many are on your body now? in your car?
  - Cell phones, PDAs, games, iPod, network computers, ...

- **Typically very constrained hardware resources**
  - Slow processors
  - Small amount of memory
  - No disk or tiny disk
  - Typically only one dedicated application
  - Limited power

- **But technology changes fast**
“Ontogeny recapitulates phylogeny”*

The development of an embryo repeats the evolution of the species (*Ernst Haeckel)
Has it all been done then?

- New challenges constantly arise
  - Embedded computing
  - Sensor networks
  - Multiple cores
  - Cloud services
  - Peer-to-peer computing
  - Software defined networking

- Old problems redefine themselves
  - Evolution of smart phones recapitulated that of PC, which recapitulated that of minicomputers which …
  - But the ubiquity of PC’s redefined the issues of protections, phones are doing this again …
Summary

- In this class you will learn
  - Major components of an OS
  - How are they structured
  - The most important interfaces
  - Policies typically used in an OS
  - Algorithms used to implement those policies

- Philosophy
  - You may not ever build an OS, but
  - As a CS/CE you need to understand the foundations
  - Most importantly, OSs exemplify the sorts of engineering tradeoffs you'll need to make throughout your careers