CS 153
Design of Operating Systems

Fall 20

Lecture 2: Historical Perspective

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

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 an OS class?

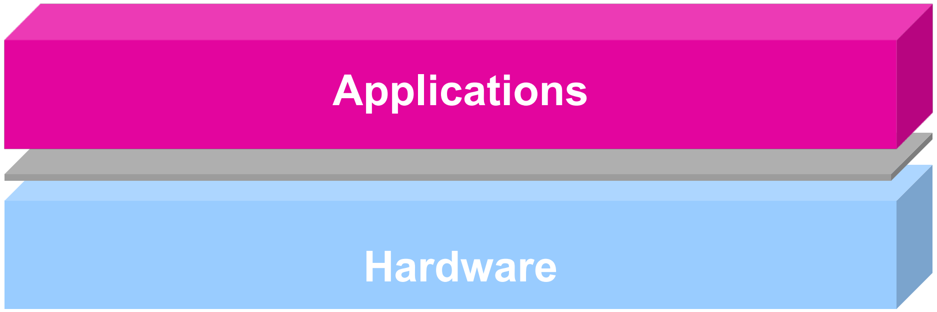
- Why are we making you sit here today, having to suffer through a course in operating systems?
  - After all, most of you will not become OS developers
- The concepts/problems are very general
  - We also encounter these problems in our daily life
  - Many abstractions like threads and synchronization are used pervasively in computer science
- Learn about **complex** software systems
  - Many of you will go on to work on large software projects
  - OS serve as examples of an evolution of complex systems
- Understand what you use (and build!)
  - Understanding how an OS works helps you develop apps
  - System functionality, debugging, performance, security, etc.

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?

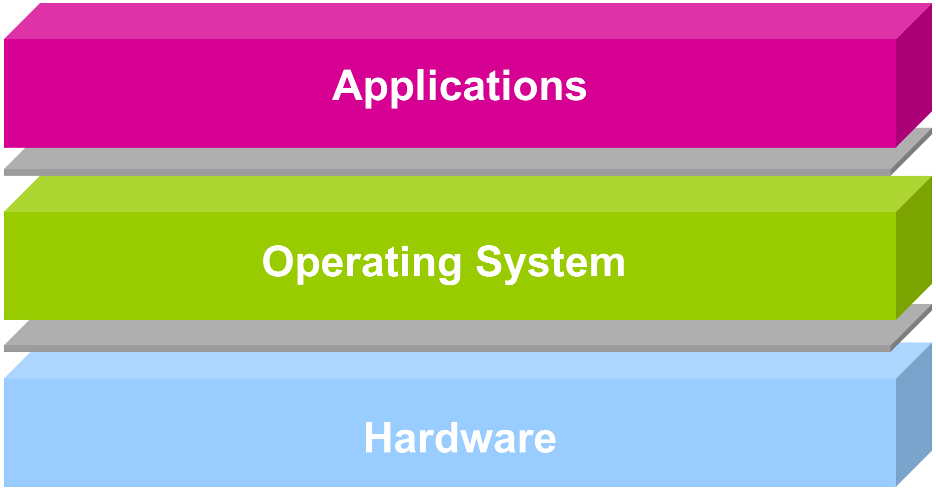
- What if applications ran directly on hardware?



- Problems:
  - Portability → OS Task 1: abstraction
  - Resource sharing → OS Task 2: multiplexing

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.

Fundamental Issues

- The fundamental issues/questions in this course are:
  - **Management**: how to allocate and schedule resources?
  - Performance: how to do better?
  - Protections: how to make sure things won't go wrong?
  - Security: how to create a safe environment?
  - Communication: how to enable collaboration?
  - Reliability and fault tolerance: how to mask failures?
  - Usability: how to enable the users/programs to do more?

Basic Roles of an OS

- **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

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

- **Persistence**: how to keep and share data

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.

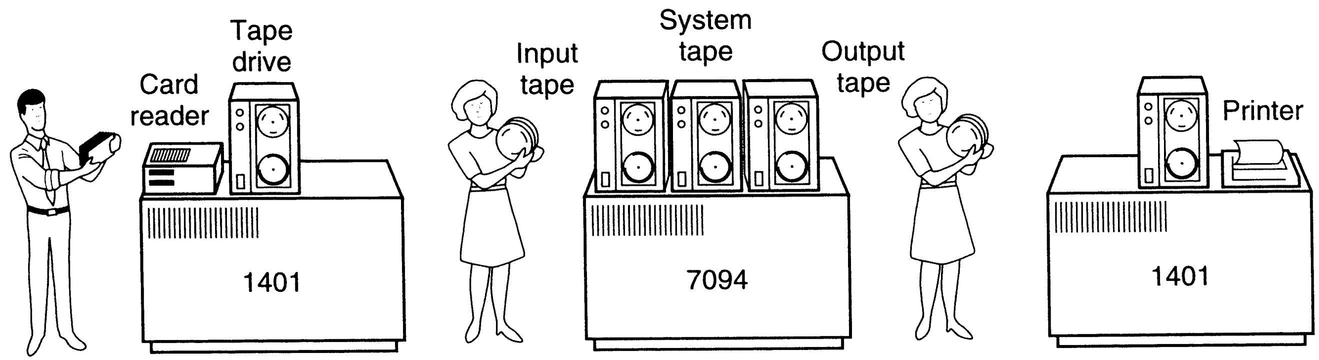
Phase 0

- In the beginning, OS is just runtime libraries (routines)
  - 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 tape and cards
  - An interesting side effect: less bugs

Phase 1: batch systems (1955-1970)



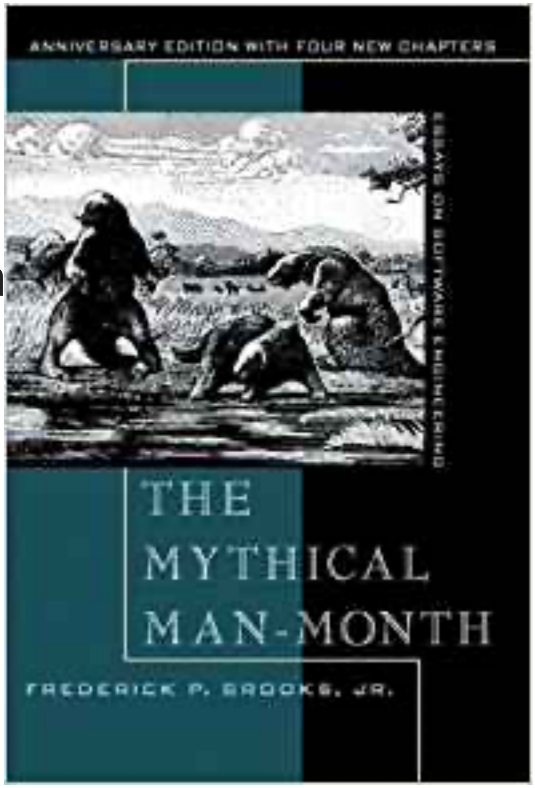
- Computers expensive; people cheap
  - Use computers efficiently – move people away from machine
- OS in this period became a program loader
  - Loads a job, runs it, outputs result, then moves on to next
  - More efficient use of hardware but increasingly difficult to debug
    - » Still less bugs ☺

Advances in OS in this period

- SPOOLING/Multiprogramming
  - Simultaneous Peripheral Operations 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
  - 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 what can go wron
- Scheduling
- Coordinating concurrent activities
- People time is still being wasted
- Operating Systems didn't really work
  - The mythical man month
  - Birth of software engineering



Phase 2: 1970s

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

- Interactive time-sharing: let many people use the same machine at the same time

- 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)

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

- 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/power consumption/...
  - Requires high speed communication
  - We want to share data not hardware
- Networked applications drive everything
  - Web, email, messaging, social networks, …
  - Chromebook

New problems

- Large scale
  - Google file system, map-reduce, …
- Parallelism on the desktop (multicores)
- Heterogeneous systems, IoT
  - GPU, FPGA, …
  - Real-time; energy efficiency
- Security and Privacy

Phase 5

- New generation?
- Computing evolving beyond networked systems
  - Cloud computing, edge computing, IoT, wearable devices, drones, cyber-physical systems, autonomous cars, computing everywhere
  - 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?

Why start with architecture?

- Recall: Key roles of an OS are
  1) Wizard: isolation and resource virtualization
  2) Referee: efficiency, fairness, and security

- Architectural support can greatly simplify – or complicate – OS tasks
  - Easier for OS to implement a feature if supported by hardware
  - OS needs to implement everything hardware doesn‘t

- OS evolution accompanies architecture evolution
  - New software requirements motivate new hardware
  - New hardware features enable new software

For next class...

- Continue to get familiar with xv6
  - Chapter 0
  - Appendix A and B