Lecture 2: Historical perspective
Instructor: Nael Abu-Ghazaleh
Last time

- What is an OS?
- What roles does it play?
- Today: Historic evolution of Operating Systems (and computing!)
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?
  ◆ Vista (2006): 50M (XP + 10M)
    » What is largest kernel component?
  ◆ OS X (2006): 86M

● 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 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, 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

- 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?

- 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

● New generation?

● Computing evolving beyond networked systems
  ● Cloud computing, IoT, Drones, Cyber-physical systems, 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
Some questions to get you thinking

- What is the OS? Software?

- Is the OS always executing?
  - If not, how do we make sure it gets to run?

- How do we prevent user programs from directly manipulating hardware?
Sleeping Beauty Model

- Answer: Sleeping beauty model
  - Technically known as Controlled direct execution
  - OS runs in response to “events”; we support the switch in hardware

- Most of the time the OS is sleeping
  - Good! Less overhead
  - Good! Applications are running directly on the hardware