Slide 1

Administrivia

- Note that these slides are linked from the schedule page of the course Web site. Should be available before class, and updated after class if need be.
- A little attempt at social connection: I encourage you to turn on video so we can see each other, but if you'd rather not, no worries.

  Also as a way of connecting just a little, how about if everyone sends a short chat message with your name and where you are? related in some way to the course or computing).

Slide 2

Administrivia

- About office hours: Tentative times M/W 1:30pm–2:20pm, 5:15pm–6:15pm, sometime F after 1:30pm. I'll set up Zoom meetings and provide information via my home page and the course Web site.
- About minute essays: In addition to answering whatever questions I ask, feel free to ask me, well, anything (preferably related in some way to the course or computing).
- About the reading: What may work well is to wait until after class, and then focus on things mentioned in class while still at least skimming other material (might be something that would interest you!).

Slide 3

Minute Essay From Last Lecture

- Many people did interesting things over the summer! lots of internships.

Slide 4

What Is An Operating System? (Review)

- Definition by example:
  - Recent: Windows, Linux, UNIX, OS X (Mac), iOS, Android . . .
  - Older: MULTICS, VMS, MVS, VM/370, . . .
  - (Also special-purpose O/S's for special-purpose hardware.)
- Definition(s) from operating systems textbook:
  - Something that provides "virtual machine" for application programs and users ("top down").
  - Something that manages computer's resources ("bottom up").
- Another view — key part of bridging gap between what hardware can do (not much, but very fast) and what users want.

Slide 5

What The Hardware Can Do

- CPU: fetch machine instruction from memory; execute; repeat.
- Disk: read data from / write data to location on disk.
- And so forth — very primitive.

Slide 6

What The Software Must Do

- Programs students usually write in CS1, CS2:
  - Define and manipulate data structures.
  - Do arithmetic/logical calculations.
  - Read stdin / write stdout.
  - Call GUI/graphics library routines.
- The magic cloud (operating system):
  - Read from keyboard, write to screen.
  - Manage what's on screen — windows, taskbar, etc.
  - Run multiple applications "at the same time".
  - Manage disk contents — files, directories/folders.
  - Share the machine with other users.

Slide 7

Why Review History?

- To understand roots/development of current operating systems.
- As a way of getting many perspectives on "what do we want an O/S to do, and how do we make it do that?"
- Because history is intrinsically interesting? Try to imagine what using some of those early machines might have been like.
- (To allow the instructor to relive the days of his/her/their youth?)

Slide 8

The Early Days (1940s)

- Programming done by making physical connections on a plugboard (!).
- Better than no computer at all, but tedious and inefficient!
- Example: the ENIAC (picture via "links" page).

Slide 9

The Early Days (1940s – 1950s)

- Key improvements: stored-program concept, punch cards.
- Programming done by encoding machine language into cards.
- Program included code to start up computer, read rest of program into memory, do all input and output, etc. — no operating system.
- One program at a time, machine operated by programmer.
- Better, but still tedious and inefficient!

Slide 10

The Early Days (1950s)

- Key improvements: assemblers and compilers, libraries of commonly-used code, specialists to run machine (operators).
- Programming done in assembly language (or early high-level language), punched into cards.
- Separate steps to translate to machine language, execute.
- One program at a time, but machine operated by specialist.
- Less tedious, less inefficient.
- Still cumbersome for programmers, CPU idle between steps.

Slide 11

Batch Systems (1950s)

- Key improvement: "batch" idea — automate transitions between steps (translate program, execute, translate next program, etc.).
- How to make this work? separate input by "control cards", write primitive operating system to interpret them, manage transitions.
- Less inefficient, but I/O devices slow, so CPU idle a lot — still one program at a time.
- Still cumbersome for programmers — punch program into cards, give to operator, wait for output.

Slide 12

Sidebar: Punch Cards

- (If we were in the classroom, I'd circulate a stack representing a programming assignment for a long-ago class I took! But we're not, alas.)
- Stiff paper, each card logicallya arranged into columns, rows. One card per line of source code, with columns representing characters. Within a column, rows used to encode character.
- Special keypunch device let you type and have the result encoded into cards. No backspace though! Editing? Add/remove/replace whole cards. (Sound tedious? It was!)
- Other specialized devices sometimes used to manipulate cards — e.g., "sorter" that would sort on a column.

Slide 13

Control Cards — Example

Sketch of control cards for IBM mainframe O/S to compile and run:

```
//jobname  JOB  acctno,name, ....
//stepname EXEC PGM=compiler_name,PARM=(options)
//STEPLIB  DD   DSNAME=path_for_compiler
//SYSUT1   DD   UNIT=SYSDA,SPACE=(parameters)
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=object_code,UNIT=SYSDA,
//              DISP=(MOD,PASS),SPACE=(parameters)
//SYSIN    DD   *
source code
/*
//stepname EXEC PGM=load-and-go
....
```

Slide 14

Multiprogramming Systems (1960s – ?)

- Key improvement: “multiprogramming” — more than one program in memory, so when one has to wait another can run.
- How to make this work? requires much more complex operating system — must share memory and I/O devices among programs, switch between them, etc.
- Efficient use of hardware.
- Still cumbersome for programmers — no real changes here.
- Example: IBM mainframe and peripherals (pictures on “links” page).

Slide 15

Timesharing Systems (1960s – ?)

- Key improvements: "interactive" users (using text terminals), utility programs to support them (shells, text editors, etc.).
- How to make this work? like multiprogramming, but now programs sharing memory are interactive users wanting fast response.
- Efficient use of hardware.
- Much less cumbersome for program development!
- Example: IBM terminal (picture on "links" page).

Slide 16

Personal Computers (1980s – ?)

- Similar evolution of operating systems — initially very simple, gradually becoming more complex/capable.
- Features from mainframes adopted as hardware permitted.
- A key difference — emphasis on user convenience rather than efficient use of hardware.

Slide 17

Evolution of Operating Systems, Recap

- Increasing hardware capability.
- Increasing O/S functionality and complexity — from simple program loader to complex multitasking system.
- Parallels between evolution of mainframe O/S and PC O/S. (Similar evolution may be happening with operating systems for smartphones?)

Slide 18

Minute Essay

- Do you have a copy of the textbook? (I hope everyone will soon.) Paper or electronic? (I'm curious!)
- What's the most primitive and/or cumbersome system you've personally used? (I mean system-as-a-whole here, not specific tools — a PC without a GUI, say, or a mainframe of some sort.)