Slide 1

Administrivia

- These presentations will be linked from the schedule page. Usually I'll post a preliminary version before class, a more-final version later (sometimes there are typo fixes, e.g.).
- I've also made a Google Doc with links to Zoom recordings and shared it with all of you. There's a link to it at the bottom of the schedule page too.
- Also, you may notice in your Google Drive "shared with me" a folder with name "CSCI-3323-*your_name*"? This is the shared-only-by-you-and-me folder where I plan to put information about grades and where I want you to put things you're turning in.
- I'm hoping to get the first reading quiz (over chapter 1 and 2) posted over the long weekend.

Slide 2

Introduction

- The textbook's first two chapters are — I think — such a good, and readable, introduction to what we will talk about that I doubt I can improve on them much, so I won't try.
- Yes, the authors are opinionated, and yes, they're a bit Linux-centric, but then so am I, and it does makes for livelier reading (well, okay, maybe not if you're a fan of some other operating system).
- A few things worth noting . . .

Slide 3

Textbook Versus(?) Topics from Syllabus

- Role and purpose of operating systems — introduction.
- History of operating systems — introduction (in a bit more depth shortly).
- Processes and process management, including a discussion of concurrency and related issues — virtualization (of CPU), concurrency.
- Memory management — virtualization (of memory).
- Input/output and device management — persistence.
- File systems — persistence.
- Operating system security — everywhere?
- Other topics as time permits (very likely some programming with Linux system calls) — everywhere?

Slide 4

Another Definition of Role and Purpose of O/S

- (From the textbook I had been using.)
- Top-down view: Provide a “virtual machine” easier to use than the real one. Key abstractions are processes, files.
- Bottom-up view: Manage physical resources on behalf of multiple applications, possibly multiple users.

Slide 5

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 author and/or instructor to relive the days of his/her youth? Not so much with the authors of this year's book!)

Slide 6

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 7

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 8

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 9

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 10

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 11

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 12

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 13

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 14

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 “smart phones”?)

Slide 15

Minute Essay

- Any questions so far?