Slide 1

## Administrivia

- (None.)

Slide 2

## 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 youth?)

Slide 3

## 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 here).

Slide 4

## 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 5

## 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 6

## 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 7

## Control Cards — Example

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

Slide 8

## 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 (picture here, 1964); keypunch machine (picture here); line printer (picture here).

Slide 9

## 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 terminals (picture here, late 1970s).

Slide 10

## 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 11**

## 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.

**Slide 12**

## Minute Essay

- What's the most primitive and/or cumbersome system you've personally used to write programs?