Administrivia

- 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).

Slide 1

Minute Essay From Last Lecture

- Many people did interesting things over the summer! lots of internships. (For those of you who did an internship — please consider Dr. Lewis's request to speak briefly at the colloquium.)

Slide 2

What Is An Operating System? (Review)

- Definition by example:
  - Recent: Windows, Linux, UNIX, iOS, OS X (Mac), . . .
  - Older: MULTICS, VMS, MVS, VM/370, . . .
  - (Also special-purpose O/S's for special-purpose hardware.)

Slide 3

- 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.

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 4

What The Software Must Do

Slide 5

- 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.

Why Review History?

Slide 6

- 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?)

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.

Slide 8

- One program at a time, machine operated by programmer.

- Better, but still tedious and inefficient!

Slide 9

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 10

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 11

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 12

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).

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.

Slide 13

- Efficient use of hardware.

- Much less cumbersome for program development!

- Example: IBM terminal (picture on "links" page).

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

Slide 14

  hardware.

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

- 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.)

Slide 16