Slide 1

Administrivia

- Reminder: Homework 1 due Friday. Turn in non-programming problems in hard copy, in class or in my mailbox in HAS 200 by 5pm. Turn in programming problem by e-mail by 11:59pm.
- I say in the syllabus that I try to respond promptly to e-mail. Exceptions are minute essays and homeworks, which I don't always look at right away. If you need a quick reply, make that apparent on the subject line please!
- Notes on different versions of GNU compilers linked from my home page.
- Quiz 1 next Wednesday. Short question(s) based on chapter 1, lectures. "Open book/notes", meaning access to textbook, your notes, anything on course Web site.

Slide 2

Sidebar: C/C++ Programming Advice

- I strongly recommend always compiling with flags to get extra warnings. There are lots of them, but you can get a lot of mileage just from `-Wall`. Add `-pedantic` to flag nonstandard usage.

  Warnings are often a sign that something is wrong. Sometimes the problem is a missing `#include`. `man` pages tell you if you need one.
- If you want to write "new" C (including C++-style comments), add `-std=c99`.
- If typing all of these gets tedious, consider using a simple makefile. Create a file called `Makefile` containing the following (the first line for C, the second for C++):

  ```
  CFLAGS = -Wall ....
  CXXFLAGS = -Wall ....
  ```

  and then compile `hello.c` to `hello` by typing `make hello`, or

Slide 3

similarly for `hello.cpp`.

Slide 4

Overview of Hardware — Recap

- Idea is to get a sense of what o/s designers/developers have to work with.
- Notice also what features seem intended to make it possible to write an o/s that can defend itself!
- (I won't talk in class about the sections on buses and booting, but do read them.)

Slide 5

Operating System Functionality — Recap

- "Operating system as virtual machine" must provide key abstractions (processes, filesystems).
- "Operating system as resource manager" must manage resources (memory, I/O devices, etc.).
- Operating system functionality typically packaged as "system calls".
- Details obviously vary among systems, but some ideas are common to most/many . . .

Slide 6

Processes — Abstraction

- Basic idea — a program (application or background activity) together with its current state (registers and memory contents).
- In order to have more than one at a time, need some way to share the physical machine among them.
- May be useful to think in terms of each process having its own simulated processor and memory ("address space"), with operating system providing infrastructure to map that onto the hardware. How to do that? (Next slide.)
- Other relevant concepts include process ownership, hierarchical relationships among processes, interprocess communication.
- Relevant system calls — create process, end process, communicate with another process, etc.

Slide 7

Processes — Implementation

- Managing the “simulated processor” aspect requires some way to timeshare physical processor(s). Typically do that by defining a per-process data structure that can save information about process. Collection of these is a “process table”, and each one is a “process table entry”.
- Managing the “address space” aspect requires some way to partition physical memory among processes. To get a system that can defend itself (and keep applications from stepping on each other), memory protection is needed — probably via hardware assist. Some notion of address translation may also be useful, as may a mechanism for using RAM as a cache for the most active parts of address space, with other parts kept on disk.

Slide 8

Filesystems

- Most common systems are hierarchical, with notions of “files” and “folders”/“directories” forming a tree. “Links”/“shortcuts” give the potential for a more general (non-tree) graph.
- Connecting application programs with files — notions of “opening” a file (yielding a data structure programs can use, usually by way of library functions).
- Many, many associated concepts — ownership, permissions, access methods (simple sequence of bytes, or something more complex?), whether/how to include direct access to I/O devices in the scheme.
- Relevant system calls — create file, create directory, remove file, open, close, etc., etc.
- See text for some UNIXisms — single hierarchy, regular versus special files, pipes, etc.

Slide 9

I/O

- As noted previously — hardware is diverse, and communicating with it may involve a lot of messy details.
- So — typically there is an "I/O subsystem", often involving multiple layers of abstraction. More later!

Slide 10

Hardware, Software, and History

- Textbook has a section called "Ontogeny Recapitulates Phylogeny". Many interesting general observations:
- What seems like a good idea in software is strongly influenced by what the hardware can do. (I think it goes the other way too, but that's speculation.)
- As in other areas of human endeavor, evolution of operating systems is in some ways cyclic: What seems brilliant now may be "ready for the scrap heap" in a few years — and then resurface as brilliant later. (This is why it's not useless to read about approaches not currently in use.)

Slide 11

Operating System Structures

- Clearly o/s could involve a whole lot of code (tens of millions of lines of code, if one can believe Wikipedia). How to structure?
- Choices include:
  - Monolithic systems.
  - Layered systems.
  - Microkernels.
  - Client-server model.
  - Virtual machines.
  - Exokernels.
- (More next time?)

Slide 12

Minute Essay

- Recently I heard a story on NPR that described changes to Apple's operating system for mobile devices, iOS. But to me it sounded more like what is changing is the user interface. Skim the full story

  (http://www.cs.trinity.edu/~bmassing/local-only/CS3323_2013spring/0911.html)

  and tell me what you think: Are they changing the operating system or the user interface — or is that even a meaningful distinction?