Slide 1

Administrivia

- For minute essays where there's some notion of a "right answer", it will be in the final version of the slides online, sometime after class.

Slide 2

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 3

Processors — Recap

- Basic cycle (fetch instruction, decode, execute) — good first approximation, but most modern processors are more complicated. In theory they behave as if the approximation were true, though.
- Dual-mode operation and privileged instructions help with writing an o/s that can defend itself.
- Interrupt mechanism — useful in various ways (more later).

Slide 4

Memory Hierarchy — Recap

- Registers included with processor; explicitly managed by software.
- Caches can be part of processor or separate; mostly/entirely managed by hardware. If more than one processor, caches can be shared — which can get interesting (more later?).
- Main memory (RAM) also explicitly managed by software. Interface between processors and memory often involves an MMU that can provide memory protection as well as relocation (more later). (Review minute essay from last time.)
- Disks and tapes can be considered lower levels in the hierarchy.

Slide 5

I/O Devices

- What they provide (from the user's perspective):
  - Non-volatile storage (disks, tapes).
  - Connections to outside world (keyboards, microphones, screens, etc., etc.).
- Distance between hardware and "virtual machine" is large here, so usually think in terms of:
  - Layers of s/w abstraction (as with other parts of o/s).
  - Layers of h/w abstraction too: most devices attached via controller, which provides a h/w layer of abstraction (e.g., "IDE controller").

Slide 6

I/O Basics

- CPU communicates with device controller by reading/writing device registers; device controller communicates with device.
- Memory-mapped I/O versus I/O instructions.
- Polling versus interrupts.
- Functionality for a particular device packaged as "device driver".
- I/O in application programs — make system call to invoke o/s services (more about system calls later).

Slide 7

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 8

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 9

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 10

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 11

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 12

Command Shells

- History — early batch systems had to interpret "control cards"; modern equivalent is to interpret "commands" (usually interactive).
- Not technically part of o/s, but important and related.
- Typical shell functionality:
  - Invocation of programs (optionally in background).
  - Input/output redirection.
  - Program-to-program connections (pipes).
  - "Wildcard" capability.
  - Scripting capability.
- Examples — MS-DOS `command.com`; UNIX `sh`, `bash`, `csh`, `tcsh`, `ksh`, `zsh`, ...

Slide 13

Hardware, Software, and History

- Textbook has a section called “Ontogeny Recapitulates Phylogeny”. Many interesting examples of some 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 14

Minute Essay

- Multiuser systems often devote considerable attention to sharing a processor (or a few processors) among many processes. Can you think of a hardware change that would allow simplifying or even eliminating this part of the o/s? Does it seem feasible?
- Can you think of another hardware change that would allow simplifying or even eliminating some other source of o/s complexity? Does it seem feasible?

Slide 15

Minute Essay Answer

- A partial answer — perhaps if there were so many processors that one could "permanently" assign a different one to each process. Doesn't seem feasible right now, but who knows?
- A partial answer — if systems had "enough" memory, all the complexity of keeping some parts of processes' address spaces on disk could go away. Sounds more promising, but history suggests that however much memory is provided, people will use it.