Slide 1

Administrivia

- Reminder: Homework 6 due today.
- Next homework (about I/O) to be assigned Wednesday, due after holiday.

Slide 2

Minute Essay From Last Lecture

- Several people said the programming problems were fun and/or interesting.
- For the first problem, it *is* interesting how the order in which you access array elements can affect performance.
- And yes, the point of the second problem was to have you think through the details of at least one page-replacement algorithm.

Slide 3

I/O Continued — Device Specifics

- Textbook presents a tour of major classes of devices. For each, it looks first at what the hardware can typically do, and then at what kinds of device-driver functionality we might want to provide.
- Worth reviewing; we will look at a few today. (In reading, okay to skim things not mentioned in lecture.)

Slide 4

Disks — Hardware

- Magnetic disks:
  - Cylinder/head/sector addressing may or may not reflect physical geometry — controller should handle this.
  - Controller may be able to manage multiple disks, perform overlapping seeks.
- RAID (Redundant Array of Inexpensive/Independent Disks):
  - Basic idea is to replace single disk and disk controller with “array” of disks plus RAID controller.
  - Two possible payoffs — redundancy and performance (parallelism).
  - Six “levels” (configurations) defined. Read all about it in textbook if interested.

Slide 5

Disks — Hardware, Continued

- Solid-state disks/drives — not much in the textbook, but Wikipedia article (usual caveats!) has some details. Executive-level summary:
  - Basic idea is to provide something that to the O/S looks like a traditional disk but without moving parts. Various implementations.
  - Some implementations provide non-volatile storage, but not all do.
  - Lack of moving parts means access times don't include seek time; many implications.
  - Currently faster but more costly and (sometimes?) less reliable. Also some implementations limit number of writes to particular block.

Slide 6

Disk Formatting

- Low-level formatting — each track filled with sectors (preamble, data, ECC bits).
- Higher-level formatting — master boot record, partitions (logical disks), partition table. Master boot record points to boot block in some partition. Partition table gives info about partitions (size, location, use).
- Partition formatting — boot block, blocks for file system.

Slide 7

Disks — Software

- Many devices really pretty much have to be controlled by one process at a time — keyboard, mouse, etc.
- Disks, however, often(?) need to be able to service many processes more or less concurrently.
- So device drivers typically have some way of queueing requests and managing the queue.

Slide 8

Disk Arm Scheduling Algorithms

- A little more about hardware: Time to read a block from disk depends on seek time, rotational delay, and data transfer time. First two usually dominate.
- Typical device driver for disk maintains a queue of pending requests (one per disk, if controller is managing more than one). What order to process them in? several "disk arm scheduling algorithms":
  - FCFS (first come, first served).
  - SSF (shortest seek first).
  - Elevator.

  How do they compare with regard to ease of implementation, efficiency?

Slide 9

Disk Error Handling

- Almost all disks have sectors with defects. Some controllers can recognize them (repeated failures) and avoid them; if not, O/S (device driver) must do this.
- Other kinds of errors also possible, e.g., failure to correctly position read/write head; also must be handled either by controller (if possible) or O/S.

Slide 10

Clocks — Hardware

- System clock — can be simple or programmable. Programmable clock can generate either one interrupt after specified interval or periodic interrupts ("clock ticks").
- Backup clock — usually battery-powered, used at startup and perhaps periodically thereafter.

Slide 11

Clocks — Software

- Clock(s) can be treated as I/O devices, with device driver(s). Functions to provide:
  - Maintain time of day.
  - Enforce time limits on processes.
  - Provide timer / alarm-clock function.
  - Do accounting, profiling, monitoring, etc.
  - Do anything required by page replacement algorithm (e.g., turn off R bits in page table entries).
- Provide this functionality in code to be called on periodic clock-tick interrupts.

Slide 12

Character-Oriented Terminals — Hardware Overview

- Hardware consists of character-oriented display (fixed number of rows and columns) and keyboard, connected to CPU by serial line.
- Actual hardware no longer common (except possibly in mainframe world), but emulated in software (e.g., UNIX terminal windows) so old programs still work. (Why does anyone care? those “old programs” include command shells, text editors, etc., which some of us claim are still useful, and likely to be stable.)

Slide 13

Character-Oriented Terminals — Keyboard

- Hardware transmits individual ASCII characters.
- Device driver can pass them on one by one without processing, or can assemble them into lines and allow editing (erase, line kill, suspend, resume, etc.). Typically provide both modes ("raw" and "cooked").
- Device driver should also provide:
  - Buffering, so users can type ahead.
  - Optional echoing.

Slide 14

Character-Oriented Terminals — Display

- Hardware accepts regular characters to display, plus escape sequences (move cursor, turn on/off reverse video, etc.).

  In the old days, escape sequences for different kinds of terminals were different — hence the need for a `termcap` database that allows calling programs to be less aware of device-specific details.
- Device driver should provide buffering.

Slide 15

Character-Oriented Terminals — Programs

- Many examples of software that uses this kind of device — basically, anything text-based but not line-oriented, for example text editors such as `vim`, `emacs` (in non-graphical mode).
- Libraries for writing such software are system-dependent. One commonly used in UNIXworld is `ncurses`.

Slide 16

GUIs — Hardware Overview

- PC keyboard — sends very low-level detailed info (keys pressed/released); contrast with keyboard for character-oriented terminal.
- Mouse — sends (delta-x, delta-y, button status) events.
- Display can be vector graphics device (rare now, works in terms of lines, points, text) or raster graphics device (works in terms of pixels). Raster graphics device uses graphics adapter, which includes:
  - Video RAM, mapped to part of memory.
  - Video controller that translates contents of video RAM to display. Typically has two modes, text and bitmap.

  High-end controllers may incorporate processor(s) and local memory. (Indeed, they're becoming usable for general-purpose computing — "GPGPU".)

Slide 17

GUI Software — Basic Concepts

- "WIMP" — windows, icons, menus, pointing device.
- Can be implemented as integral part of O/S (Windows) or as separate user-space software (UNIX).

Slide 18

GUIs — Keyboard

- Hardware delivers very low-level info (individual key press/release actions).
- Device driver translates these to character codes, typically using configurable keymap.

Slide 19

GUIs — Display (Windows Approach)

- Each window represented by an object, with methods to redraw it.
- Output to display performed by calls to GDI (graphics device interface) — mostly device-independent, vector-graphics oriented.

Slide 20

GUIs — Display (Traditional UNIX Approach)

- X Window System (the pedantic call it that and not “X Windows”) designed to support both local input/output devices and network terminals, based on a client/server model.
- “Clients” here are programs that want to do GUI I/O; “server” is a program that provides GUI services. An “X server” can run on the same system as the clients, a different UNIX system, an “X terminal (where it’s the “O/S”), or under another O/S (“X emulators” for Windows).
- Core system is client/server communication protocol (input and display events akin to those in Windows) and windowing system. “Window manager” and/or “desktop environment” is separate, as are “widget” libraries. Modularity makes for flexibility and portability, at a cost in performance. Some Linux distributions moving toward alternatives (presumably to emphasize performance over flexibility).

Slide 21

GUIs — Programs

- Of course, many examples of software using this kind of device.
- Libraries for writing such software vary by language:
- Java and Scala include lots of library classes, mostly fairly high-level/abstract.
- Nothing standard in C, but most platforms offer various libraries. Lowest-level one in UNIXworld is “X11”.

Slide 22

Minute Essay

- Anything noteworthy about Homework 6?