Slide 1

Administrivia

- Homework 4 on Web, due 11/28.
- Class Wednesday optional — you'll get an extra attendance point for showing up.

Slide 2

I/O Continued — Device Specifics

- Next, a tour of major classes of devices. For each, we look first at what the hardware can typically do, and then at what kinds of device-driver functionality we might want to provide.

Slide 3

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 4

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 (turn off R bits in page table entries, e.g.).
- Provide this functionality in code to be called on clock-tick interrupts.

Slide 5

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 in mainframe world), but emulated in software (e.g., Unix xterm) so old programs still work. (Why does anyone care? some of those old programs are still useful — e.g., text editors — and usually very stable.)

Slide 6

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.
- Device driver should also provide:
  - Buffering, so users can type ahead.
  - Optional echoing.

Slide 7

Character-Oriented Terminals — Display

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

  In olden 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 8

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. Has two modes, text and bitmap.

Slide 9

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 10

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 11

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. A `.wmf` file (Windows `metafile`) represents a collection of calls to GDI procedures.

Slide 12

Network Terminals — Hardware

- Keyboard, mouse, and display as described previously, plus local processor; connected to remote system.
- Local processor can be very capable (X terminal, or even PC configured to run as one) or more primitive (SLIM terminal).

Slide 13

GUIs — Display (Unix Approach)

- X Window System designed to support both local input/output devices and network terminals, in terms of:
  - Programs that want to do GUI I/O.
  - Program that provides GUI services. Can run on the same system as applications, a different Unix system, an X terminal (where it's the "o/s"), or under another o/s ("X emulators" for Windows — e.g., Exceed, XFree86).

  Which is the "client" and which the "server"?
- Core system is client/server communication protocol (input, 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.

Slide 14

GUI-Based Programming

- Input from keyboard and mouse captured by o/s and turned into messages to process owning appropriate window.
- Typical structure of GUI-based program is a loop to receive and dispatch these messages — "event-driven" style of programming.
- Details vary between Windows and X, but overall idea is similar. See example programs in textbook.

Slide 15

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 and RAID controller.
  - Two possible payoffs — redundancy and performance (parallelism).
  - Six "levels" (configurations) defined.
- Optical disks — CD, CD-R, CD-RW, DVD. Okay to skim details!

Slide 16

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 (more about that in next chapter).

Slide 17

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.
- Earlier we said that 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.
  - SSF (shortest seek first).
  - Elevator.

Slide 18

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 19

I/O in Unix/Linux

- Access to devices provided by special files (`/dev/*`), to provide uniform interface for callers. Two categories, block and character. Each defines interface (set of functions) to device driver. Major device number used to locate specific function.
- For block devices, buffer cache contains blocks recently/frequently used. (See figure on p. 729.)
- For character devices, optional line-discipline layer provides some of what we described for text-terminal keyboard driver. (See figure on p. 729.)
- Streams provide additional layer of abstraction for callers — can interface to files, terminals, etc.

Slide 20

I/O in Windows

- Hardware Abstraction Layer (HAL) attempts to insulate rest of o/s from some low-level details — e.g., I/O using ports versus memory-mapped I/O. (See figure p. 779.)
- Standard interface to device drivers — Windows Driver Model. Drivers are passed I/O Request Packet objects. (See figure on p. 829.)
- Interesting comparison of o/s sizes on p. 771.

Slide 21

Minute Essay

- None — sign in.