

### Administrivia

Slide 1

- Reminder: Homework 4 on the Web; due Monday 12/1. No more than two more short homeworks, to be assigned by 12/1 and due last day of class.
- Quizzes 5 and 6 12/1 and 12/3 respectively. (Quiz 6 may be dropped based on responses to e-mail.)
- Quiz 4 scores generally very low. Remember that I drop your lowest quiz score, and also that grades are "curved".

### I/O Continued — Device Specifics

Slide 2

- 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 but we will look today at only a few. (In reading, okay to skim things not mentioned in lecture.)

### Disks — Hardware

Slide 3

- 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.

### Disks — Hardware, Continued

Slide 4

- 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 writes to particular blocks.

## Disk Formatting

Slide 5

- 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.

## Disk Arm Scheduling Algorithms

Slide 6

- 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 (first come, first served).
  - SSF (shortest seek first).
  - Elevator.How do they compare with regard to ease of implementation, efficiency?

### 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 7

### 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 8

### Clocks — Software

Slide 9

- 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 periodic clock-tick interrupts.

### Character-Oriented Terminals — Hardware Overview

Slide 10

- 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 terminal windows) 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.)

### Character-Oriented Terminals — Keyboard

Slide 11

- 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.

### Character-Oriented Terminals — Display

Slide 12

- 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.

### 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 13

### 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 14

### 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 15

### 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. (Hm!)

Slide 16



Slide 17

### GUIs — Display (Traditional UNIX Approach)

- X Window System (its real name) 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).

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. Some Linux distributions moving toward alternatives (presumably to emphasize performance over flexibility).

Slide 18

### Minute Essay

- A while back I argued with a Windows person about schemes for representing devices: UNIX uses "special files", normally in `/dev` but can be anywhere, identifiable as different from normal files; Windows puts them all at the top level, prefix similar to drive letter.

Which seems more logical to you, and why? from the standpoint of end users, application programmers, o/s developers?