Slide 1

Administrivia

- Homework 5 (about I/O) coming soon.

Slide 2

Minute Essay From 10/31

- (Question was about memory-mapped I/O versus special I/O instructions, with regard to ease of writing device drivers in C. Some answers were interesting!)
- Which requires you to know more about the device? (Aren't they the same?)
- Which can be done directly from C? (Need to be able to set a pointer to a numeric value.)
- Could you use a library function to package the special instructions? (Probably.) How about a system call? (Probably/maybe — efficient?)
- (Textbook also discusses other tradeoffs — e.g., hardware complexity, reduction in address space.)

Slide 3

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. Read all about it in textbook if interested.
- Optical disks — CD, CD-R, CD-RW, DVD. Okay to skim details!

Slide 4

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 5

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 (first come, first served).
  - SSF (shortest seek first).
  - Elevator.

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

Slide 6

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

Other I/O-Related Topics

- "Stable storage" — use two disks to provide what appears to be a single more reliable one (i.e., write either succeeds or leaves old data in place).
- Power management significant — some devices have "sleeping" and "hibernating" states, o/s can try to determine when it would make sense to use them. Example — screen blanking.

Slide 8

I/O in Unix/Linux

- Access to devices provided by special files (normally in `/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. (This is what you access with `*scanf`, `*printf`.)

  (Aside: How do you get the man page for the `printf` function? (`man printf` gives you something else.) Can be several man pages for given name, in different "sections". Get all of them with `man -a`.)

Slide 9

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 10

Minute Essay

- Recently 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?
- This wraps up what I plan to say about I/O (though not about filesystems, which we'll talk about starting next time). Anything else you'd like to hear about?