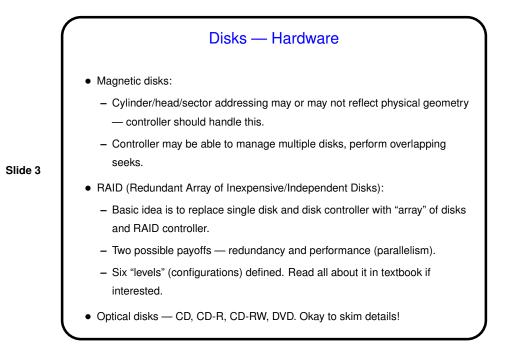
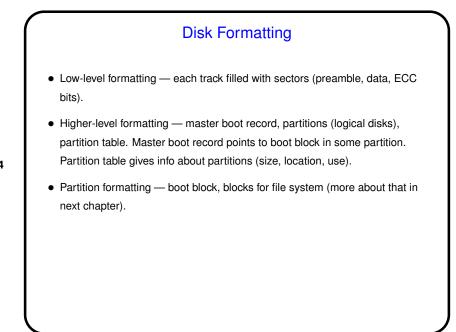
Slide 2

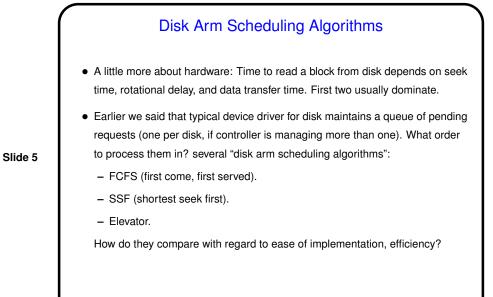


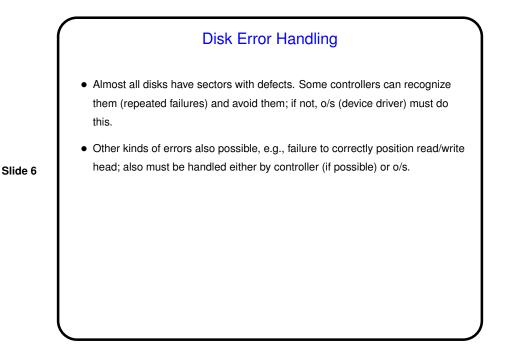
Minute Essay From 10/31
(Question was about memory-mapped I/O versus special I/O instructions, with regard to ease of writing device drivers in C. Some answers were interesting!)
Which requires you to know more about the device? (Aren't they the same?)
Which can be done directly from C? (Need to be able 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 4





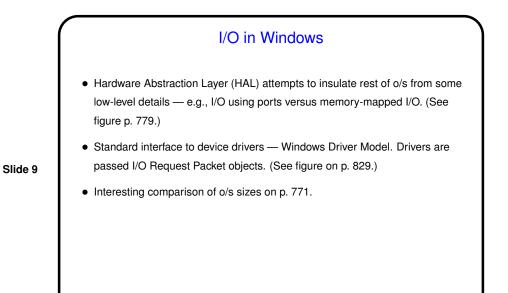
Other I/O-Related Topics

- "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 7

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.)



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?

Slide 10