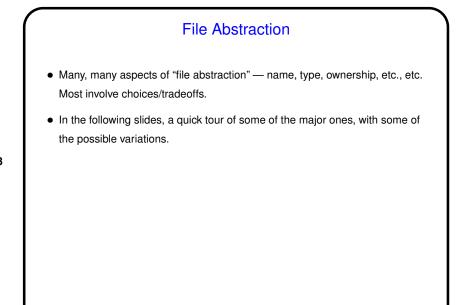
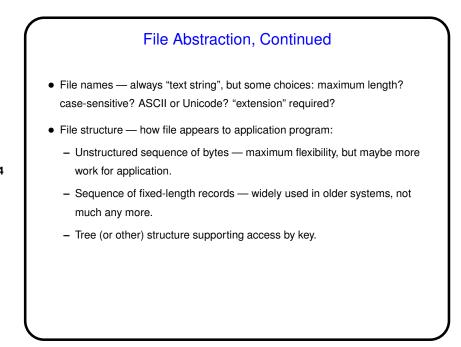
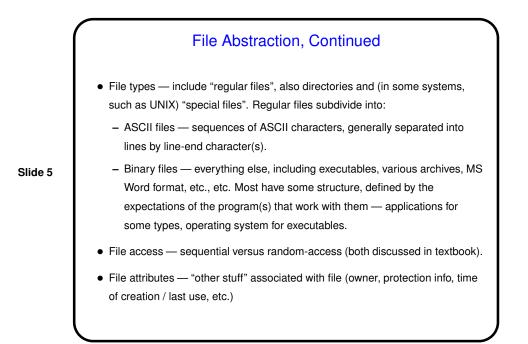


1





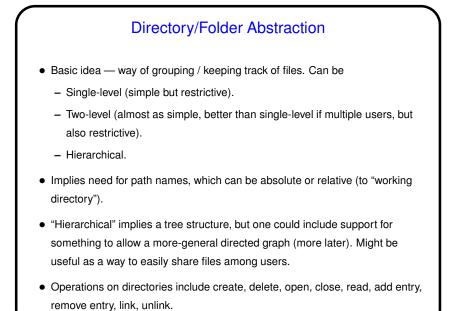


### File Abstraction, Continued

- File operations (things one can do to a file) include create, delete, open, close, read, write, get attributes, set attributes.
- Many systems also support operations for "memory-mapped files" (read whole file into memory, process there, write back out — as mentioned in previous discussion of virtualizing memory).

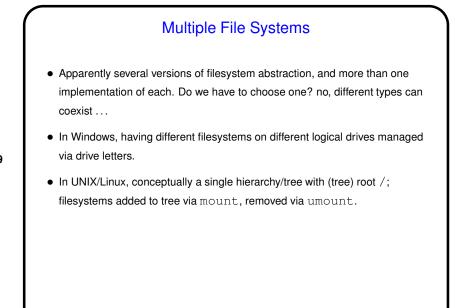
Slide 6

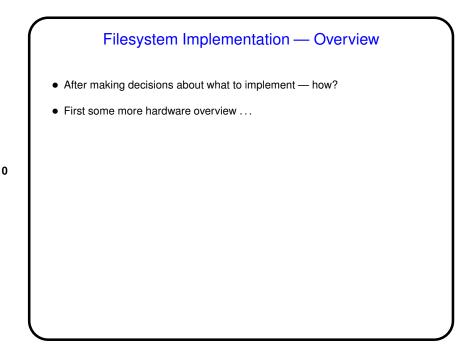
### 3

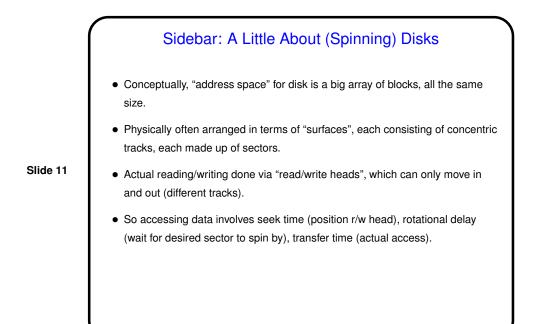


# Linux/UNIX "Everything's a File" UNIX represents a lot of resources as "files" (so that programmers can work with them using familiar(?) mechanisms for accessing files). /dev contains "special files" representing I/O devices, real and pretend ("pseudo-terminals"). Somewhat similar is /proc, which presents information about system and all running processes as "files" (but they aren't really). /sys (Linux-specific?) is similar.

Slide 8







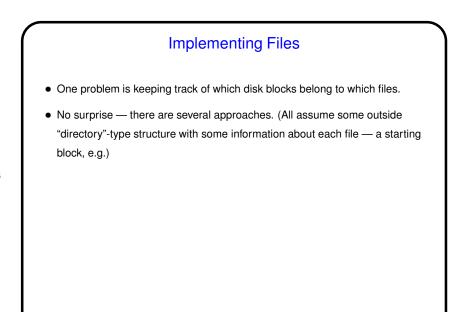
## Filesystem Implementation — Overview Continued

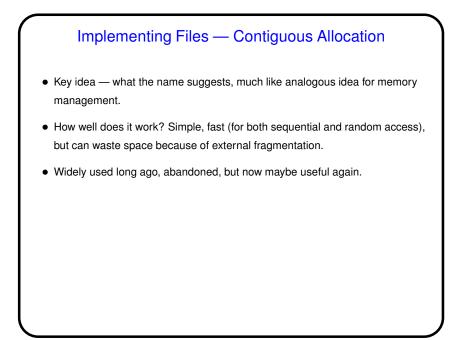
- Basic organization of disk:
  - Master boot record (includes partition table).
  - Partitions, each containing boot block and lots more blocks. Abstract view of access to disk is in terms of reading/writing specified block.

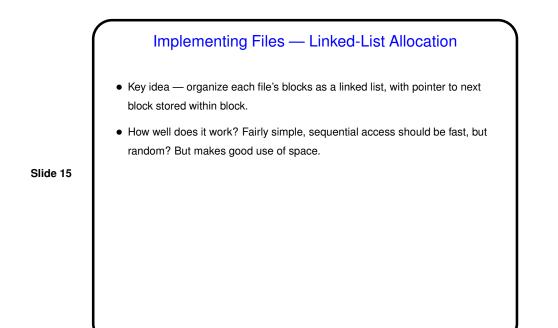
Slide 12

• How to organize/use those "lots more blocks"? Must keep track of which blocks are used by which files, which blocks are free, directory info, file attributes, etc., etc.

Typically start with superblock containing basic info about filesystem, then some blocks with info about free space and what files are there, then the actual files.





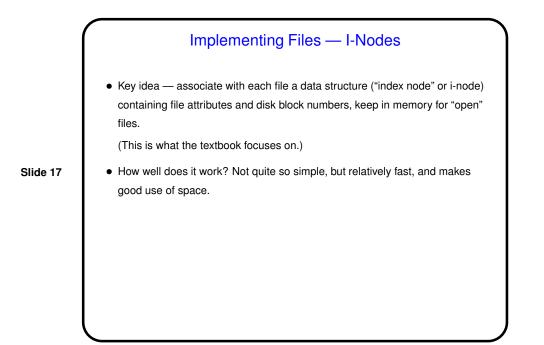


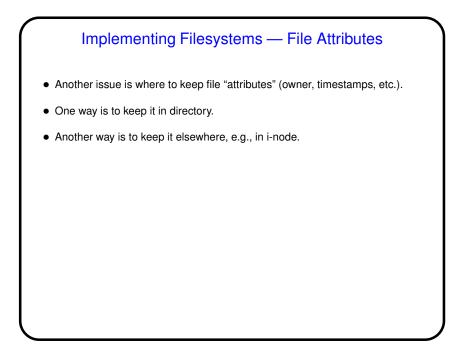
### Implementing Files — Linked-List Allocation With Table In Memory

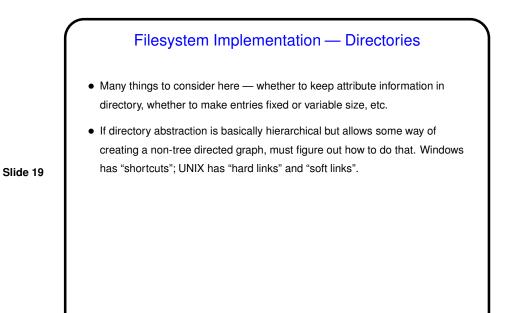
• Key idea — keep linked-list scheme, but use table in memory (File Allocation Table or FAT) for pointers rather than using part of disk blocks.

Slide 16

• How well does it work? Same advantages as what was just described, plus random access is fast. Works pretty well for small disks, not so well for large ones (consider table size! and it must be kept in memory).

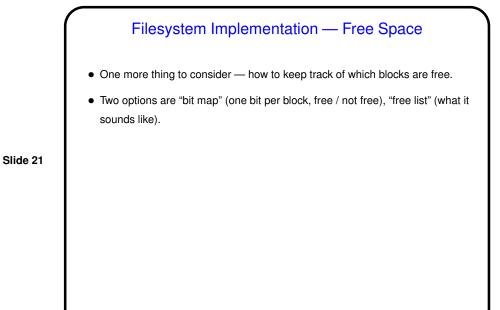




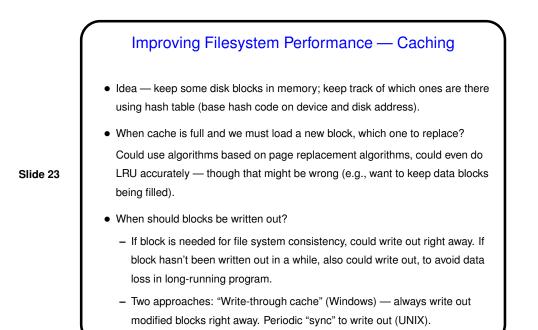


UNIX Filesystems — Hard Links versus Symbolic Links

- "Hard" links allow multiple directory entries to point to the same i-node.
- "Soft" (symbolic) links are a special type of file containing a pathname (absolute or relative).
- (Why two? Good question. Compare and contrast ...)



# **Filesystem Performance** • Access to disk data is much slower than access to memory, with seek time being slowest, then rotational delay, then transfer time. (Well, for disks that spin. Solid-state disks don't, but they may have their own issues, e.g., limits on number of writes?) • So, file systems include various optimizations .... Slide 22



### Improving Filesystem Performance — Block Read-Ahead

 Idea — if file is being read sequentially, can read some blocks "ahead". (Of course, doesn't help if file is being read non-sequentially. Decide based on recent access patterns.)

### Improving Filesystem Performance — Reducing Disk Arm Motion

- The less we have to move the read/write heads around (seek time), the better overall performance will be.
- Drivers for (spinning) disks typically have requests queued. So it makes sense to try to rearrange this queue a little to minimize seek time. Textbook discusses several algorithms.
- Also helps to group blocks for each file together as much as possible.
- And if i-nodes are being used, helps to place them so they're fast to get to (and so maybe we can read an i-node and associated file block together).

### Filesystems — What Do Current Systems Use?

- Linux default is now probably ext4. Other filesystems possible/supported, and support for accessing various Windows filesystems provided via Samba.
- Mac OS X ("macOS"?) Apple File System, externally pretty UNIX-like, possibly internal differences.

Slide 26

• Windows — NTFS is default, support still provided for FAT-xx.

