Slide 1

## Administrivia

- Reminder: Homework 5 due today.

Slide 2

## Minute Essay From Last Lecture

- (Review question.)
- Problems if different filesystems use extensions to mean different things. (Yes, maybe . . . Is that really a function of the filesystem?)

  (One student reported an incident involving copying CSS/HTML from Windows to OS X — ?)
- Problems if different rules about what can be in a filename — maybe some files can't be copied. Similar problems with case sensitivity differences.
- Some sort of adapter needed to deal with different filesystems.
- Problems with differences in absolute pathnames. (Maybe — if this information is *in* the filesystem and not the o/s.)

Slide 3

## Minute Essay From Last Lecture, Continued

- Problems if different ideas about end of file.
- Problems with different ways of storing files on disk. (I'm skeptical — If o/s provides a way to access a type of filesystem at all, it should "just work"? if properly implemented anyway?)
- Problems if different encodings. (Yes, maybe. Similar potential issues with line-end conventions. o/s might provide some help here.)
- Problems if different ideas about permissions, or other "extra" information.

Slide 4

## Filesystem Implementation — Overview

- Last time we talked about many aspects of filesystem abstraction. After making decisions about what to implement — how?
- Recall(?) basic organization of disk:
  - Master boot record (includes partition table)
  - Partitions, each containing boot block and lots more blocks.
- 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.

Slide 5

## Implementing Files

- One problem is keeping track of which disk blocks belong to which files.
- No surprise — there are several approaches. (All assume some outside "directory"-type structure with some information about each file — a starting block, e.g.)

Slide 6

## Implementing Files — Contiguous Allocation

- Key idea — what the name suggests, much like analogous idea for memory management.
- How well does it work? consider simplicity, speed (both sequential and random access), possibility of fragmentation (wasted space).
- Widely used long ago, abandoned, but now maybe useful again.

Slide 7

## Implementing Files — Linked-List Allocation

- Key idea — organize each file's blocks as a linked list, with pointer to next block stored within block.
- How well does it work? consider simplicity, speed (both sequential and random access), possibility of fragmentation (wasted space).

Slide 8

## 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.
- How well does it work? consider simplicity, speed (both sequential and random access), possibility of fragmentation (wasted space).

Slide 9

## Implementing Files — I-Nodes

- Key idea — associate with each file a data structure ("index node" or i-node) containing file attributes and disk block numbers, keep in memory.
- How well does it work? consider simplicity, speed (both sequential and random access), possibility of fragmentation (wasted space).

Slide 10

## Filesystem Implementation — Recap

- Idea of filesystems — directory entry for a file points to something we can use to find file's blocks:
  - First block and size of contiguous sequence.
  - First block of linked list of blocks.
  - Entry in FAT, which points to first block and holds linked lists.
  - I-node, which contains list of blocks.

  Directory entry can also contain file attributes, or they can be stored elsewhere (e.g., in i-node).
- Notice how this is somewhat analogous to memory management — similar tradeoffs.

Slide 11

## Filesystem Implementation — Directories

- Many things to consider here — whether to keep attribute information in directory, whether to make entries fixed or variable size, etc.
- Also consider whether to allow some sort of sharing (making the hierarchy a directed graph rather than a tree). Different possibilities here; contrast UNIX "hard links" (in which different directory entries point to a common structure describing the file) and "soft (symbolic) links" (in which the link is a special type of file).

Slide 12

## Managing Free Space — Free List

- One way to track which blocks are free — list of free blocks, kept on disk.
- How this works:
  - Keep one block of this list in memory.
  - Delete entries when files are created/expanded, add entries when files are deleted.
  - If block becomes empty/full, replace it.

Slide 13

## Managing Free Space — Bitmap

- Another way to track which blocks are free — "bitmap" with one bit for each block on disk, also kept on disk.
- How this works:
  - Keep one block of map in memory.
  - Modify entries as for free list.
- Usually requires less space.

Slide 14

## Minute Essay

- I mentioned that the contiguous-allocation scheme for files had gone out of favor but is now sometimes useful again. What are some circumstances in which it might be a good choice (particular media, particular files, etc.)?

Slide 15

## Minute Essay Answer

- It could be a good choice for a write-once medium, assuming every files is written all at once (rather than, say, writing some blocks of one file, then some blocks of another, and so forth).
- It might even be a good choice in situations in which faster access outweighs other considerations (such as difficulties in making files bigger, or the potential for fragmentation).