Slide 1

Administrivia

- (None.)

Slide 2

Minute Essay From Last Lecture — Some Responses

- Would be good for big audio/video files, or anything where pauses in obtaining next part of file would be a problem.
- (Is it a good idea, though, to insist that *all* files be allocated this way? Are there other solutions?)

Slide 3

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. (Review slide from last time.)

  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 4

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 5

Journaling Filesystems — Overview

- As we'll discuss later (and as you may know!) — o/s sometimes doesn't perform "write to disk" operations right away (caching).
- One result is likely improved performance. Another is potential filesystem inconsistency — operations such as "move a block from the free list to a file" are no longer atomic.
- Idea of journaling filesystem — do something so we *can* regard updates to filesystem as atomic.
- To say it another way — record changes-in-progress in log, when complete mark them "done".

Slide 6

Journaling Filesystems, Continued

- Can record "data", "metadata" (directory info, free list, etc.), or both.
- "Undo logging" versus "redo logging":
  - Undo logging: First copy old data to log, then write new data (possibly many blocks) to disk. If something goes wrong during update, "roll back" by copying old data from log.
  - Redo logging: First write new data to log (i.e., record changes we're going to make), then write new data to disk. If something goes wrong during update, complete the update using data in log.
- A key benefit — after a system crash, we should only have to look at the log for incomplete updates, rather than doing a full filesystem consistency check.

Slide 7

Journaling Filesystems Versus Log-Structured Filesystems

- Log-structured filesystem — *everything* is written to log, and only to log. Seems like an interesting idea, but tough to implement on real systems.
- Journaling filesystem — log contains only recent and pending updates.

Slide 8

Virtual File Systems

- Apparently many possibilities for implementing filesystem abstraction, with the usual tradeoffs. Do we have to choose one, or can different types coexist? The latter . . .
- In Windows, different filesystems on different logical drives is managed via drive letters.
- In UNIX, current approach is usually a “virtual file system” — basically, an extra layer of abstraction (remember the adage about how that can solve any programming problem).

Slide 9

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 10

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 11

Filesystem Reliability — Backups

- Why do backups? sometimes data is more valuable than physical medium, and might need to
  - Recover from disaster (rare these days, but possible).
  - Recover from stupidity (less rare – hence "recycle bin" idea).
- Many issues involved — which files to back up, how to store backup media, etc., etc. — see textbook.

Slide 12

Filesystem Reliability — Consistency Checks

- Can easily happen that true state of filesystem is represented by a combination of what's on disk and what's in memory — a problem if shutdown is not orderly.
- Solution is a "fix-up" program (UNIX `fsck`, Windows `scandisk`). Kinds of checking we can do:
  - Consistency check: For each block, how many files does it appear in (treating free list as a file)? If other than 1, problem — fix it as best we can.
  - File consistency check: For each file, count number of links to it and compare with number in its i-node. If not equal, change i-node.
  - Etc., etc. — see text.

Slide 13

Filesystems — Quotas

- Why have quotas? Disk space is cheap, right? yes, but more space used means more to back up, and on multi-user systems there are fairness issues, and the possibility that one careless user will affect others.
- Implementation involves keeping track, for each user, of space used versus space allowed. Must be updated every time a file is changed/created/deleted. Some systems allow “grace period”, but eventually all will disallow, for user over quota, creation of new files or expansion of existing files.

Slide 14

Minute Essay

- List as many reasons as you can think of why there seem to be so many different kinds of filesystems.