Slide 1

Administrivia

- Exam 2 will be December 2. Review sheet on Web soon.
- Homework 4 on Web. Due November 30 at 5pm. (Not accepted late.)

Slide 2

Minute Essay From Last Lecture

- What to do about the fact that the classroom computers are a distraction?

  No consensus on what to do! so I will try various things.

  But do remember — what's on your screen, even if quiet, may be distracting your neighbor.

Slide 3

Filesystem Implementation — Recap

- Recall 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.
- Must also manage free space. Issues include . . .

Slide 4

Blocksize

- "I/O software" can provide a device-independent blocksize (and translate to cylinder/track/sector disk addresses).
- How big should blocks be?
  - What if they're really big?
  - What if they're really small?
  - Usually compromise, also consider page size.

Slide 5

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 6

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 7

Filesystem Reliability — Backups

- Why do backups? sometimes data is more valuable than physical medium, and might need to
  - Recover from disaster (rare).
  - 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 8

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 9

Filesystem Performance

- Access to disk data is much slower than access to memory — seek time plus rotational delay plus transfer time.
- So, file systems include various optimizations . . .

Slide 10

Improving Filesystem Performance — Caching

- Idea — keep some disk blocks in memory; keep track of which ones are there using hash table (base hash code on device and disk address).
- When cache is full and we must load a new block, which one to replace?
  Could use algorithms based on page replacement algorithms, could even do LRU accurately — though that might be wrong (e.g., want to keep data blocks being filled).
- When should blocks be written out?
  - If block is needed for file system consistency, could write out right away. If block hasn't been written out in a while, also could write out, to avoid data loss in long-running program.
  - Two approaches: "Write-through cache" (Windows) — always write out modified blocks right away. Periodic "sync" to write out (Unix).

Slide 11

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.)

Slide 12

Improving Filesystem Performance — Reducing Disk Arm Motion

- Group blocks for each file together — easier if bitmap is used to keep track of free space. If not grouped together — "disk fragmentation" may affect performance.
- Place i-nodes so they're fast to get to (and so maybe we can read an i-node and associated file block together).

Slide 13

Example Filesystem — Unix V7

- Filename restriction — each part of path name at most 14 characters.
- So, directory entry is just 14-byte name and i-node number.
- I-nodes are all stored in a contiguous array at the start of the file system (right after boot block and a "superblock" containing additional parameters).
- What's in each i-node? attributes (permission bits, numeric owner and group ID, timestamps, links count) and list of blocks — last is pointer to more blocks.
- To find a file:
  - Start with root directory — its i-node is in a known place.
  - Scan directory for first part of path, get its i-node, read it, scan for next part of path, etc.
  - Relative path names are handled by including "." and ".." in each directory, so no special code needed.

Slide 14

Disk Fragmentation

- Idea — if blocks that make up a file are (mostly) contiguous, faster to read them all. If not, "disk fragmentation".
- How likely is disk fragmentation? Depends on filesystem, strategy for allocating space for files.
- "Defragmenter" utility can be run to correct it. Windows comes with one. Linux doesn't. The claim is that Unix and Linux filesystems typically don't become fragmented unless the disk is close to full.

Slide 15

Unix Filesystems — Concepts

- (This is summarized from chapter 10, which you can skim if you want more details.)
- Single type of file — sequence of bytes. `lseek` allows random access.
- Single root directory. `mount` allows access to multiple physical devices.
- Links, hard or symbolic, to allow non-tree directory structure.
- Locks to control access to files/records.
- File descriptors for open files.
- "Pipes" — pseudofiles for connecting processes.

Slide 16

Unix Filesystems — Implementation Overview

- Superblock contains critical info — how many i-nodes, location of free list, how many blocks, etc.
- After that? In early implementations, all i-nodes followed by all data blocks. Later implementations (Berkeley FFS) use "cylinder groups", each containing superblock copy, i-nodes, and data blocks — for better performance, reliability.
- Directory entries fixed-size in early implementations, varying-size in later ones.
- I-nodes contain file attributes, link count, list of (some) blocks, pointers to indirect blocks.
- In memory — table of i-nodes for open files, table of file descriptors (containing, e.g., info about position within file).
- NFS allows access to other systems' disks.

Slide 17

Linux Filesystems — Implementation Overview

- Originally, MINIX filesystem only — similar to early Unix.
- Later, VFS (virtual filesystem) added as intermediate layer to support many kinds.
- ext2/ext3 filesystems (ext3 is ext2 with addition of "journal" file and support for journaling): Similar to FFS, but with a single blocksize, and "block groups" rather than "cylinder groups". Block group also includes bitmap for free space.

  Attempts to allocate all space for file within block group (may account for less fragmentation).

  Superblock has bit that says whether filesystem is "clean" (no `fsck` needed at boot time).
- `/proc` filesystem represents much system info.

Slide 18

Windows Filesystems — Concepts

- (This is summarized from chapter 11, which you can skim if you want more details.)
- FAT filesystems as described in chapter 6. Designed for small disks and don't work very well for large ones. Also see description of how support for long filenames was added.
- NTFS filesystems — new with Windows NT. Some basic concepts:
  - Unicode filenames.
  - File can consist of multiple "streams" (not just one as in Unix) — generalization of Mac's data fork / resource fork idea.
  - Transparent compression and encryption.

Slide 19

Windows Filesystems — NTFS Implementation

- MFT (master file table) — analogous to i-nodes. One or more entries per file/directory, plus some for system. Boot block points to start.
- For small files, data is right in MFT record. Otherwise contains list of contiguous sequences of bytes.

Slide 20

Minute Essay

- None — sign in.