Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on Web (written questions now, optional programming problem(s) later today / early tomorrow). Due next Wednesday.
- Homeworks 1 through 5 not accepted past 5pm next Thursday (12/7), except for any optional problems. Sample solutions to be available Friday (12/8).
- Extra-credit problems (Homework 6) to be available soon, due same day as final. Optional problems for other homeworks also due then.

Slide 2

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 3

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 4

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 5

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 6

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 7

Journaling Filesystems — Overview

- Recall — 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 8

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 9

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 with good performance.
- Journaling filesystem — log contains only recent and pending updates.

Slide 10

Minute Essay

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