Slide 1

Administrivia

- Next homework to be on the Web soon, maybe tomorrow. I'll send e-mail if before Wednesday. You'll have (at least) a week to work on it.

Slide 2

Minute Essay From Last Lecture

- Most people came fairly close on the question about different filesystems, but — review question/answer.
- Most people didn't have much to report about Homework 5, but one student ran into an interesting problem . . . Apparently Linux memory management is such that `malloc` can report success even if there might not be enough memory. Presumably the hope is that when the memory is needed it will likely be there, but if not — runtime crash. Web search for "Linux OOM killer" for more information.

Slide 3

Linux Memory Management — the “OOM Killer”

- Apparently on (some?) Linux systems `malloc` returns true as long as you haven’t asked for more memory than you’re allowed to have. But it doesn’t actually try to find space for the allocated memory (either in real memory or on disk) until it’s used — it “overcommits” memory resources.
- So what happens if a process tries to use space that was allocated but not previously used? system tries to find some — and if it can’t, it calls the “OOM killer” to terminate one or more processes.
- (My first reaction is “what a bad design?” but it may make sense?)

Slide 4

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).
- Already mentioned — `/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 5

UNIX Filesystems — Hard Links versus Symbolic Links, Revisited

- As mentioned previously, many filesystems provide a mechanism for creating not-strictly-hierarchical relationships among files/folders. UNIX typically has two:
  - “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 . . .)

Slide 6

Example Filesystem — MS-DOS FS

- Filename restriction — eight-character name plus three-character extension. (!) (Textbook doesn’t say this, but there are/were ways of faking longer names, basically by mapping longer names into inscrutable short-enough ones.)
- Directory entries contain filename, attributes, timestamp, size, and block number of first block. How are other blocks found? FAT (File Allocation Table).
- Various versions depending on how many bits used to store block number (FAT-12, FAT-16, FAT-32, though the last is apparently really FAT-28). Each defines a set of permitted block sizes, all multiples of 512K.
- Simple, which is good, but imposes limits on file size and partition size. Keeping entire FAT in memory may also be a problem.

Slide 7

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 three are pointers to “single indirect”, “double indirect”, and “triple indirect” blocks. (See figure 4-33 in textbook.)

Slide 8

Example Filesystem — Unix V7, Continued

- 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(!).
- Not so simple, and still imposes a limit on total file size, but flexible? and probably requires less system memory, since only i-nodes for open files need to be in memory.

Minute Essay

- None really — unless questions about filesystems before we move on?