Slide 1

Administrivia

- Homework 3 on Web. Due next Friday.

Slide 2

Minute Essay From Last Lecture — Some Responses

- Single address space would make it easier to give each process the amount of memory it needs (?).
- Simpler to implement (votes for both schemes being simpler!).
- Easier to manage pages on disk with multiple address spaces.
- What if each process's part of the single address space isn't contiguous?
- More difficult to figure out what to swap, keep track of pieces.

Slide 3

Files and Filesystems — Overview

- Very abstract view — requirements for long-term information storage are:
  - Store large amounts of information.
  - Have information survive past end of creating process.
  - Allow concurrent access by multiple processes.
- Usual solution — “files” on disk and other external media, organized into “file systems”.
- In terms of the two views of an o/s:
  - “Virtual machine” view — filesystem is important abstraction.
  - “Resource manager” view — filesystem manages disk (and other device) resources.
- We’ll look first at the user view, then at implementation.

Slide 4

File Abstraction

- Many, many aspects of “file abstraction” — name, type, ownership, etc., etc. Most involve choices/tradeoffs.
- In the following slides, a quick tour of some of the major ones, with some of the possible variations.

Slide 5

File Abstraction, Continued

- File names — always “text string”, but some choices: maximum length? case-sensitive? ASCII or Unicode? “extension” required?
- File structure — how file appears to application program:
  - Unstructured sequence of bytes — maximum flexibility, but maybe more work for application.
  - Sequence of fixed-length records — widely used in older systems, not much any more.
  - Tree (or other) structure supporting access by key.

Slide 6

File Abstraction, Continued

- File types — include “regular files”, also directories and (in some systems, such as UNIX) “special files”. Regular files subdivide into:
  - ASCII files — sequences of ASCII characters, generally separated into lines by line-end character(s).
  - Binary files — everything else, including executables, various archives, MS Word format, etc., etc. Most have some structure, defined by the expectations of the program(s) that work with them — applications for some types, operating system for executables.
- File access — sequential versus random-access.
- File attributes — “other stuff” associated with file (owner, protection info, time of creation / last use, etc.)

Slide 7

File Abstraction, Continued

- File operations (things one can do to a file) include create, delete, open, close, read, write, get attributes, set attributes. Example program using low-level wrappers for system calls on p. 266.
- Many systems also support operations for “memory-mapped files” (read whole file into memory, process there, write back out — as mentioned in previous discussion of memory management).

Slide 8

Directory/Folder Abstraction

- Basic idea — way of grouping / keeping track of files. Can be
  - Single-level (simple but restrictive).
  - Two-level (almost as simple, better if multiple users, but also restrictive).
  - Hierarchical.
- Implies need for path names, which can be absolute or relative (to “working directory”).
- Operations on directories include create, delete, open, close, read, add entry, remove entry.

Slide 9

Filesystem Implementation — Overview

- 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 10

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 11

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 12

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 13

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 14

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 15

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 16

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).