Slide 1

Administrivia

- Reminder: Homework 4 due today, Homework 5 written problems Monday. Homework 5 second programming problem still in work; I will send e-mail (tomorrow?).

Slide 2

Minute Essay From Last Lecture

- One person asked about hot new ideas in memory management. Good question.
- The discussion in the textbook (section 3.8) confirms what would have been my guess — at least of the time of its writing, memory management was pretty a solved problem, though there's some work being done in adapting to changes in hardware, such as SSDs.

Slide 3

Shared Libraries in UNIXworld

- (Later?)

Slide 4

Memory Protection, Revisited

- Paging provides one form of memory protection: If a given page in memory isn't mapped to some page in a process's address space via its page table, the process can't access the page at all.
- But that's "all or nothing", and sometimes it would be useful to have more control. Some MMU hardware supports page table entries that in addition to R and M bits have . . .
- A "read-only" bit that's what its name suggests. So for example there might be a page that's accessible (for reading) to all processes but is writeable only by the O/S.
- An "execution allowed" bit that means it's okay for the processor to fetch instructions from this page. Very useful in defending against classic buffer-overflow attacks (by not setting this bit for stack pages)!

Slide 5

Memory Management in Windows

- Apparently very complex, but basic idea is paging.
- Intraprocess memory management is in terms of code regions (some shared — DLLs), data regions, stack, and area for o/s. “Virtual Address Descriptor” for each contiguous group of pages tracks location on disk, etc.
- Memory-mapped files can make I/O faster and allow processes to (in effect) share memory.
- Demand-paged, with six (!) background threads that try to maintain a store of free page frames. Page replacement algorithm is based on idea of working set.

Slide 6

Memory Management in UNIX/Linux

- Very early UNIX used contiguous-allocation or segmentation with swapping. Later versions use paging. Linux uses multi-level page tables; details depend on architecture (e.g., three levels for Alpha, two for Pentium).
- Intraprocess memory management is in terms of text (code) segment, data segment, and stack segment. Linux reserves part of address space for O/S. For each contiguous group of pages, “vm_area_struct” tracks location on disk, etc.
- Memory-mapped files can make I/O faster and allow processes to (in effect) share memory.
- Demand-paged, with background process (“page daemon”) that tries to maintain a store of free page frames. Page replacement algorithms are mostly variants of clock algorithm.

Slide 7

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 I/O device) resources.
- We’ll look first at the user view, then at implementation.

Slide 8

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 9

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 10

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 11

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. 274.
- 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 12

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 than single-level if multiple users, but also restrictive).
  - Hierarchical.
- Implies need for path names, which can be absolute or relative (to “working directory”).
- “Hierarchical” implies a tree structure, but one could include support for something to allow a more-general directed graph (more later). Might be useful as a way to easily share files among users.
- Operations on directories include create, delete, open, close, read, add entry, remove entry, link, unlink.

Minute Essay

- Anything noteworthy about Homework 4?