Slide 1

Administrivia

- Homework 5 on the Web. Due a week from today. Short! (If you skip the optional programming problem.)

Slide 2

Shared Pages, Revisited

- We talked already about how to using paging to allow processes to share part of their address spaces.
- One use for this might be to optimize operations such as message-passing that involve multiple address spaces.
- Another use — perhaps the most common one — is to allow processes to share code. Fairly straightforward how this works if two processes are running the same program (just share the pages that contain code). A somewhat more complex use, however . . .

Slide 3

Shared Libraries

- Idea of shared libraries is allow another kind of code sharing — not whole programs, but individual functions. “Shared libraries” in UNIX-speak, “DLLs” in Windows-speak.
- An additional advantage of doing this is that functions can be updated without recompiling all programs that use them. (Then again, this can be risky! UNIX-like systems partially solve this problem by allowing multiple copies of the file containing the library code to coexist.)
- Details of how to make this work are worth thinking about . . .

Slide 4

Shared Libraries, Continued

- Review what happens when you compile a program to produce an executable: Translate high-level code into object code, then “link” with other object code to produce something that can be loaded into memory and run. Programs such as `gcc` do this in a way that’s invisible when it works (but may lead to errors that confuse beginners).
- To make the shared-library-function idea work, the linker must link in *something* — a “stub” function, to be further resolved at runtime. Object code for the shared-library functions may need to be compiled with special flags.
- Some systems allow deferring even more until runtime — e.g., Linux “dynamic linking loader” (see function `dlopen`).

Slide 5

One More Memory Management Strategy — Segmentation

- Idea — make program address "two-dimensional" / separate address space into logical parts. So a virtual address has two parts, a segment and an offset.
- To map virtual address to memory location, need "segment table", like page table except each entry also requires a length/limit field. (So this is like a cross between contiguous-allocation schemes and paging.)

Slide 6

Segmentation, Continued

- Benefits?
  - Nice abstraction; nice way to share memory.
  - Flexible use of memory — can have many areas that grow/shrink as required, not just heap and stack — especially if we combine with paging.
- Drawbacks?
  - External fragmentation possible (can offset by also paging).
  - More complex.
  - "Paging" in/out more complex — issues similar to with contiguous-allocation.

Slide 7

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 8

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 9

Minute Essay

- Anything about memory management you'd like to hear more about / have clarified?