Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on the Web (except for second programming problem). Due in a week (except for that not-there-yet problem — TBA via e-mail).

Slide 2

Minute Essay From Last Lecture

- Many people came reasonably close to the "right" answer. However . . .
- "Disks are faster" — true, but so are processors, so the disparity between speeds still holds.
- "Disks are bigger" — also true, but not relevant.
- "Memories are bigger" — true, and *does* make a difference here.
- One person thought maybe that cup of coffee was overheating the disk — interesting but probably not the case, and wouldn't cause more page faults anyway.
- Aside: More than one person, in mentioning increases, said "exponentially". Normally one of my language-usage peeves, but in computing actually sometimes correct!

Slide 3

"Thrashing"

- Recall the notion of a process's "working set" — portion of its address space currently in use.
- Q: What happens if the combined sizes of all active processes' working sets is too big for RAM?
- A: Pretty much what the sysadmins in my minute-essay story observed — system will spend so much time paging it can't do much else.

Slide 4

Memory Management Versus(?) Caching

- Main memory (RAM) is not such a scarce resource on most current mainstream systems, so managing it carefully isn't as important (though it still *could* be).
- What might matter more from an application-programming perspective is not whether the whole program and its data fit into RAM (likely) but whether the current working set fits into cache. Performance differences can be noticeable!

  Example: matrix multiplication, where the difference between the "naive" (obvious) implementation and one that works by blocks can be noticeable, almost surely because of differences in what fits in cache.

Slide 5

Shared Libraries — Recap/Review

- Idea of shared libraries ("DLLs" in Windows-speak) is to keep one copy of code in memory and have all processes that need the code use that. Key advantage is more-efficient use of memory.
- A good-and-bad aspect is that if the shared code is updated, all programs that use it are affected.
- How to make this happen . . . At link time, programs get "stub" versions of functions. References to real versions resolved at load time.
- Resolving references to shared code at load time — finer-grained version of "relocation problem", no? and fixable by making sure library contains only "position-independent code".
- (Still seems like it would be necessary for the shared code to be mapped into all address spaces at the same location. "Hm!"?)

Slide 6

Memory-Mapped File I/O

- Worth mentioning here that some systems also provide a mechanism (e.g., via system calls) to allow reading/writing whole files into/from memory. If there's enough memory, this could improve performance.
- Example of how this works in Linux — `man` page for `mmap`.

Slide 7

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 8

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 9

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 10

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 11

Minute Essay

- Any questions about memory management before we move on?