Slide 1

Administrivia

- Reminder: Homework 4 due Wednesday. Homework 5 written problems due next Monday, programming problems next Wednesday.

Slide 2

Minute Essay From Last Lecture

- Most people thought the exam was more or less what they expected. Good?
- (Review answer to other question.)

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

Paging and Virtual Memory — Recap/Review

- Basic idea is fairly simple: If there are more pages in the union of all process's address spaces than will fit into main memory, keep some (we hope the less-active ones) on disk.
- With this addition, page faults now either mean "invalid address" or "page not in memory but on disk". Page-fault interrupt handler must decide which, and if it's the latter, arrange to bring it in. Similar processing if we want to give a process a new page.
- If memory is not full, not too hard, but if it is? "Steal" a frame from its current owner (write contents to disk first if need be). Choice of page to steal determined by "page replaccment algorithm".
- Many such algorithms possible. (Slides from last time — revised a bit.)

Slide 5

Paging — Other Design Issues/Choices

- Demand paging versus prepaging.
- Global versus local allocation.
- "Paging daemon" that tries to keep a supply of free page frames.
- What to do if page to be replaced is waiting for I/O — probably trouble if we replace it anyway, since the pending I/O, when it completes, may write to a physical address. Solutions include "locking" pages, or doing all I/O to O/S pages and then moving data to user pages.

Slide 6

Modeling Page Replacement Algorithms

- Intuitively obvious that more memory leads to fewer page faults, right? Not always!
- Counterexample — "Belady's anomaly", sparked interest in modeling page replacement algorithms.
- Modeling based on simplified version of reality — one process only, known inputs. Can then record "reference string" of pages referenced.
- Given reference string, p.r.a., and number of page frames, we can calculate number of page faults. (One of the programming problems will ask you to do this.)
- How is this useful? can compare different algorithms, and also determine if a given algorithm is a "stack algorithm" (more memory always means fewer page faults).

Slide 7

Sharing Pages

- Shared pages can be useful, but can also present problems.
- Multiple processes running the same program is relatively easy (why?) but has one potential downside (what?)
- UNIX `fork` system call is — interesting in this context. POSIX definition says that child process's address space is basically a copy of the parent's address space. What's the easy-to-implement way to do this? What downside does that have in current systems? Is there a way to reduce its impact? And why duplicate in the first place?

Slide 8

Sharing Pages and `fork`

- Duplicating pages is easy but inefficient, especially if the child process is going to call `execve` or something similar right away. Some systems use "copy-on-write" to improve efficiency.
- Why did the people who designed UNIX require this duplication . . . Possibly because it makes some things easy (such as setting up parent/child pipes) and wasn't very costly when designed. Windows's system call for creating processes takes a different approach. Maybe that's better!

Slide 9

Sharing Pages, Continued

- One use for shared pages is multiple processes running the same program.
- What about sharing code at a level below whole programs (UNIX "shared libraries", Windows DLLs)?

Slide 10

Shared Libraries

- One attraction is somewhat obvious — if code for library functions (e.g., `printf`) is statically linked into every program that uses it, programs need more memory — seems wasteful if processes can share one copy of code in memory.
- Another attraction is that library code can be updated independently of programs that use it. (But is there a downside to that?)
- How to make this happen . . .

Slide 11

Shared Libraries, Continued

- 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".
- (Possibly some details of how this plays out in Linux next time?)

Slide 12

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 13

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 14

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.

Minute Essay

- I'm planning one more lecture on memory management, to include some details about shared libraries in UNIXworld and an overview of how memory management is done in a few real-world systems. Anything else you'd like to hear about?