Slide 1

Administrivia

- Reminder: Homework 4 due today (5pm).
- Homework 5 on the Web, due next Friday.
- (Review minute essay from last time.)

Slide 2

Paging — Review

- Recall basic ideas (divide address spaces and memory into fixed-size chunks, optionally-but-usually use disk to hold what we hope are less-used parts of processes' address spaces).
- One key issue in making this all work acceptably is how we choose which pages to keep in memory (page replacement algorithm).
- A few more things to consider . . .

Slide 3

Demand Paging Versus Prepaging

- The purest form of paging is “demand paging” — processes are started with no pages in memory, and pages are loaded into memory on demand only.
- An alternative is “prepaging” — try to load pages in advance of demand. How?

Slide 4

Global Versus Local Allocation

- In deciding which page to replace, consider all pages (“global allocation”), or just those that belong to the current process (“local allocation”)?
- Generally, global approach works better, but not all page replacement algorithms can work that way (e.g., WSClock). Hybrid strategy — combine local approach with some way to vary processes’ allocations.

Slide 5

Thrashing and Load Control

- What happens if combined working sets of all processes don't fit into memory? "Thrashing". (See minute essay from last time!)
- What to do? temporarily "swap out" some processes, or other forms of "load control".

Slide 6

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 7

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’ system call for creating processes takes a different approach. Maybe that’s better!

Slide 8

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)? Seems attractive; are there potential problems? (To be continued . . .)

Minute Essay

- None — sign in.