Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on the Web; due a week from today.

Slide 2

Minute Essay From Last Lecture

- (Review question.)
- Could the problem be hardware-related? disk used for paging was bad? possible, if hardware problems led to poor performance (now they might — some errors are handled by disk hardware, where previously it just reported them).
- Is the problem that only one drive is being used for paging? or that maybe it's being used for something else too? again, maybe . . .
- Is the problem that the page replacement algorithm can't find a free frame? interesting, but probably not.
- Could the drive used for paging be too small? that would be a problem, but a performance problem?

Minute Essay From Last Lecture, Continued

- In my opinion simplest / most likely explanation is “thrashing” — system is so busy paging it has no time to do useful work.
- And it can still happen, though perhaps rarely!

Slide 3

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 4

Slide 5

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 6

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 7

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 8

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 9

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 10

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?

Slide 11

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. (Is there a downside to that?)
- How to make this happen . . . At link time, programs get “stub” versions of functions. References to real versions resolved at load time. Does this remind you of anything? and suggest a possible problem? how to fix?

Slide 12

Shared Libraries, Continued

- Downside of replacing shared libraries — may break applications that call their function. UNIX provides a way around this.
- 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”.

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

Minute Essay

- None — quiz.

Slide 14