Slide 1

Administrivia

- Reminder: Homework 5 due Monday.

Slide 2

Minute Essay From Last Lecture

- (Review question.)
- Most people didn't (IMO) come very close. Which disk(s) are being used for paging was significant only in tipping them off that it was being used a lot — which meant a lot of paging activity, and *that* was the source of the trouble.
- Whether it still happens, well, it's probably still possible, but the size of real memory makes it a lot less likely.

Slide 3

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.
- How is this useful? can compare different algorithms, and also determine if a given algorithm is a “stack algorithm” (more memory means fewer page faults).

Slide 4

Page Replacement Algorithms — Recap

- Nice summary in textbook (table at end of section 3.4).
- Tanenbaum says best choices are aging, WSClock. (Review these two from 11/13 notes.)
- Now move on to other issues to consider . . .

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.

Thrashing and Load Control

Slide 7

- 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".

One More Design Issue

Slide 8

- Page replacement algorithms as discussed all seem based on the idea that we let memory fill up, and then "steal" page frames as needed. Is that really the best way . . .
- An alternative — background process ("paging daemon") that tries to keep a supply of free page frames, or at least ones that can be stolen without needing to write out their contents. Can use algorithms similar to page replacement algorithms to do this.

Paging — Operating System Versus MMU

- Some aspects of paging are dealt with by hardware (MMU) — translation of program addresses to physical addresses, generation of page faults, setting of R and M bits.
- Other aspects need o/s involvement. What/when?

Slide 9

Paging — Operating System Involvement

- Process creation requires setting up page tables and other data structures. Process termination requires freeing them.
- Context switches require changing whatever the MMU uses to find the current page table.
- And of course it's the operating system that handles page faults!
- Some details . . .

Slide 10

Slide 11

Processing Memory References — MMU

- Does cache contain data for (virtual) address? If so, done.
- Does TLB contain matching page table entry? If so, generate physical address and send to memory bus.
- Does page table entry (in memory) say page is present? If so, put PTE in TLB and as above.
- If page table entry says page not present, generate page fault interrupt. Transfers control to interrupt handler.

Slide 12

Processing Memory References — Page Fault Interrupt Handler

- Is page on disk or invalid (based on entry in process table, or other o/s data structure)? If invalid, error — terminate process.
- Is there a free page frame? If not, choose one to steal. If it needs to be saved to disk, start I/O to do that. Update process table, PTE, etc., for "victim" process. Block process until I/O done.
- Start I/O to bring needed page in from swap space (or zero out new page). If I/O needed, block process until done.
- Update process table, etc., for process that caused the page fault, and restart it at instruction that generated page fault.

Slide 13

Minute Essay

- None — quiz.