Slide 1

Administrivia

- None really. But I'm hoping to finish grading Homework 2 and send out midsemester grade summaries soon.

Slide 2

Sidebar: Memory Management Within Processes

- What if we don't know before the program starts how much memory it will want? with very old languages, maybe not an issue, but with more modern ones it is.

  I.e., we might want to manage memory within a process's "address space" (range of possible program/virtual addresses).

- Typical scheme involves
  - Fixed-size allocation for code and any static data.
  - Two variable-size pieces ("heap" and "stack") for dynamically allocated data.
  - Notice — combined sizes of these pieces might be less than size of address space, maybe a lot less.

Paging — Review

Slide 3

- Recall basic ideas of paging:
  - Divide address spaces into pages, memory into page frames; allocate memory page (frame) by page (frame).
  - Use page tables (one per process) to keep track of things.
  - Use MMU to translate program (virtual) addresses into memory locations — using page table for current process. Generate “page fault” interrupt if impossible.
- Can be extended to provide a “virtual memory” abstraction, in which some pages are kept on disk . . .

Page Fault Interrupts

Slide 4

- We said MMU should generate a “page fault” interrupt for a page that’s not present in real memory. What happens then? It’s an interrupt, so . . .
- Control goes to an interrupt handler. What should it do? (Are there different possibilities for what caused the page faults?)

Slide 5

Page Fault Interrupts, Continued

- One possible cause — an address that's not valid. You know (sort of) what happens then . . .
- Another cause — an address that's valid, but the page is on disk rather than in real memory. So — do I/O to read it in. Where to put it? If there's a free page frame, choice is easy. What if there's not?

Slide 6

Finding A Free Frame — Page Replacement Algorithms

- Processing a page fault can involve finding a free page frame. Would be easy if the current set of processes aren't taking up all of main memory, but what if they are? Must steal a page frame from someone. How to choose one?
- Several ways to make choice (as with CPU scheduling) — "page replacement algorithms".
- "Good" algorithms are those that result in few page faults. (What happens if there are many page faults?)
- Choice usually constrained by what MMU provides (though that is influenced by what would help o/s designers).
- Many choices . . .

Slide 7

“Optimal” Algorithm

- Idea — if we know for each page when it will next be referenced, choose the one for which that’s the furthest away.
- Theoretically optimal, though can’t be implemented.
- Useful as a standard of comparison — run program once on simulator to collect data on page references, again to determine performance with this “algorithm”. (Not clear that this is really possible with multiprogramming.)

Slide 8

Sidebar: Page Table Entries, Revisited

- Recall — many architectures’ page table entries contain bits called “R (referenced) bit” and “M (modified) bit”.
- Idea is that these bits are set by hardware on any memory reference, and cleared by software (o/s) in some way that’s useful.

"Not Recently Used" Algorithm

Slide 9

- Idea — choose a page that hasn't been referenced/modified recently, hoping it won't be referenced again soon.
- Implementation — use page table's R and M bits, group pages into four classes:
  - $R = 0$, $M = 0$.
  - $R = 0$, $M = 1$.
  - $R = 1$, $M = 0$.
  - $R = 1$, $M = 1$.

  Choose page to replace at random from first non-empty class.
- How good is this? Easy to understand, reasonably efficient to implement, often gives adequate performance.

"First In, First Out" Algorithm

Slide 10

- Idea — remove page that's been there the longest.
- Implementation — keep a FIFO queue of pages in memory.
- How good is this? Easy to understand and implement, no MMU support needed, but could be very non-optimal.

Slide 11

"Second Chance" Algorithm

- Idea — modify FIFO algorithm so it only removes the oldest page if it looks inactive.
- Implementation — use page table's R and M bits, also keep FIFO queue. Choose page from head of FIFO queue, *but* if its R bit is set, just clear R bit and put page back on queue.
- Variant — "clock" algorithm (same idea, keeps pages in a circular queue).
- How good is this? Easy to understand and implement, probably better than FIFO.

Slide 12

"Least Recently Used" (LRU) Algorithm

- Idea — replace least-recently-used page, on the theory that pages heavily used in the recent past will be heavily used in the near future. (Usually true).
- Implementation:
  - Full implementation requires keeping list of pages ordered by time of reference. Must update this list on every memory reference.
  - Only practical with special hardware — e.g.:
    Build 64-bit counter C, incremented after each instruction (or cycle).
    On every memory reference, store C's value in PTE.
    To find LRU page, scan page table for smallest stored value of C.
    (Is 64 bits enough?)
- How good is this? Could be pretty good, but requires hardware we probably won't have.

Slide 13

"Not Frequently Used" (NFU) Algorithm

- Idea — simulate LRU in software.
- Implementation:
  - Define a counter for each PTE. Periodically ("every clock-tick interrupt") update counter for every PTE with R bit set.
  - Choose page with smallest counter.
- How good is this? Reasonable to implement, could be good, but counters track full history, which might not be a good predictor.

Slide 14

"Aging" Algorithm

- Idea — simulate LRU in software (like NFU), but give more weight to recent history.
- Implementation similar to NFU, but increment counters by shifting right and adding to *leftmost* bit — in effect, divide previous count by 2 and add bit for recent references.
- How good is this? Pretty good approximation to LRU, though a little crude, and limited by size of counter.

Slide 15

Minute Essay

- Another story from long ago: Once upon a time, a mainframe computer was running very slowly. The sysadmins were puzzled, until one of them noticed that one of the disk drives seemed to be very busy and asked “which disk are you using for paging?” The answer made everyone say “aha!” What was wrong (to make the system so slow)?
- Does anything like this still happen?

Slide 16

Minute Essay Answer

- The disk being used for paging was the one that was very busy. So, mostly likely the system was spending so much time paging (“thrashing”) that it wasn’t able to get anything else done. Usually this means that the system isn’t able to keep up with active processes’ demand for memory.
- This can indeed still be a problem — only a few years ago, with the Xenas trying to run both Eclipse and a Lewis simulation.