Slide 1

Administrivia

- (By e-mail.)

Slide 2

Swapping — Review/Recap

- One way to think of this:
- Each process has an address space consisting of “valid” pages.
- Collectively, these form the state “of memory”.
- Physical memory acts as a cache for the most-active parts of that. Less-active parts kept on disk (in “swap space”).
- Pages move between physical memory and swap space as needed.
- As with other caches, if physical memory fills up, need to make good choices about what to remove to make room for additions — “page replacement policies” (a.k.a. algorithms).

Slide 3

Optimal Page Algorithm

- First policy sounds — and is — impossible to implement: Choose page whose time of next access is furthest away.
- Why even consider this? as a standard of comparison — if come close, you have about as good a policy as possible, even if not great in absolute terms. "Hm!"?

Slide 4

"First In, First Out" Algorithm

- 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.
- (Aside: Probably seems intuitively obvious that more physical memory means fewer page faults, right? True for some algorithms but for this one not so for a few sequence of page references — "Belady's Anomaly".)

Slide 5

Random Algorithm

- Idea — choose page at random to remove.
- Implementation trivial, no?
- How good is this? Easy to understand and implement, no MMU support needed, *could* be almost as good as optimal, or could be very non-optimal.

Slide 6

History-Based Algorithms

- Most page-replacement algorithms try to exploit locality of time (remember from TLB replacement strategies).
- One family of algorithms bases decisions on keeping most-frequently-accessed pages in memory.
- Another tries to keep most-recently-accessed pages in memory.

Slide 7

Sidebar: MMU Additions

- Are you asking yourself how the O/S knows which pages have been used frequently or recently? Relies largely on hardware . . .
- Very common for PTEs to contain two extra bits per entry:
- One (“use bit” or “referenced bit”) set by MMU every time page is referenced, cleared by O/S when useful for algorithm (depends on algorithm).
- One (“dirty bit” or “modified bit”) set by MMU any time page is modified, cleared by O/S any time page is written to disk.

  Why do we care? Think about what should happen if this page is evicted from physical memory! If page exists on disk and hasn’t been changed, no need to write it again, but if it has?
- Following the textbook I used last year, I’ll call these R and M bits.

Slide 8

Least-Frequently-Used Algorithms

- (Not in textbook?)
- One way — track how many times each page has been referenced.
- To do this perfectly, need support for per-page counters managed by MMU.
- Can approximate using R bits using one (O/S-managed) counter per page:
- Periodically scan R bits and update counters for pages with R bit set. Then clear all R bits.
- When free page is needed, choose one with smallest counter.
- How good is this? Easy to understand, reasonably efficient to implement.

Slide 9

Least-Frequently-Used Algorithms, Continued

- Relatively simple and avoids evicting heavily-used pages.
- But weights all history the same, recent or not, which sounds not-right.
- How to fix that? “Aging”, so count is weighted:

  Update counters not by just adding one, but by dividing by two and dropping R bit in as first bit.

  Now counters heavily favor recently-used pages and break ties using historical data.
- How good is this? Easy to understand, reasonably efficient to implement, might give acceptable performance.

Slide 10

Least-Recently-Used Algorithms

- Idea of LRU algorithms is simple: Evict page least recently used.
- Performance can be quite good or not good at all (textbook describes a looping workload that causes big trouble).
- In any case — not really feasible:
- Keeping track of time of last use would require MMU support, and scanning all of them looking for time of least recent use probably not practical.
- So consider how to approximate . . .

Slide 11

"Not Recently Used" Algorithm

- (Not in textbook, but in last year's and appealingly simple.)
- Based on page table's R and M bits, with R bits cleared periodically. 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, might give adequate performance.

Slide 12

"Clock" Algorithm

- Key idea that improves performance is tracking only pages in memory:
- Think of all pages as forming a circular queues, like numbers on an analog clock, with pointer like one hand of clock.
- When a free frame is needed, look at page pointed to by hand. If R bit on, not a good candidate, so move on — but first clear R bit. If R bit off, choose it. (Note that this does stop eventually!)
- How good is this? Makes good choices, practical to implement.

Slide 13

“Clock” Algorithm — Variation (“WSClock”)

- Note that clock algorithm doesn’t take into account that some pages can be evicted right away while others have to be written out first.
- So before choosing bit with R=0, “WSClock’ checks its M bit. If 0, proceed; if not, start a disk write so it will be free next time, but for now move on.
- With this variation — authors of textbook I used last year say this is widely used.

Slide 14

Other Policy Choices

- Try to predict which pages will be needed (“prefetch”) or wait until needed (“demand paging”)? Should seem plausible that either has advantages.
- Write out pages one at a time or in clusters? Also either choice could be good.

Slide 15

Thrashing

- Term used in other textbooks is “working set” — per-process set of pages in current active use, which for acceptable performance all need to be in physical memory.
- If combined working sets all fit, all is well, and all processes run acceptably fast, though probably not as fast as they would if no swapping were needed.
- But if they don’t? I have another antique war story . . .

Slide 16

Minute Essay

- A story I heard a long time ago, from the mainframe days: One afternoon, the mainframe seemed to be unusually slow. The sysadmins congregated in the computer room to try to figure out why. They were puzzled, until one of them noticed something about one of the disk drives . . . What was the problem?
- Can something similar happen nowadays?
- (Pause the video, think, write your best guess.)

Slide 17

Minute Essay Answer

- It was so busy swapping it didn't have time to do anything else!
- Yes. Try running a modern tool such as Eclipse on a system with very limited memory.