Slide 1

Administrivia

- Homework 3 on Web (except for programming problem, to be added later today). Due next Friday.

Slide 2

Page Replacement Algorithms, Continued

- Key idea — if all page frames (slots in main memory) are in use, but we need to bring in a page from swap space on disk, must "steal" a page frame. Which one?
- Many strategies, varying degrees of practicality.
- A few more to discuss, based on idea that each process has a "working set" of pages that need to be in memory.

  (Minute essay question from previous lecture — what happens if combined sizes, plus pages for o/s, exceeds main memory?)

Slide 3

"Working Set" Algorithm

- Idea — steal / replace page not in recent working set. Define working set by looking back τ time units (w.r.t. process's virtual time). Value of τ is a tuning parameter, to be set by o/s designer or sysadmin.
- Implementation:
  - For each entry in page table, keep track of time of last reference.
  - When we need to choose a page to replace, scan through page table and for each entry:

    If R bit is set, update time of last reference.

    Compute time elapsed since last use. If more than τ, page can be replaced.
  - If we don't find a page to replace that way, pick the one with oldest time of last use. If a tie, pick at random.
- How good is this? Good, but could be slow.

Slide 4

"WSClock" Algorithm

- Idea — efficient-to-implement variation of previous algorithm, based on circular list of pages-in-memory for process.
- Implementation — like previous algorithm, but when we need to pick a page to replace, go around the circle and:
  - If R=1, update time of last use. Compute time since last use.
  - If time since last use is more than τ and M=1, schedule I/O to write this page out (so it can maybe be replaced next time — M bit will be cleared when I/O completes). No need to block yet, though.
  - If time since last use is more than τ and M=0, replace this page.

  The idea is to go around the circle until we find a page to replace, then stop. (If we get all the way around the circle, we'll pick some page with M=0.)
- How good is this? Makes good choices, practical to implement, apparently widely used in practice.

Slide 5

Review — Page Replacement Algorithms

- Nice summary in textbook, table on p. 228.
- Author says best choices are aging, WSClock.

Slide 6

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 7

Paging — Other Design Issues

- 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.

- What happens if combined working sets of all processes don't fit into memory? "Thrashing".

  What to do? temporarily "swap out" some processes, or other forms of "load control".

- Maintaining a supply of free frames — desirable, could do by having a "paging daemon" in background.

Slide 8

Paging — Other Hardware Issues

- What if page to be replaced is waiting for I/O? probably trouble if we replace it anyway.
- One solution — allow pages to be "locked".
- Another solution — do all I/O to o/s pages, then move to user pages.

Slide 9

Minute Essay

- Consider the following code:

```
double a[BIGNUM][BIGNUM];
for (int i = 0; i < BIGNUM; ++i)
  for (int j = 0; j < BIGNUM; ++j)
    a[i][j] = i+j;
```

Reversing the order of the loops can have a big effect on execution time. Why? (Actually there are two possible explanations, depending on how big BIGNUM is.)

Slide 10

Minute Essay Answer

- Accessing all elements of a large data structure is faster if contiguous elements are accessed one after another (rather than skipping around). Depending on the size of the array, skipping around could produce extra paging or more cache misses. Either would reduce performance.