Paging Algorithms — Last Time

On a page fault, we need to choose a page to kick out.

Random: amazingly, this algorithm works pretty well.

FIFO: First-In, First-Out. Throw out the oldest page. Simple to implement, but the OS can easily throw out a page that is being accessed frequently.

MIN: (a.k.a. OPT) Look into the future and throw out the page that will be accessed farthest in the future. Provably optimal [Belady'66], but not feasible to implement.

LRU: Least Recently Used. Approximation of MIN that works well if the recent past is a good predictor of the future. Throw out the page that has not been used in the longest time. Too expensive to implement.

LRU Approximations:

Hardware support: reference bit(s). Each access to a page causes the hardware to set the reference bit to '1'.

Second Chance: Keep pages in a circular queue. If reference bit is '1', set it to '0'. If reference bit is '0' kick out the page.

Enhanced Second Chance: (Today) using more bits and modification status to improve LRU performance.

Multiprogramming: (Today) dividing pages among distinct processes

Enhanced Second Chance

It is cheaper to replace a page that has not been written since it need not be written back to disk, thus the OS can give preference to paging out un-modified pages.

- Hardware keeps a *modify* bit:

  '1': page is modified (different from the copy on disk)

  '0': page is the same as the copy on disk

- the reference bit and modify bit form a pair (r,m) where

  1. (0,0) neither recently used nor modified - replace this page!
  2. (0,1) not recently used but modified - not as good to replace, since the OS must write out this page, but it might not be needed anymore.
  3. (1,0) recently used and unmodified - probably will be used again soon, but OS need not write it out before replacing it
  4. (1,1) recently used and modified - probably will be used again soon and the OS must write it out before replacing it

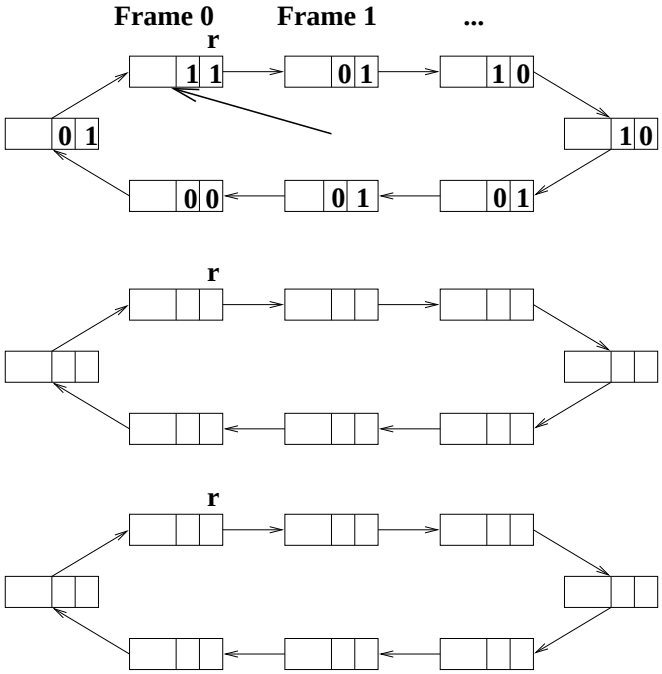
- On a page fault, the OS searches for the first page in the lowest nonempty class.

Enhanced Second Chance

The OS goes around at most three times searching for the (0,0) class.

1. If the OS finds (0,0), it replaces that page.
2. If the OS finds (0,1), it initiates an I/O to write out that page, locks the page in memory until the I/O completes, clears the modified bit, and continues its search in parallel with the I/O.
3. For pages with the reference bit set, the reference bit is cleared.
4. If the hand goes completely around once, there was no (0,0) page. On the second pass, a page that was originally (0,1) or (1,0) might have been changed to (0,0). If the page is being written out, it waits for the I/O to complete and then removes the page. A (0,1) page is treated as on the first pass. By the third pass, all the pages will be at (0,0).

Clock Example



Replacement Policies for Multiprogramming

Thrashing: the memory is over-committed and pages are tossed out while they are still in use. Instead of the illusion of RAM speed, memory references take disk time since many memory references require a page to be faulted in. The result is a serious and very noticeable loss of performance. This problem was severe in early systems, e.g., the VAX.

What can we do in a multiprogrammed environment to limit thrashing?

Proportional allocation: allocate more page frames to large processes.

Global replacement: put all pages from all processes in one pool so that the physical memory associated with a process can grow

Advantages: Flexible, adjusts to divergent process needs

Disadvantages: Thrashing might become even more likely. Why?

Replacement Policies for Multiprogramming

Per-process replacement: Each process has its own pool of pages.

- Run only groups of processes that fit in memory, and kick out the rest.
- How do we figure out how many pages a process needs, i.e., its working set size?
- The working set is informally the pages the process is using right now. More formally, it is the set of all pages that a process referenced in the past T seconds.
- How does the OS pick T?

  1 page fault = 10msec

  10msec = 2 million instructions

  So T needs to be a whole lot bigger than 2 million instructions.

  What happens if T is too small? too big?

Replacement Policies for Multiprogramming

Per-process replacement:

- Working sets are expensive to compute.
- Can track page fault frequency of each process.
- If the page fault frequency is above some threshold, give it more page frames.
- If the page fault frequency is below a second threshold, take away some page frame.
- Goal: the system-wide mean time between page faults should be equal to the time it takes to handle a page fault.
- Might be necessary to suspend a process until overall memory demands decrease.

Per-process replacement:

Advantages: Thrashing less likely as process only competes with itself. More consistent performance independent of system load.

Disadvantages: The OS has to figure out how many pages to give each process and if the working set size grows dynamically adjust its allocation.

Page Sizes

Page sizes have been growing slowly but steadily. Why?

Reasons for small pages:

- More effective memory use.
- Higher degree of multiprogramming possible.

Reasons for large pages:

- Smaller page tables
- Reduced I/O time (it takes much longer to move the disk head to the start of a block than to transfer the data once the position is set)
- Fewer page faults (for processes that exhibit locality of references)

Page sizes are growing because:

- Physical memory is cheap. As a result, page tables could get huge with small pages. Also, internal fragmentation is less of a concern with abundant memory.
- CPU speed is increasing faster than disk speed. As a result, page faults result in a larger slow down than they used to. Reducing the number of page faults is critical to performance.

Summary of Memory Management

Contiguous allocation - simple but not flexible enough.

Segmentation - better, but still not flexible enough.

Paging - very flexible, enables sharing, does not match process structure.

Segmented paging - very flexible, matches process structure, and enables sharing at the right granularity.

Virtual memory and Demand Paging - enable the process memory size to exceed the physical memory size and uses memory more efficiently.

Page replacement algorithms - how to pick pages to replace.

Next Time

User level memory management: how to manage the user heap.