

CS 537

Lecture 8

Page Replacement Policies

Michael Swift

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

1

Evicting the best page

- The goal of the page replacement algorithm:
 - reduce fault rate by selecting best victim page to remove
 - the best page to evict is one that will never be touched again
 - as process will never again fault on it
 - “never” is a long time
 - Belady’s proof: evicting the page that won’t be used for the longest period of time minimizes page fault rate
- Rest of this lecture:
 - survey a bunch of replacement algorithms

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

2

#1: Belady’s Algorithm

- Pick the page that won’t be used for longest time in future
 - Provably optimal lowest fault rate (remember SJF?)
 - Why?
 - Problem: impossible to predict future
- Why is Belady’s algorithm useful?
 - as a yardstick to compare other algorithms to optimal
 - if Belady’s isn’t much better than yours, yours is pretty good
- Is there a lower bound?
 - unfortunately, lower bound depends on workload
 - but, random replacement is pretty bad

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

3

#2: FIFO

- FIFO is obvious, and simple to implement
 - when you page in something, put in on tail of list
 - on eviction, throw away page on head of list
- Why might this be good?
 - maybe the one brought in longest ago is not being used
- Why might this be bad?
 - then again, maybe it is being used
 - have absolutely no information either way
- FIFO suffers from **Belady’s Anomaly**
 - fault rate might **increase** when algorithm is given more physical memory
 - a very bad property

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

4

Example of Belady's Anomaly

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4	3 pages
Newest Page	3	2	1	0	3	2	4	4	4	1	0	0	
		3	2	1	0	3	2	2	2	4	1	1	
Oldest Page			3	2	1	0	3	3	3	2	4	4	
Page Requests	3	2	1	0	3	2	4	3	2	1	0	4	4 pages
Newest Page	3	2	1	0	0	0	4	3	2	1	0	4	
		3	2	1	1	1	0	4	3	2	1	0	
			3	2	2	2	1	0	4	3	2	1	
Oldest Page				3	3	3	2	1	0	4	3	2	

(red italics indicates page fault)

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

2/26/13

5

#3: Least Recently Used (LRU)

- LRU uses reference information to make a more informed replacement decision
 - idea: past experience gives us a guess of future behavior
 - on replacement, evict the page that hasn't been used for the longest amount of time
 - LRU looks at the past, Belady's wants to look at future
 - when does LRU do well?
 - when does it suck?
- Implementation
 - to be perfect, must grab a timestamp on every memory reference and put it in the PTE (way too \$\$)
 - so, we need an approximation...

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

2/26/13

6

Approximating LRU

- Many approximations, all use the PTE reference bit
 - keep a counter for each page
 - at some regular interval, for each page, do:
 - if ref bit = 0, increment the counter (hasn't been used)
 - if ref bit = 1, zero the counter (has been used)
 - regardless, zero ref bit
 - the counter will contain the # of intervals since the last reference to the page
 - page with largest counter is least recently used
- Some architectures don't have PTE reference bits
 - can simulate reference bit using the valid bit to induce faults
 - hack, hack, hack

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

2/26/13

7

#4: LRU Clock

- AKA Not Recently Used (NRU) or Second Chance
 - replace page that is "old enough"
- Arrange all physical page frames in a big circle (clock)
 - just a circular linked list
 - a "clock hand" is used to select a good LRU candidate
 - sweep through the pages in circular order like a clock
 - if ref bit is off, it hasn't been used recently, we have a victim
 - so, what is minimum "age" if ref bit is off?
 - if the ref bit is on, turn it off and go to next page
 - arm moves quickly when pages are needed
 - low overhead if have plenty of memory
- if memory is large, "accuracy" of information degrades
 - add more hands to fix
- SHOW EXAMPLE!

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

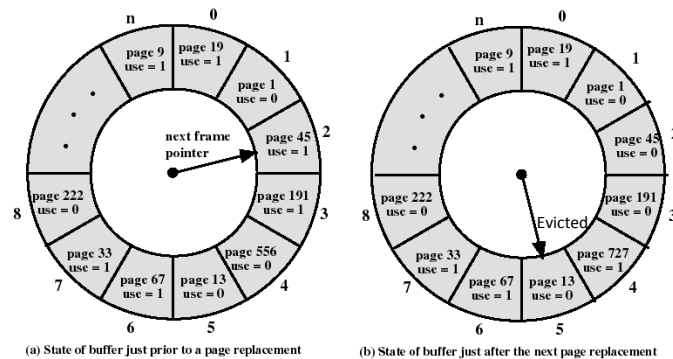
2/26/13

8

Clock page replacement

- Circular list instead of queue
- Clock hand points to oldest page
- If (Referenced==0) then
 - Page is unused so replace it
- Else
 - Clear Referenced
 - Advance clock hand
- (very similar to second chance – queue instead of list)

The Clock Policy: an example



Another Problem: allocation of frames

- In a multiprogramming system, we need a way to allocate physical memory to competing processes
 - what if a victim page belongs to another process?
 - family of replacement algorithms that takes this into account
- Fixed space algorithms
 - each process is given a limit of pages it can use
 - when it reaches its limit, it replaces from its own pages
 - **local replacement**: some process may do well, others suffer
- Variable space algorithms
 - processes' set of pages grows and shrinks dynamically
 - **global replacement**: one process can ruin it for the rest
 - linux uses global replacement

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

11

#5: 2nd Chance FIFO

- LRU Clock is a **global** algorithm
 - It looks at all physical pages, from all processes
 - Every process gets its memory taken away gradually
- Local algorithms: run page replacement separately for each process
- 2nd Chance FIFO:
 - Maintain 2 FIFO queues per process
 - On first access, pages go at end of queue 1
 - When the drop off queue 1, page are invalidated and move to queue 2
 - When they drop off queue 2, they are replaced
 - If they are accessed in queue 2, they are put back on queue 1
- Options:
 - Move to queue 1 immediately when referenced: mark "invalid" when on queue 2
 - Move to queue 2 when about to be evicted: looks like clock
- Comparison to LRU clock:
 - Per-process, not whole machine
 - No scanning
 - Replacement order is FIFO, not PFN
 - Used in Windows NT, VMS

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

12

Second chance page replacement

- Inspect R bit of oldest page
 - Recall: R bits are set when page is referenced (read or write); periodically (after k clock interrupts), R bits are cleared.
 - If $R=0$ then
 - page is old & unused so replace it
 - Else
 - Clear R bit
 - Move page from head to tail of FIFO
 - (treating it as a newly loaded page)
 - Try a different page

Second chance page replacement

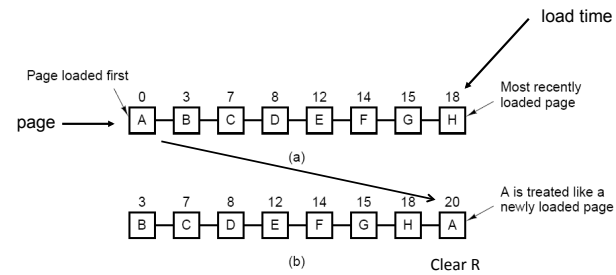


Fig. 4-16. Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and A has its R bit set. The numbers above the pages are their loading times.

The Working Set Strategy

- Is a variable-allocation method with local scope based on the assumption of locality of references
- The working set for a process at time t , $W(D,t)$, is the set of pages that have been referenced in the last D virtual time units
 - virtual time = time elapsed while the process was in execution (eg: number of instructions executed)
 - D is a window of time
 - at any t , $|W(D,t)|$ is non decreasing with D
 - $W(D,t)$ is an approximation of the program's locality

15

The Working Set Strategy

- The working set of a process first grows when it starts executing
- then stabilizes by the principle of locality
- it grows again when the process enters a new locality (transition period)
 - up to a point where the working set contains pages from two localities
- then decreases after a sufficient long time spent in the new locality

16

#6: Working Set Size

- The working set size changes with program locality
 - during periods of poor locality, more pages are referenced
 - within that period of time, the working set size is larger
- Intuitively, working set must be in memory, otherwise you'll experience heavy faulting (thrashing)
 - when people ask "How much memory does Firefox need?", really they are asking "what is Firefox average (or worst case) working set size?"
- Hypothetical algorithm:
 - associate parameter "w" with each process = # of unique pages referenced in the last "t" ms that it executed
 - only allow a process to start if it's "w", when added to all other processes, still fits in memory
 - use a local replacement algorithm within each process (e.g. clock, 2nd chance FIFO)

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

17

The Working Set Strategy

- the working set concept suggest the following strategy to determine the resident set size
 - Monitor the working set for each process
 - Periodically remove from the resident set of a process those pages that are not in the working set
 - When the resident set of a process is smaller than its working set, allocate more frames to it
 - If not enough free frames are available, suspend the process (until more frames are available)
 - ie: a process may execute only if its working set is in main memory

18

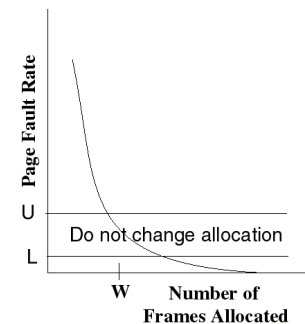
The Working Set Strategy

- Practical problems with this working set strategy
 - measurement of the working set for each process is impractical
 - necessary to time stamp the referenced page at every memory reference
 - necessary to maintain a time-ordered queue of referenced pages for each process
 - the optimal value for D is unknown and time varying
- Solution: rather than monitor the working set, monitor the page fault rate!

19

The Page-Fault Frequency Strategy

- Define an upper bound U and lower bound L for page fault rates
- Allocate more frames to a process if fault rate is higher than U
- Allocate less frames if fault rate is $< L$
- The resident set size should be close to the working set size W
- We suspend the process if the PFF $> U$ and no more free frames are available



20

Summary

- demand paging
 - start with no physical pages mapped, load them in on demand
- page replacement algorithms
 - #1: Belady's – optimal, but unrealizable
 - #2: Fifo – replace page loaded furthest in past
 - #3: LRU – replace page referenced furthest in past
 - approximate using PTE reference bit
 - #4: LRU Clock – replace page that is “old enough”
 - #5: 2nd Chance FIFO – replace local page that is “old enough”
 - #6: working set – keep set of pages in memory that induces the minimal fault rate
- local vs. global replacement
 - should processes be allowed to evict each other's pages?

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

21

Thrashing

- What the OS does if page replacement algo's fail
 - happens if most of the time is spent by an OS paging data back and forth from disk
 - no time is spent doing useful work
 - the system is overcommitted
 - no idea which pages should be in memory to reduced faults
 - could be that there just isn't enough physical memory for all processes
 - solutions?
- Yields some insight into systems research[ers]
 - if system has too much memory
 - page replacement algorithm doesn't matter (overprovisioning)
 - if system has too little memory
 - page replacement algorithm doesn't matter (overcommitted)
 - problem is only interesting on the border between overprovisioned and overcommitted
 - many research papers live here, but not many real systems do...

2/26/13

© 2004-2007 Ed Lazowska, Hank Levy,
Andrea and Remzi Arpaci-Dusseau, Michael
Swift

22