Enes Yazgan
Ian Feekes
CMPS111 assignment3: page replacement

Results:

As specified in the assignment pdf, the parameters our benchmarks were measuring are the following: pages scanned, the number of active to inactive pages, the number of inactive to cache/free, and the number of pages queued for flush. Each of these parameters can be tracked and calculated within the pageout_scan() method of vm_pageout.c for both the default CLOCK and our modified FIFO cases.

We are fairly certain that our measurement and logging of each of the parameters is accurate as we have put a large amount of time in scrutinizing the code and verifying the behavior of pageout_scan().

Since we ran two separate memory-hogging tests for both of the cases, for each of the parameters the left chart will represent the test using a single hog taking up very large amounts of memory, and the right chart will represent the test using several hogs taking up their own equal chunks of memory. (Test 1 and Test 2 in the design doc)

Pages Scanned:

The amount of pages scanned for each scan differs greatly between the default FreeBSD CLOCK virtual memory replacement algorithm and our own FIFO algorithm we implemented.

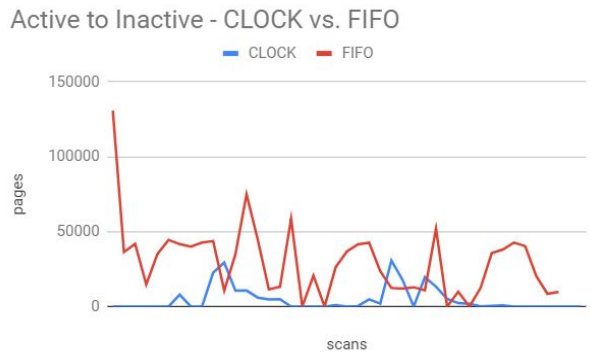
With the CLOCK algorithm, the pages scanned fluctuates in rapid spikes of pages scanned, whereas our FIFO algorithm remains relatively consistent, but also fairly low with its rate of pages scanned in comparison - resembling the behavior of the CLOCK method without the large spikes in both test cases.

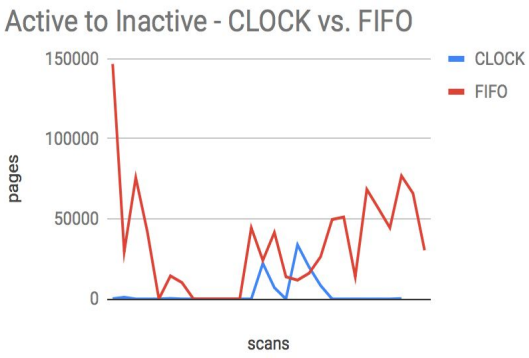
Some reasoning behind the difference in behavior as seen above may lie in the fact that the efficiency of the CLOCK algorithm in comparison with our FIFO algorithm allows a significantly larger amount of pages to be scanned in bursts, whereas the FIFO algorithm is consistently slow-performing in its capacity to scan pages. Also, with our FIFO queues, significantly more pages get flushed (as you can see several graphs down), so they no longer

will be present within the inactive queue for scanning, which illustrates the behavior you see above.

We make this assertion because the algorithm for keeping track of the number of pages scanned is relatively trivial, and we are very confident with it: There are two for loops in the pageout_scan() method, one for the inactive queue and one for the active queue respectively. The number of pages scanned gets incremented at the beginning of each iteration of both of the two loops. This way each page in the active queue and in the inactive queue that has been scanned will be added into our variable that we use to track this.

Active to Inactive:

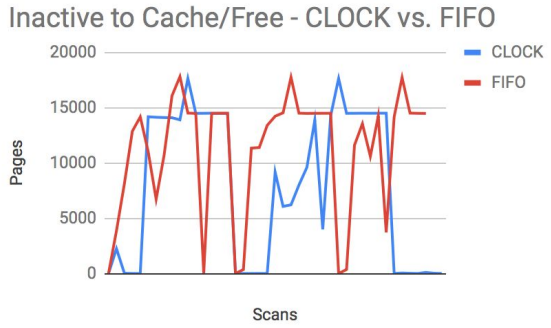


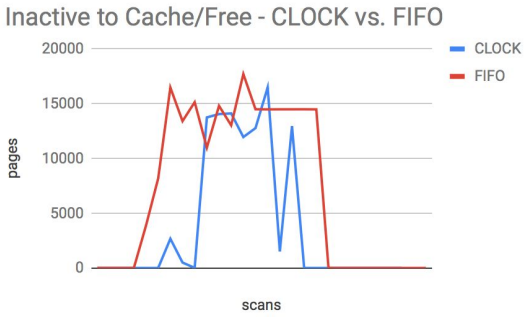


The main difference to note is that throughout the execution of both of the memory hogging programs, our FIFO queue consistently dropps many more pages from active to inactive.

This can be attributed to the fact that due to the behavior of the FIFO algorithm, during a page fault, the head of the active queue always will be thrown into the tail of the inactive queue, whereas with the CLOCK algorithm, virtual memory pages will only be transferred from active to inactive via a call to *vm_page_deactivate(m)* if the page's activity counter has reached 0, which can take a relatively long time with a large number of scans to occur.

Inactive to Cache/Free:



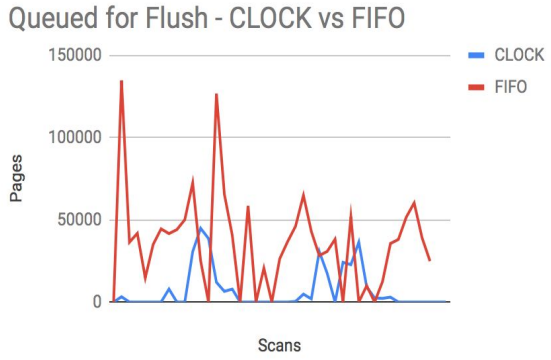


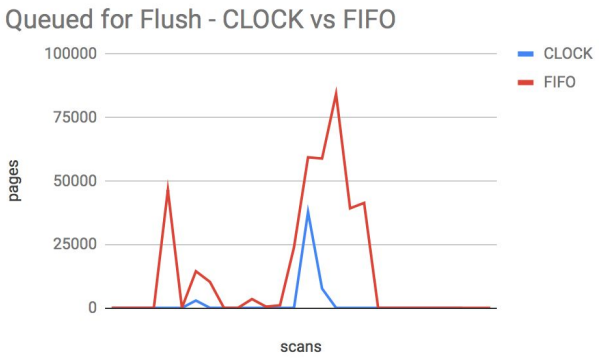
The statistics for inactive to cache/free between CLOCK and FIFO resemble each other much more similarly than in the graphs for the previous parameters measured in this assignment.

Throughout both tests, while the maximal and minimal quantities of pages moved are quite nearly the same, our FIFO algorithm seems to begin to move these pages for longer periods of time, of for more scans that seem to bound scans that represent times where page-faulting occurs.

This may be so because the FIFO queues we implemented consistently will have larger inactive queues from sending more pages at the head of the active queues into the inactive queues. Because of this, since there are more pages within the inactive queues, there are more pages that the *pageout_scan()* method determines should be moved to cache or even freed.

Queued for Flush:





This parameter illustrates the most significant dissimilarities between the FreeBSD CLOCK algorithm and our own FIFO algorithm we implemented.

Our FIFO algorithm consistently queues significantly more pages to flush. This may be attributed to the same source mentioned in the previous graph: since our FIFO virtual memory replacement algorithm has way more pages sent to the inactive queue, it makes sense that while scanning, the method would determine there are more pages that should be queued for flush.

Sources of Error:

Our graphs do not resemble our expected contrast between the CLOCK and FIFO virtual memory replacement algorithms, and as such it makes us believe that we have several sources of error to be attributed.

For our FIFO queue, it moves an active virtual memory page to the inactive queue whenever it determines that:

1) There is availability in the inactive queue
2) The page is at the head of the active queue

We believe that due to this, throughout the for loop (due to the cyclical nature of the preexisting queues), each active page that essentially isn't laundered is sent to the inactive queue (where it will be processed to be sent back upon the next scan).

When thinking about this, I modified the behavior to make the active queue non-cyclical, so that when it removes the head into the inactive queue (as per specifications of FIFO behavior), this only happens once per scan. This led to each scan only one page being moved

from active to inactive (as expected), and benchmarking tests that were so slow that we determined that this approach must be wrong.