Assignment 3 Design Document

Charles Tzou,David Lung, Payton Javete
CruzID: chtzou, dflung, pjavete

1 Goal

The goal of this assignment was to change the current paging algorithm in FreeBSD to a FIFO algorithm

2 Assumptions

The program should only work with FreeBSD and is not meant to work as a replacement for the Mac or Windows kernel.

3 Design

Our approach to implementing the FIFO algorithm was to keep as much of the original code as possible. We mainly commented out sections of vm_pageout.c that put pages back onto queues that weren't necessary for FIFO (e.g. putting a page back onto active queue from inactive queue). The only things that we added to the code were benchmarks.

4 Explanation

Line numbers here are based on the original location of that code in the FreeBSD kernel (before any edits)

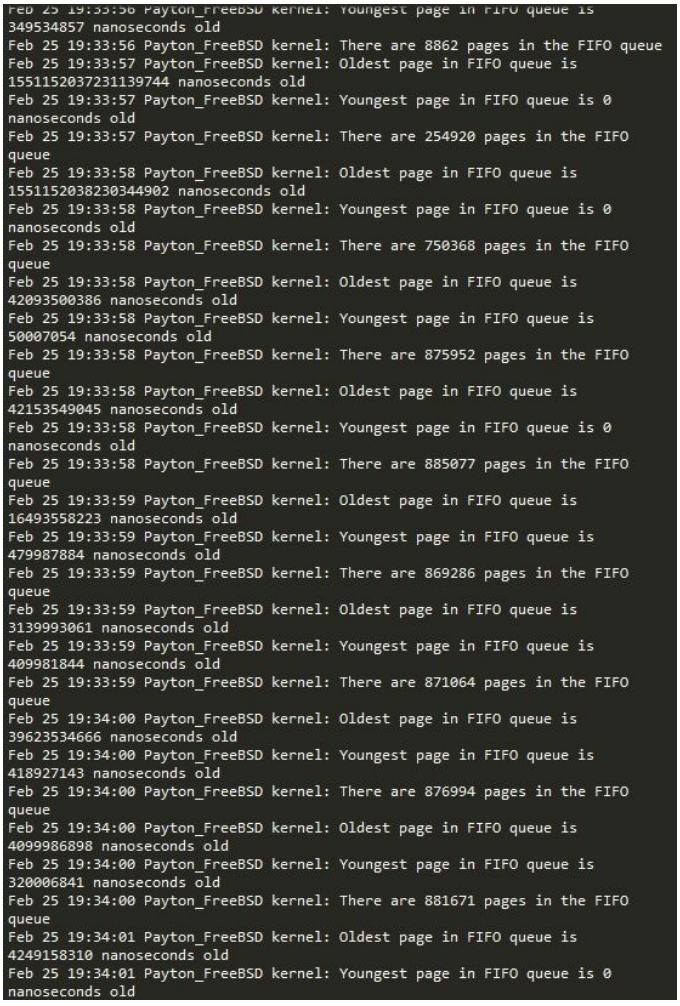
1. Commenting out line 1222 to line 1255 (in vm_pageout_scan). This code is to requeue into the inactive queue or to reactivate the page and place it back in the active queue. Also commented out the drop_page label on line 1284.
    a. Compiled and booted successfully
2. Commenting out line 762 to line 800 (in vm_pageout_launder). This code is to requeue into the launder queue or to reactivate the page and place it back in the active queue. Also commented out the requeue_page label on line 834.
    a. Compiled and booted successfully
3. Commenting out line 1393 to line 1466 (in vm_pageout_scan). This code is to determine where to move an active page based on its recent activity (i.e. number of requests). We also added our own functionality to just start removing whatever is first in the active queue (since it is the oldest) until we have fulfilled the inactive queue shortage.
    a. Compiled and booted successfully
4. Adding at the start of pageout_scan (line 1080 in pageout.c) print functions to print the statistics mentioned below for the benchmark. Also adding two fields in struct vm_page (line 128 in page.h) called time_t timestamp_sec and long timestamp.nsec to keep track of when the page was activated. Finally, adding code to vm_page_activate (line 2727 in

page.c) to set that timestamp fields when the page is activated. Included sys/time.h to use getnanotime in page.c. Already included in pageout.c.
    a. Compiled and booted successfully
        i. Was printing every second so put it in an if statement that checks if there is a page shortage and only prints the statistics if page_shortage > 0 (this was removed eventually)
5. Adding initializers for timestamp_sec and timestamp_nsec in vm_page_init_page (line 415 in page.c) that set them to the current time.
    a. Compiled and booted successfully
6. Changed pageout_update_period to 1 (line 176 in pageout.c) to see if we can change the frequency that pageout scan runs.
    a. Compiled and booted successfully
        i. Seemed to run faster, may need to reduce it below 1 to see effects better
7. Changed act_scan_laundry_weight (line 194 in pageout.c) to 1. Added check to see if oldest page is the marker page. Changed where timestamps are set in vm_page_activate. (page.c)
    a. Compiled and booted successfully
        i. The oldest page is the marker page!
8. Added code to make sure the oldest page we printed was not the marker page (i.e. we ignore the marker pages).
    a. Compiled and booted successfully
        i. It worked!
        ii. Occasionally did epoch time for youngest page, may need to check if that is a marker as well
9. Added back setting timestamps in vm_page_init_page in page.c (line 415) after it crashed while just running normally. Additionally added checks in while loops that check if a page is a marker to also check if the next page that the while loop will check is NULL.
    a. Compiled and booted successfully
        i. Started to use stress package, somewhere between 3594 and 4000 seems good for the part before M in the command stress -c 2 -i 1 -m 1 --vm-bytes 3594M -t 10s
10. Removed setting the timestamps in vm_page_init_page in page.c (line 415). Also cleaned up while loops to find a non marker page.
    a. Compiled and booted successfully
11. Stress the memory of the system with code
    a. Using a stress package
    b. This is the command we use to stress the memory:
        i. stress -c 2 -i 1 -m 1 --vm-bytes 3594M -t 10s
        ii. Can increase the number before M

12. Changed vm_pageout_update_period to be equal to 10 on line 1833 in pageout.c. This causes the active queue scan (for loop) in pageout_scan to run more times since min_scan will be larger (since it is divided by vm_pagout_update_period).
    a. Compiled and booted successfully
        i. No observable behavior, but it is being set to 10.

4 Benchmarking

The issue of the FIFO algorithm is the fact that pages that are old might be used in the next clock cycle. However, since FIFO means that the oldest pages are paged out first, the processes that will need to use the page will have to fetch the page from memory again, triggering a page fault. As a result, this will lead to a greater overhead since there will be more page faults. This is why the default algorithm in FreeBSD is a Second Chance algorithm. This allows older pages to get a "second chance" before being paged out so it lowers the number of page faults.



When we run our the program with our benchmark, the youngest page's age is always younger than the oldest page, and sometimes equals 0. This means that the page was probably just paged in due to a page fault since we removed the places where pages are placed in active queue.

THINGS WE TRIED

1. Changed vm_page_activate to vm_page_deactivate (line 535 in pageout.c) in order to stop anything from going back to active queue.
    a. Compiled and booted successfully
        i. Still oldest page has time 0 and has holds
        ii. It killed the VM after running benchmark
        iii. Reverted to activate
2. Commented out line 534 to 536 (in pageout.c) which was in pageout_flush function. This would reactivate the page if it couldn't be paged out. Instead for our FIFO queue we free the page.
    a. Compiled and booted successfully
        i. During benchmark had fatal error when trying to pageout busy page
        ii. reverted
3. Same as 2 except we added an if statement to check if that page is busy, on hold, or wired. Also added a check to see if youngest page is a marker page.
    a. Compiled and booted successfully
        i. Crashed the same way as 6
        ii. reverted
4. Commenting out like in 2 but not adding anything else.
    a. Compiled and booted successfully
        i. Crashed the same way as 6
        ii. Reverted
5. Changed printing statistics to be inside vm_page_activate in page.c (line 2727) before we enqueue the new page. Also removed setting the timestamps in vm_page_init_page in page.c (line 415).
    a. Had to redefine several variables
    b. Compiled but did not boot, no logs to check
    c. Reverted
    d. Also took out if statement around print functions
6. Setting vm_pageout_update_period to 5 on line 161 in pageout.c.
    a. Compiled and booted successfully
        i. Crashed after running stress test
        ii. Reverted

NOTES:

Might need to keep track of addl_page_shortage = 0;

Atomic_readandclear_int (i think) is an atomic process that reads and clears and int value that we pass in (http://fxr.watson.org/fxr/source/powerpc/include/atomic.h?v=FREEBSD11#L422)

vm_paging_target() returns number of pages needed - number of pages free
http://fxr.watson.org/fxr/source/sys/vmmeter.h?v=FREEBSD11#L168

Probably need to start modifying at line 1120

PCPU_INC referenced in pcpu.h might be deciding what gets used by cpu

In the for loop part of pageout_scan in pageout.c we need to make sure that it is ordered from oldest to newest so that it removes the oldest inactive pages first.

*****LINE 1337 SHOULD BE THE BIG STARTING POINT*****
Line 1344 - 1346 might want to remove since it deals with active queue

Lowmem_period = 10 might have to do with the fact that we currently scan every 10 sec???

*****LINE 1429 change moving from active queue being based on recent references to that page, being based on the age of the page
~~May need to add a field to the page struct that keeps track of its age~~
Instead just add new pages to the end of the active queue and when we need more pages start by removing from the front until we have enough

Might need to comment out things starting from line 757 in vm_pageout_launder

```
#define VM_LAUNDER_RATE      10
```

When running benchmark test eventually an error is printed (swap_pager_getswapspace(32): failed) after which the oldest page in the FIFO queue (first page in the inactive queue) prints out the linux current time

since for some reason the timestamps are getting set back to 0 for that page. It could mean one of these things:

1. There is a page with timestamp 0 stuck at the front of the inactive queue
2. All things getting recycled back into the inactive queue somehow have timestamps set to 0
    a. This either happens when it is brought back from the launder queue or when its dequeued from the active queue

For some reason there occasionally is a page that comes through that does not have timestamp 0 and is the first thing in the inactive queue even after the error and after its been consistently printing the large nanosecond times that are the current linux time. This could be due to a few things, two ideas of which are below:

1. It somehow has recycled stuff from the launder queue back to the inactive queue and cannot free it for some reason and keeps cycling it through, but occasionally something from the active queue slips in.
2. Same as above except its the active queue sending stuff to the inactive queue that cannot be freed and the launder queue only occasionally slips something in.
3. Could also be that the pages are being freed fine but the timestamps are getting set to 0 somehow