Assignment 3 Design Document

March 3, 2017

Nikola Panayotov (npanayot@ucsc.edu)
Marko Jerkovic (mjerkovi@ucsc.edu)
Edwin Ramirez (edalrami@ucsc.edu)
Jeremiah Liou (jliou@ucsc.edu)

1 Abstract

In this assignment, the main focus was to experiment with the pageout daemon by modifying how FreeBSD scans page queues and where it moves pages based on their activity. The paging algorithm that we were required to implement was the FAT CHANCE algorithm which changes how the activity count is decrimented and how the pages are placed on a queue. The effect of the FAT CHANCE algorithm causes the computer to have more activity on scanning the pages and moving them. After implementing FAT CHANCE algorithm, the logged information seems to match with what was expected.

2 Implementation

The first modification we made was lowering the memory on the virtual machine. Originally, the installation we were testing on had 4096 MB of memory. With this much memory, paging would be rare and the comparison between the default paging algorithm and fat chance paging algorithm would not be comprehensive. Instead, the memory was decreased to 128 MB, resulting in more frequent paging calls. The following files (bold) were modified in order to implement the actual fat chance paging algorithm, as well as log statistics about the implementations:[1]

2.1 Logging

vm_pageout.c:
vm_pageout_scan:

[1]The pdf says to modify the files in sys/vm, but the files in usr/src/sys/vm were modified in this assignment.

In order to log information, log() was used, along with multiple counter variables in the scanning function. The first set of information we logged was the number of pages in each queue. The FreeBSD implementation contains an array of 2 queues inside of each vm_domain struct: a page queue PQ_ACTIVE and page queue PQ_INACTIVE. Each of these page queues contains a value of the page count. In the beginning of every call to scan, these two values are recorded to the debug.log file.

vm_pageout_scan first loops through the inactive queue, looking for pages to be cached or freed. A flag is set to determine if once the loop was entered, if it was the first time scanning the inactive queue on that run of the vm_pageout_scan. If so, queues scanned count is incremented. In every iteration of this for-loop, the count of inactive pages scanned is incremented. Each time a page is invalid and vm_page_free was called, the freed pages counter is incremented, and similarly, each time a page's dirty bit was 0 (clean), vm_page_cache was called and the cached pages counter is incremented. Toward the end of the for-loop scanning the inactive queue, pages were queued to be cleaned. This is where the count for pages queued to be flushed is incremented. Once the for-loop is completed, the number of pages cached, freed, and queued for flushing are logged into the debug file.

In the second for-loop inside of vm_pageout_scan, the active queue is scanned to determine which pages can be moved to the inactive queue. Like the first loop, a flag is set inside to determine if the active queue was being scanned for the first time in the call to scan, and the queues scanned counter is incremented. In every iteration of the for-loop, the active pages scanned count increments. Inside the loop, the activity count is checked to determine if the page is to be deactivated, or to remain in the active queue. If it is the former, then the deactivated pages count is incremented. Finally, once this loop terminates, the number of pages moved from the active to inactive queue is logged, as well as the number of pages scanned in the inactive queue (the first for-loop), the number of pages scanned in the active queue (second for-loop), the total number of pages scanned (first for-loop + second for-loop), and the total number of queues scanned.

To summarize, the following information was logged in the debug.log file:

1. Pages in active queue (beginning of the scan call)
2. Pages in inactive queue (beginning of the scan call)
3. Inactive pages moved to cache (end of the first for-loop)
4. Inactive pages move to free (end of the first for-loop)
5. Pages queued for flushing (end of the first for-loop)
6. Active pages moved from active queue to inactive queue (end of the second for-loop)
7. Active pages scanned (end of second for-loop)
8. Inactive pages scanned (end of second for-loop)
9. Total pages scanned (active pages + inactive pages scanned)

10. Queues scanned (end of second for-loop)

2.2 FAT CHANCE paging

vm_pageout.c:
vm_pageout_init:

The first actual change to the code is to make the pageout daemon scan more frequently. Originally, the daemon runs at least once every 10 minutes (600 seconds). This value will be changed to run once every 10 seconds, resulting in the pageout daemon scanning the queues of pages more often.

vm_pageout_scan:

The activity count was changed where it was decremented. Instead of decrementing it we halved the activity count by 2 instead. This causes the pages to be moved to the inactive queue quicker. Also a parameter was added to the function vm_page_requeue_locked to modify how pages went into the inactive list. By adding this parameter, pages went to the front instead of the rear.

vm_page.c:
vm_page_requeue_locked:

A parameter was added to this function to determine whether or not pages went to the front or rear. By adding this parameter, it acted like a flag to put it at the front when needed while the rest was acting like how it was before. So at the correct place in vm_pageout_scan, when a page went to the inactive list, it went to the front and not the rear.

vm_phys.c:
vm_phys_free_pages:

This function is called by vm_page_free which then calls vm_freelist_add. The function was modified so pages with even page numbers are added to the front of the freelist while pages with odd pages numbers are added to the rear of the freelist. We determine whether the page number is odd or even by bit shifting the physical address by $\log_2(4096) = 12$ bits to the right, then taking that number modulus 2, (m->phys_addr>> 12) % 2.

vm_page.h:
vm_page_requeue_locked:

Added a parameter to this function to act like a flag. This flag would determine whether or not pages went to the front or the rear. The flag was turned on at the correct place while everything else functioned like before.