CMPS 111 — Spring 2017 – ASGN3

Alan Duncan, Aryan Samuel, Khoa Hoang

Description

After running a series of test on both the default and modified daemons of the FreeBSD OS, we came up with the following results and data. But before we dive into it, the following is the method we used to test our daemons.

The Stress Test

Since each stress test logs data to the $messages$ file in $/var/log/$, we need to clear the data in this file before each test. After researching online, the most efficient way to do this was use the following command

sudo cp /dev/null /var/log/messages

which essentially empties the messages log file so that we may have a fresh file to log our data into.

Before adding the main testing command line, we used the $logger$ command to insert a tag in the log file that lets us know when to start reading data. Similarly, after the stress test has ended, we use this command again to add a tag to the log file that lets us know when to stop taking in data. This is simply for convenience.

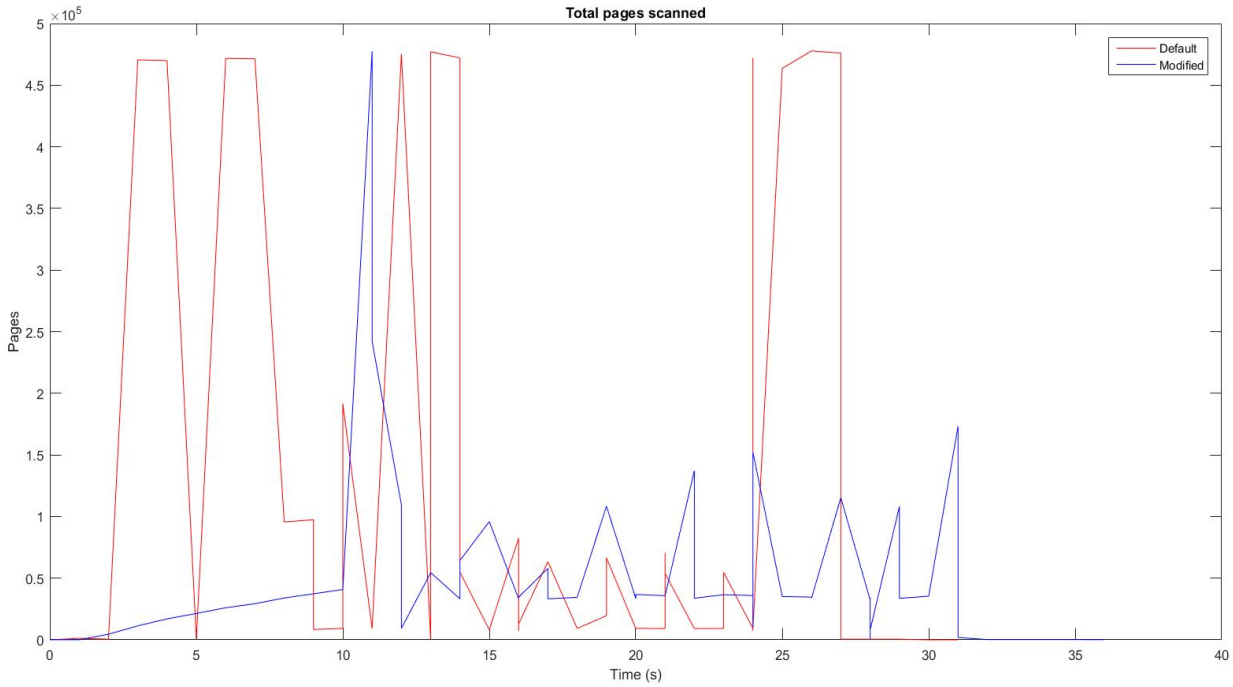
During each test, we used the stress command in FreeBSD as our main testing function.

stress –vm 15 –vm-bytes 200M –vm-stride 4096B –timeout-30s

Our VM had 2048 MB of RAM allocated to it, so we needed the test to use a little more memory than the RAM. As a result, we gave the stress test 15 workers, each of whom had 200 MB of memory allocated to them, giving us a total of 3000 MB of memory usage. Note that our VM also has extra swap memory, which is why even with 3000 MB of usage we did not get any swap errors, since there was sufficient amount of extra swap memory the base 2048 MB of RAM does not account for.
Combining these three methods, our input on the console was as follows:

```
$ sudo cp /dev/null /var/log/messages
$ logger START_LOG
$ stress –vm 15 –vm-bytes 200M –vm-stride 4096B –timeout-30s
$ logger END_LOG
```

The above steps were repeated 15 times for each of the daemons (Default and Modified), and we made a simple script to parse the $messsages$ file's logs into a series of .csv files that you can find in our submission. The following graphs represent the data within one of these .csv files, specifically file 15.

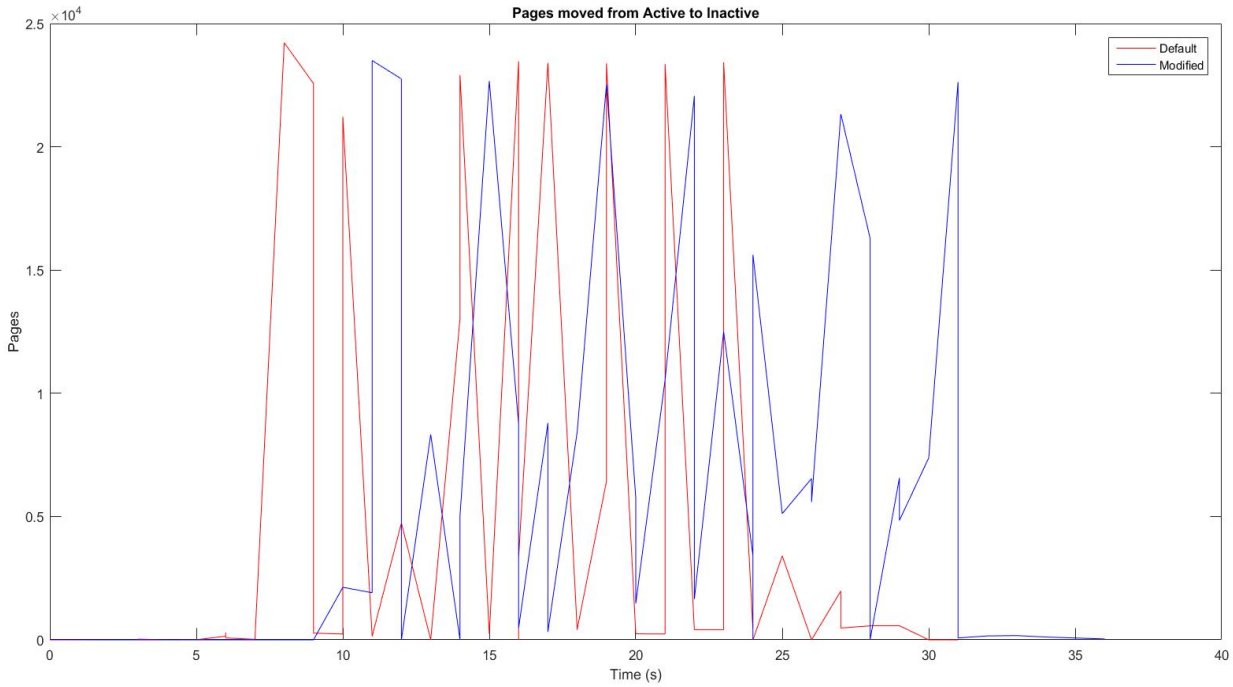


Total Pages

We originally believed that the number of pages scanned will not have a huge difference between one another, The reasoning for this is because we simply changed the update period to reflect 10 seconds(modified) rather than 10 minutes(default). However, the data shows that the number of pages scanned is actually much larger in the default FreeBSD page replacement algorithm. The reason for this is because the activity count is halved by two every time in the modified code rather than simply subtracting the activity count from a number.

Default Average: 137204.7

Modified Average: 53031.17

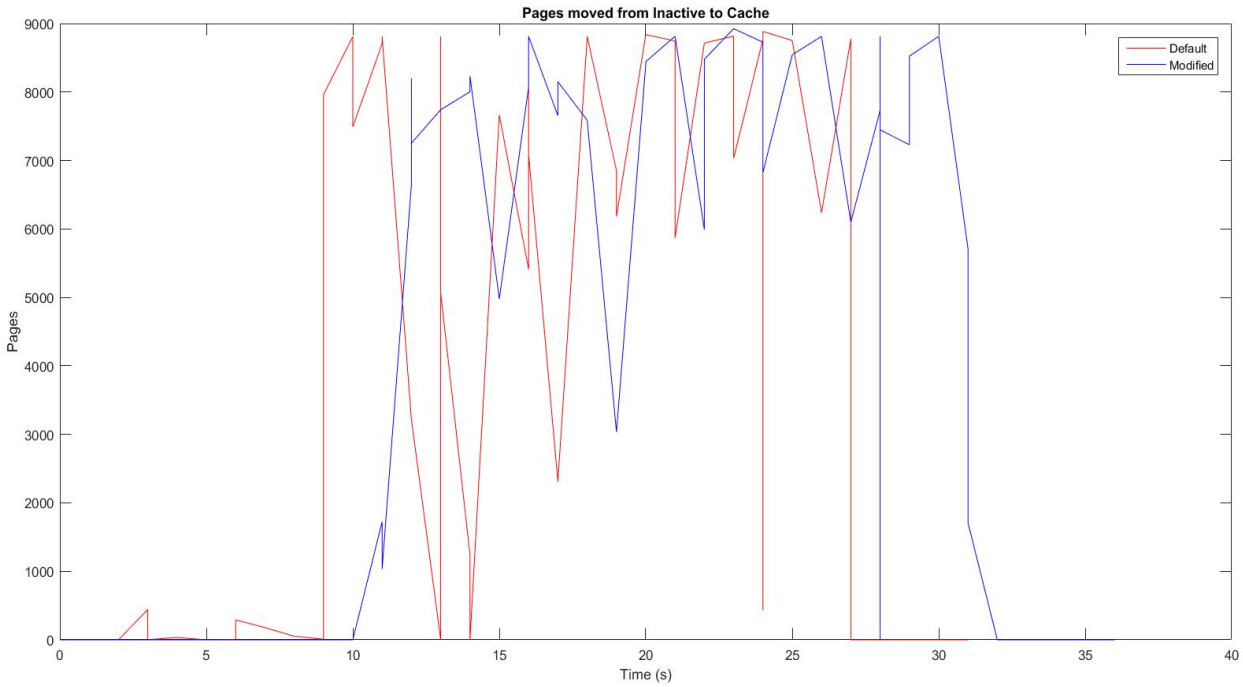


Active to Inactive

From the data we can see that the default and modified code for number of pages being moved from $Active$ to $Inactive$ is similar at first but then eventually the Slim Chance algorithm overtakes the default. Clearly, we can see that the difference of the amount of pages being moved is huge. The reasoning for this can be found in the name of the algorithm we are implementing, Slim Chance algorithm. The original, Second chance algorithm gives pages a ”second chance” - an old page that has been referenced is most likely not in use and should not be swapped out over a new page that has then been referenced. The Slim chance algorithm does not do this and will thus cause many active pages to swapped to inactive fairly quickly. Or in more simple terms, the pages will have less of a chance to be revisited inside the active queue.

Default Average: 4461.797

Modified Average 5953.774

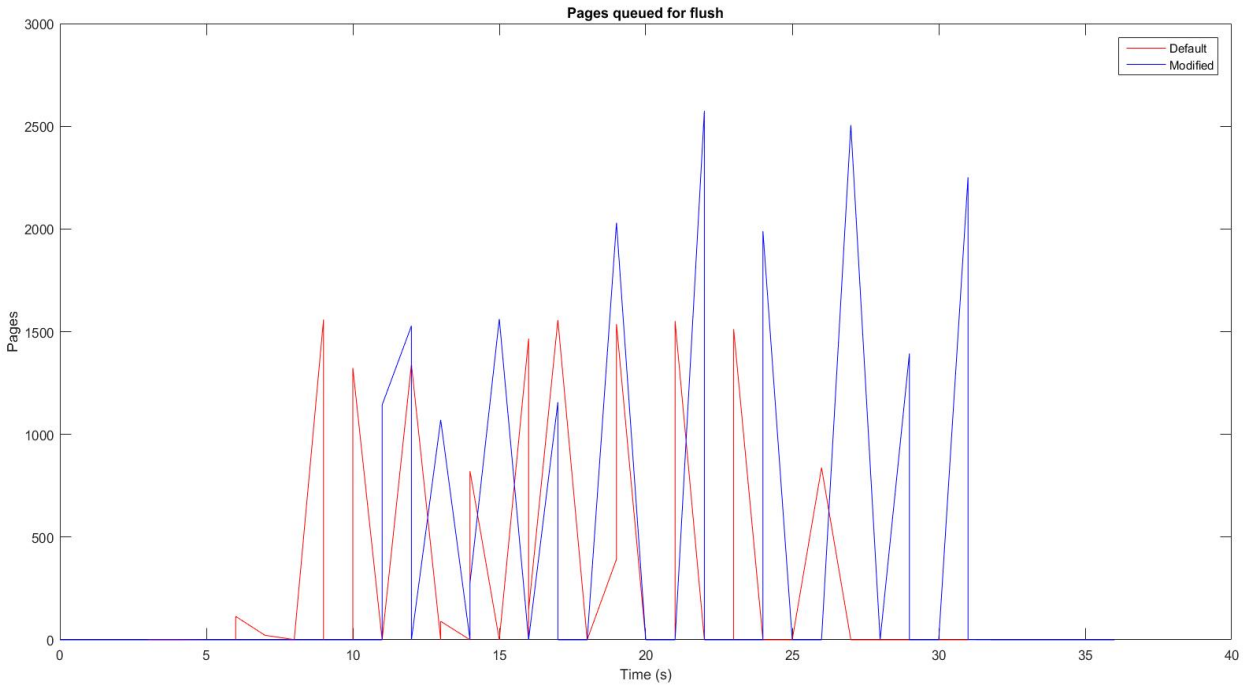


Inactive to Cache

Here, we can see there is huge fluctuations of pages being moved from the $Inactive$ queue to the $Cache$ queue. This can most likely be attributed to the number of $Active$ pages being moved into the $Inactive$ queue, thus this affects the total number of pages being moved from $Inactive$ to $Cache$. This is once again in the nature of the slim chance algorithm, being a less efficient page replacement policy compared to second chance. If we look closely at the data, we can see that towards the end, the number of pages being moved to the $Cache$ is a result of the above graph, where $Active$ to $Inactive$ pages is a large amount.

Default Average: 4102.34

Modified Average: 4992.32



Queued for Flush

In this last set of data, we can see that the number of pages being queued for flush in our modified code is significantly larger than the original FreeBSD algorithm. Not to sound like a broken record, but this is indeed once again a result of the nature of how slim chance algorithm works. To reiterate, the default one gives each page a ”second chance” to be revisited. We can see in the graph that there is a relatively consistently low amount of pages being queued for flush in out default compared to our modified code. The number of inactive pages being pushed into the cache is much larger in our modified version and thus will have more pages being queued for flush. We can clearly see that the original is a much more efficient algorithm

Default Average: 274.1

Modified Average: 367.6

Other data

Modified:

735186 - page reclaims

18891 - page faults

Default:

16209095 - page reclaims

13157 - page faults

Here we can see that the number of page reclaims is much larger in the default code. In addition, the number of page faults in the default is much less than the modified one. This is a clear indication that the default one is a far more efficient algorithm than the modified one as evidenced by the data above.