Writeup for Assignment 3 - Page Replacement

Kenneth Mayer
Marian Vladoi

March 5, 2018

1 Introduction

This document shows the results of some tests done by starting a memory-intensive process, running it to completion, and plotting the results.

2 Testing Methodology

We attempted a few different methods for testing the page replacement algorithm. We tried the FreeBSD port of the `stress` package, but we encountered some difficulties. It seemed we couldn't get the program to provide the correct level of stress without running out of swap space. We also tried the program posted on Piazza that stresses memories with multiple threads, but that didn't seem to work that well either.

Eventually, we settled on the memoryHammer.c program, which seemed to provide enough stress. This was the input that we used to run the test.

```
cp /dev/null/var/log/messages ; ./a.out ; cp /var/log/messages test
```

Where `a.out` is the binary of memoryHammer.c I chose to allocate 4000 MB (my VM had 4096 MB). The memoryHammer.c program by default runs forever, but I modified to run the main loop three times.

That command first clears the messages file (where the system log is written to), then it runs the program, then copies the output to a file called test.

After this is complete, I would run a script on it to extract the information into a format that was excel friendly, which I could then use to compile the plots.

3 Charts

I ran ten passes of the test using both the modified and the default algorithms. Unfortunately, the timing of the log entries is inconsistent, so I was only able to correctly plot one of the passes vs time without writing some more clever scripts, which I just wasn't up to.

Nonetheless, I will provide the average for each of the metrics over all of the ten passes.

3.1 Active Scanned

Figure 3.1 shows whether the active list was scanned in this particular test pass over time. The active list is scanned every time the daemon runs, so the times where it wasn't scanned means the daemon simply wasn't run at that particular second.

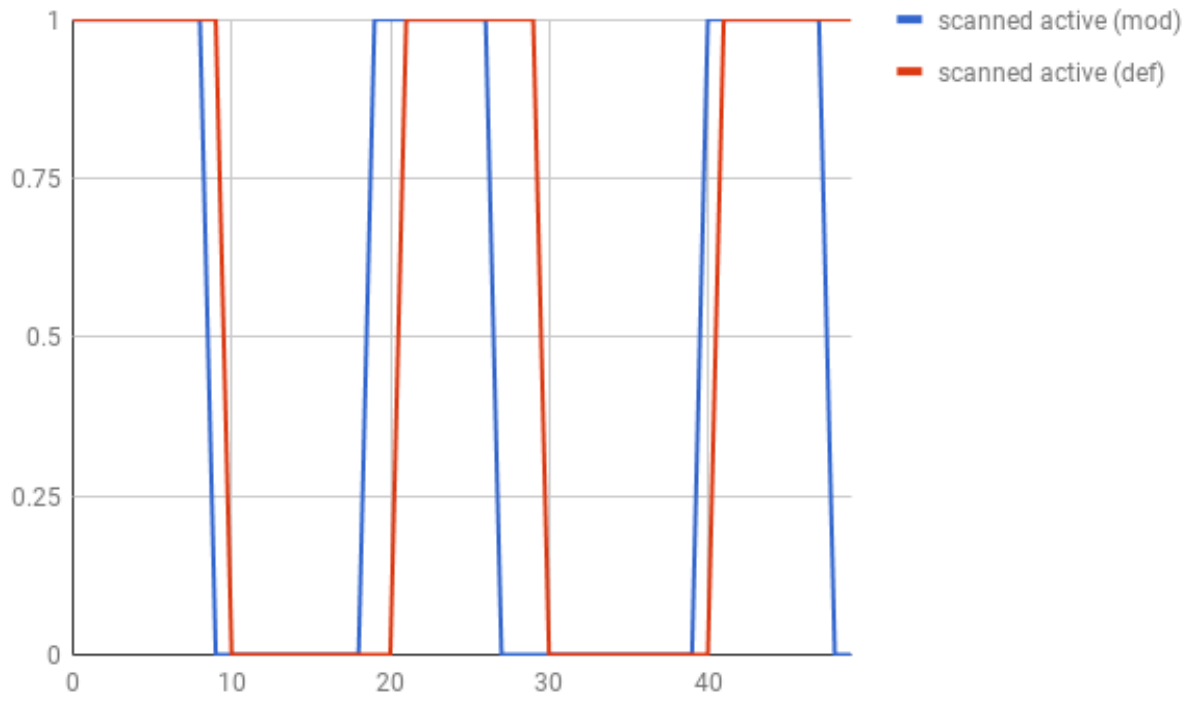

Figure 3.1: Chart showing how the active list was scanned over time.

There's very little to say about this, as the active list is always scanned.

3.2 Inactive Scanned

Figure 3.2 shows whether the inactive list was scanned in this particular test pass over time. Because we were so low on memory, it was scanned on almost every pass.

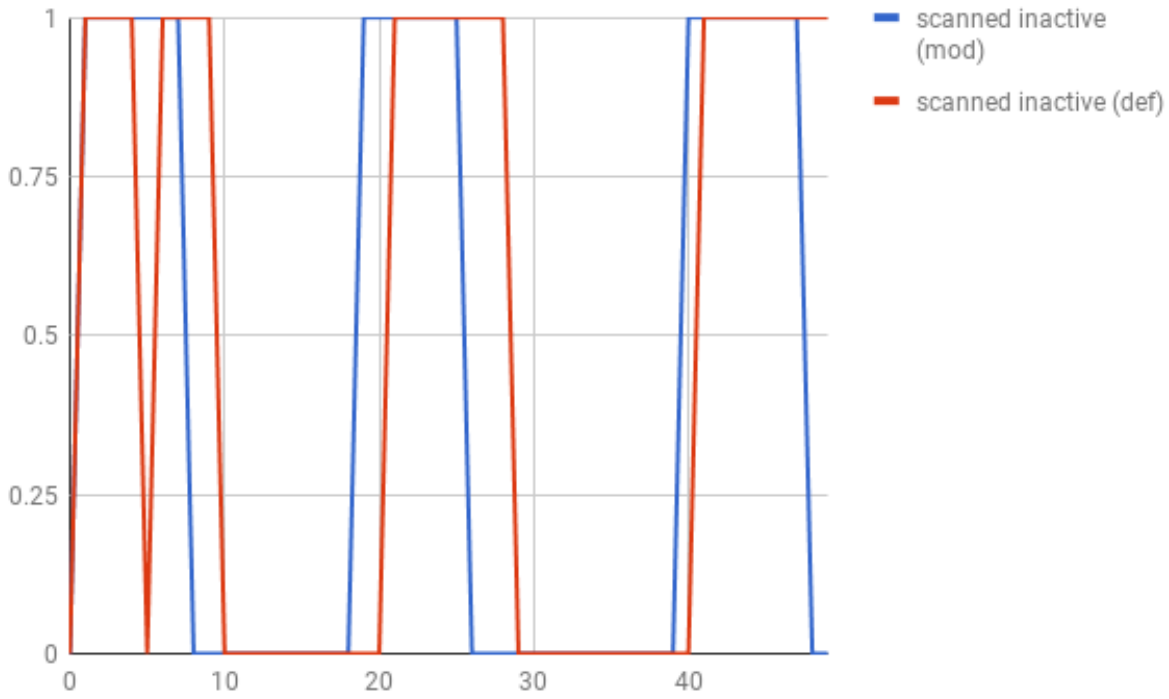

Figure 3.2: Chart showing how the inactive list was scanned over time.

3.3 Active to Inactive

Figure 3.3 shows how many pages were moved from the active list to the inactive list in order to find more pages to free.

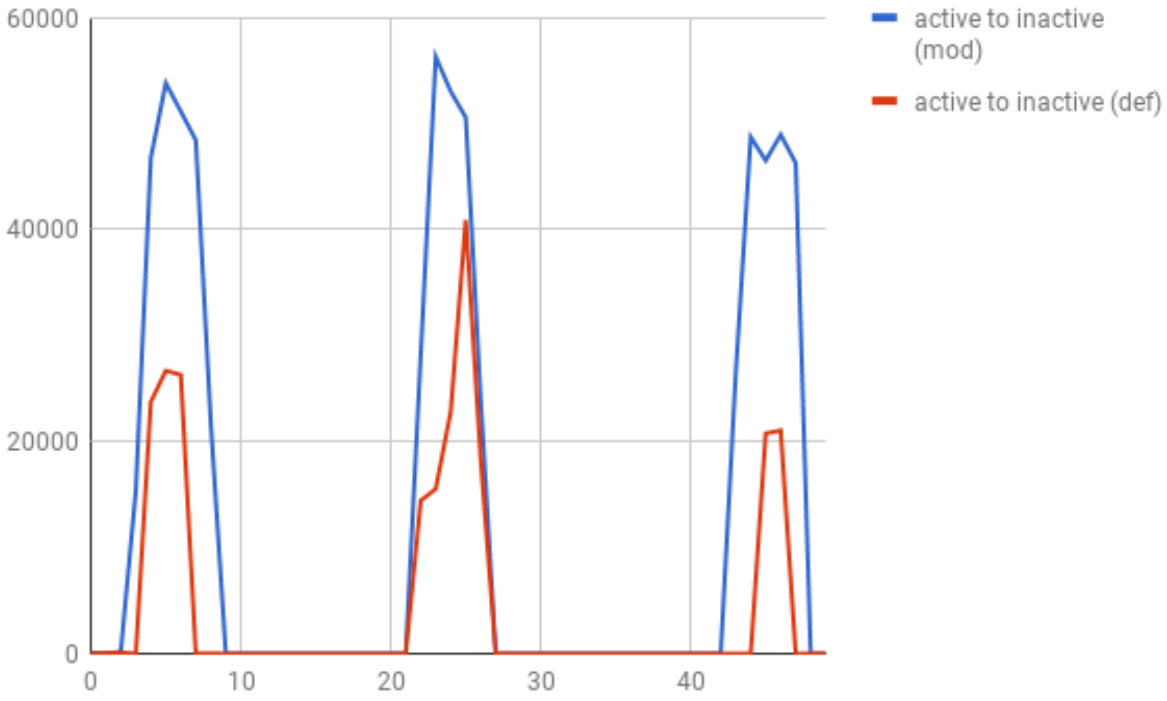

Figure 3.3: Chart showing how many pages were moved from the active list to the inactive list over time.

The modified algorithm moves a lot more pages from the active list to the inactive list. This has to do with the fact that we're moving every page to the front of the active list, even ones that were not used often. If a page was already due to be demoted from the active to the inactive, the next scan is going to check it again first, and most likely do that, where the original algorithm would have placed it on the end of the active list, so that it wouldn't be looked at again for a while.

Ten pass average (over time):
Default: 12288
Modified: 20914

3.4 Inactive to Cache/Free

Figure 3.4 shows how many pages were moved from the inactive list to the free list in order to fulfill more page requests.

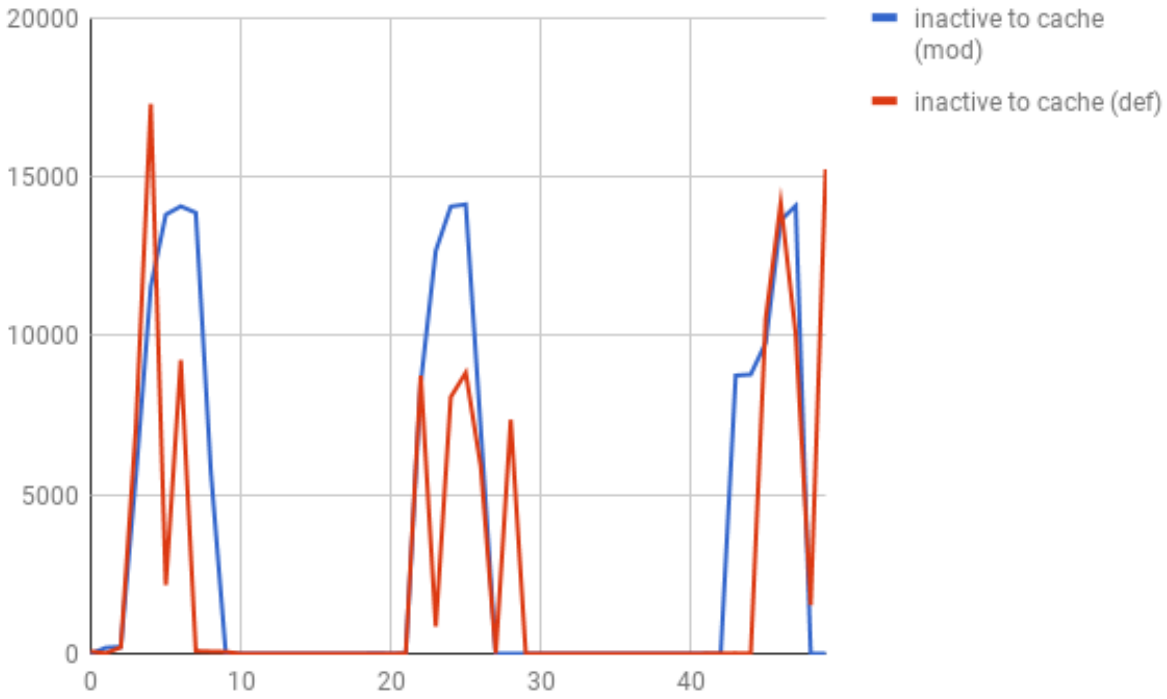

Figure 3.4: Chart showing how many pages were moved from the inactive list to the cache/free list over time.

Slightly more pages are moved from the inactive list to cache/free overall in the modified algorithm. This is likely because the fat chance algorithm simply has a harder time maintaining its cache/free target due to more page requests needing to be fulfilled.

Ten pass average (over time):
Default: 5226
Modified: 5800

3.5 Inactive Queued for Flush

Figure 3.5 shows how many pages in the inactive list were queued for flush over time.

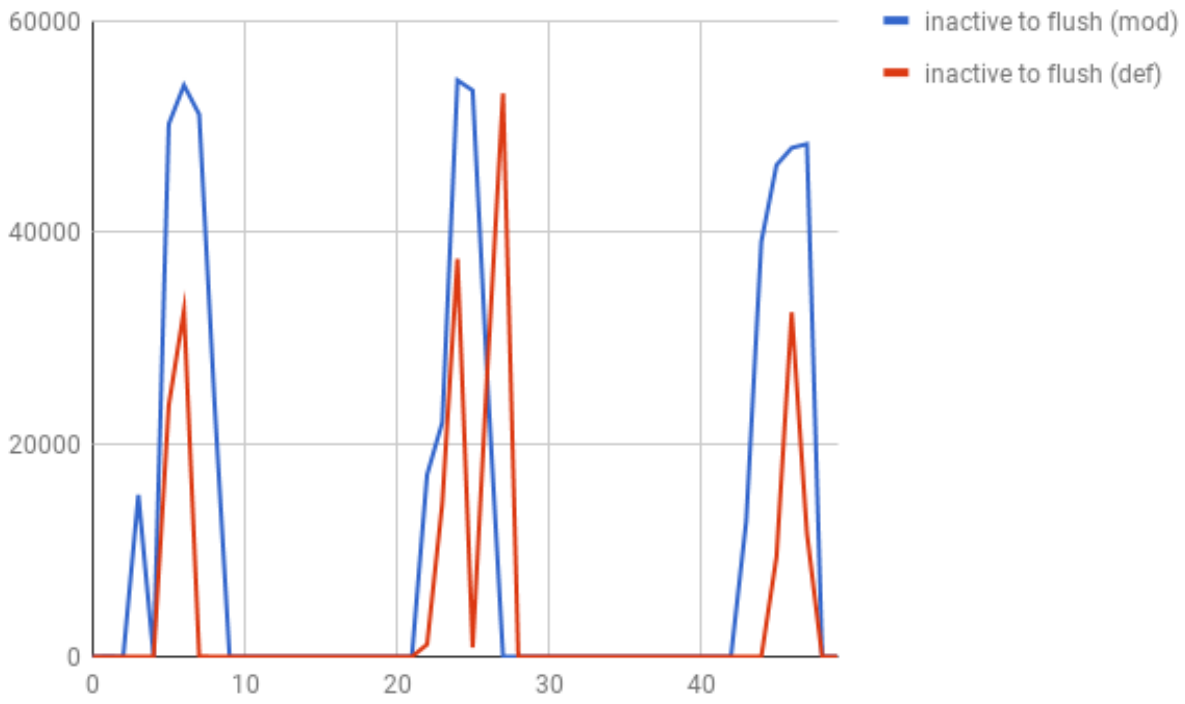

Figure 3.5: Chart showing how many pages in the inactive list were queued for flush over time.

The modified algorithm queues a fair amount more pages for flush on each run that the default. This is heavily related to the increase in pages moved from inactive to cache with the new algorithm. Because pages must be cleaned before they can be freed, and more pages need to be freed with the new algorithm, more pages get queued for flush.

Ten pass average (over time):
Default: 13538
Modified: 20000

3.6 Total Inactive Pages

Figure 3.6 shows the total number of inactive pages after the scan is completed. This should show how well each algorithm manages to maintain its page shortage target.

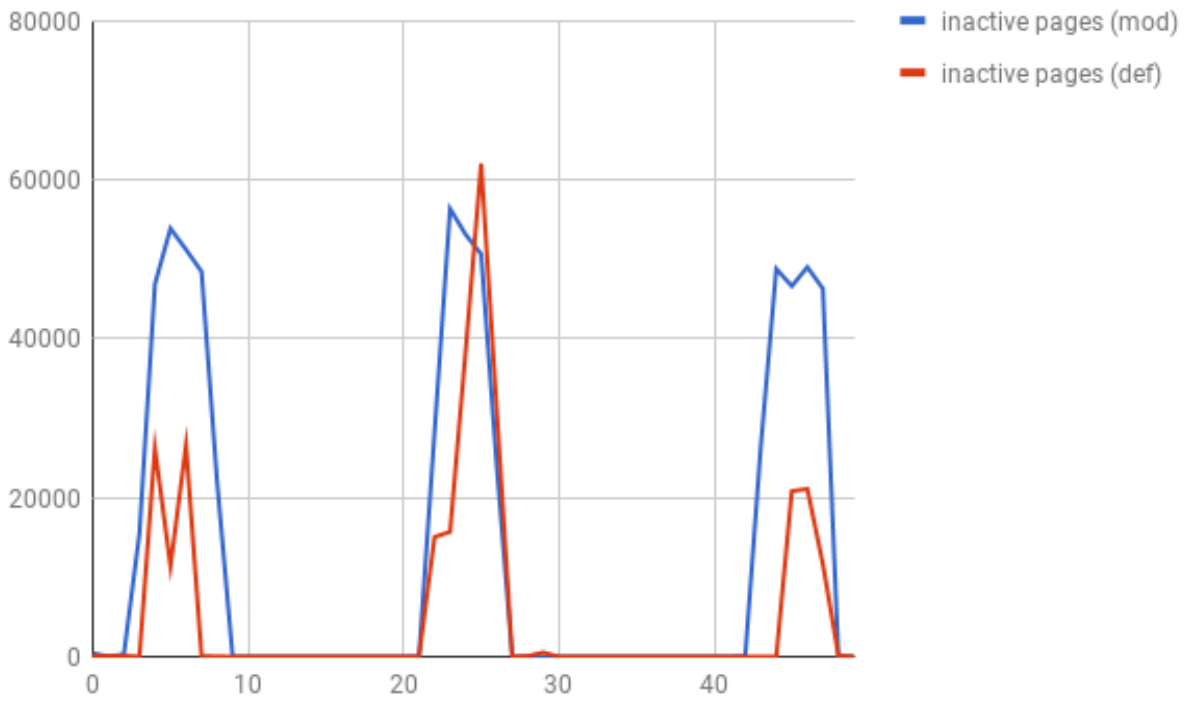

Figure 3.6: Chart showing how many pages there were in the inactive list over time.

Interestingly, the modified algorithm seems to maintain this shortage more effectively, which is undoubtedly related to the way it throws pages into the inactive list with reckless abandon.

Ten pass average (over time):
Default: 20999
Modified: 14333

3.7 Total Active Pages

Figure 3.6 shows the total number of active pages after the scan is completed.

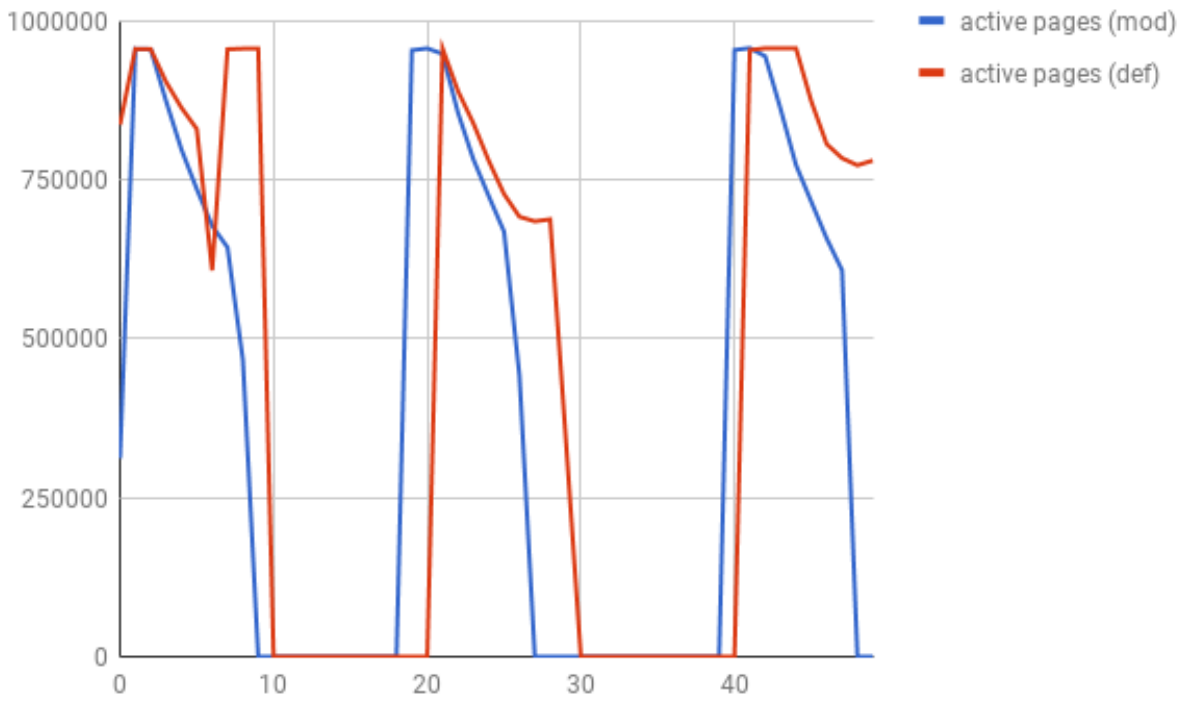

Figure 3.7: Chart showing how many pages there were in the active list over time.

There's not really too much to say about this. The number of active pages is so high compared to the inactive that both algorithms show very similar behavior.

Ten pass average (over time):
Default: 727397
Modified: 616021

3.8 Final Comment

Because the test we used was of variable time, we might have expected to see a difference in the average run time of the memoryHammer program between the two algorithms. However, the difference was fairly small, with the total time taken for the default algorithm actually being larger. This could very well be due to test variance.