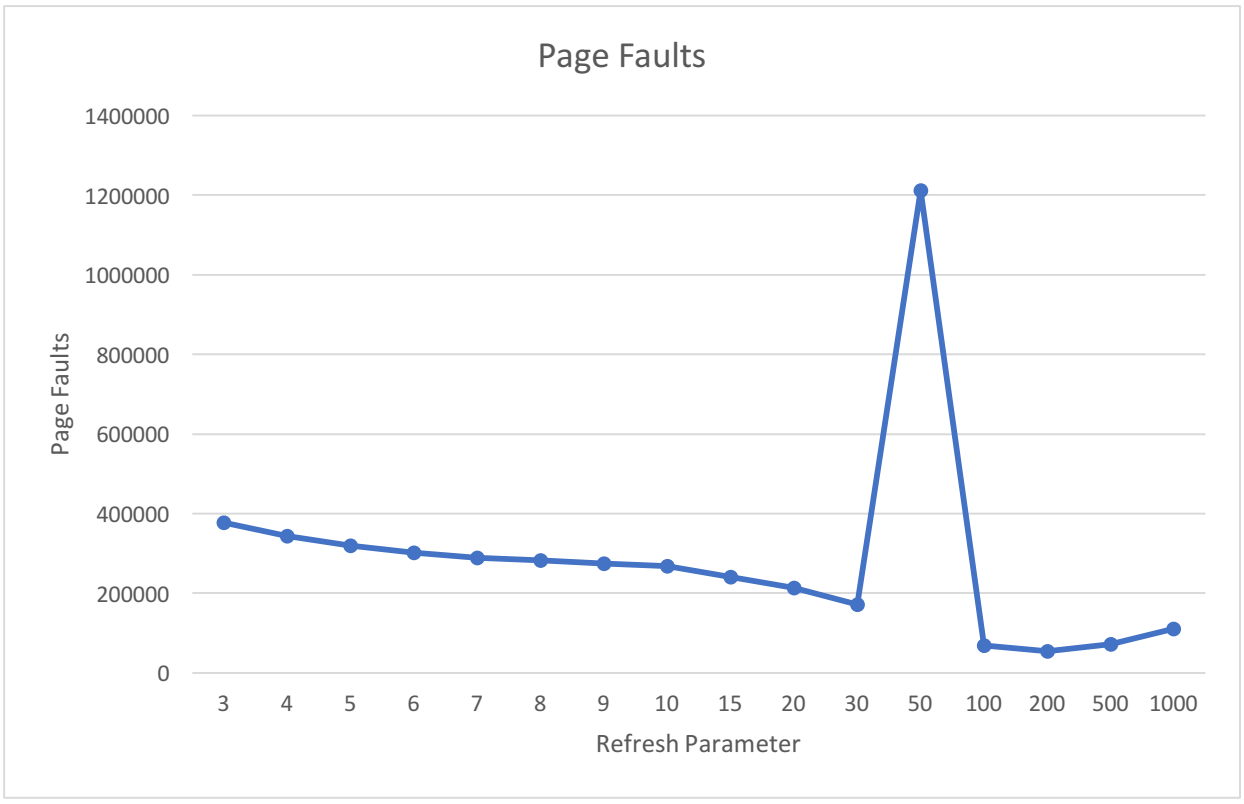
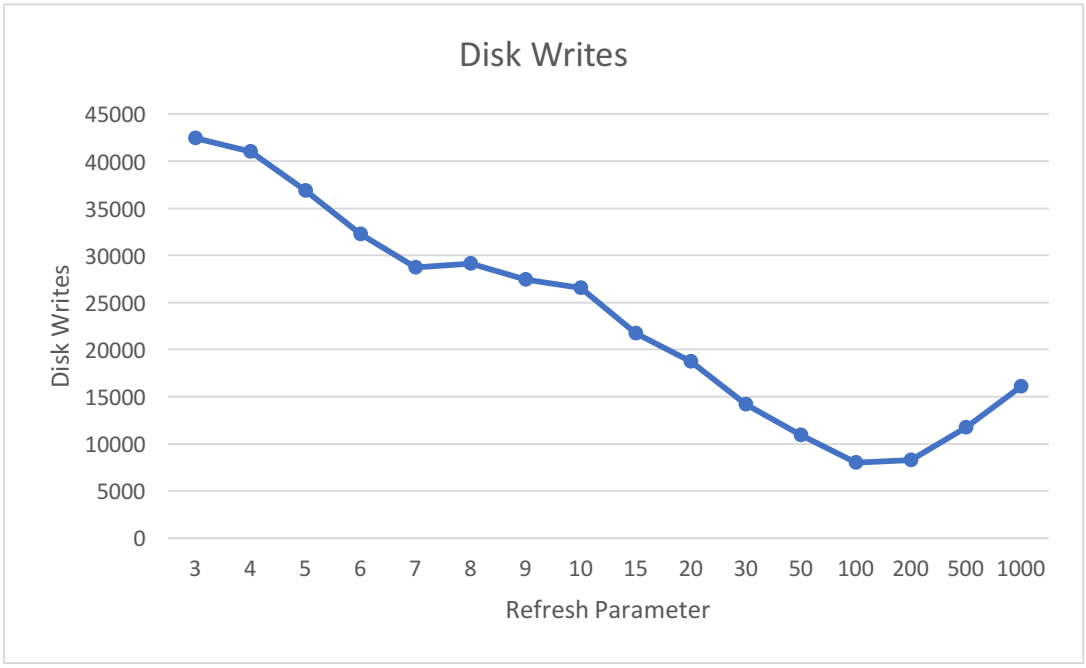
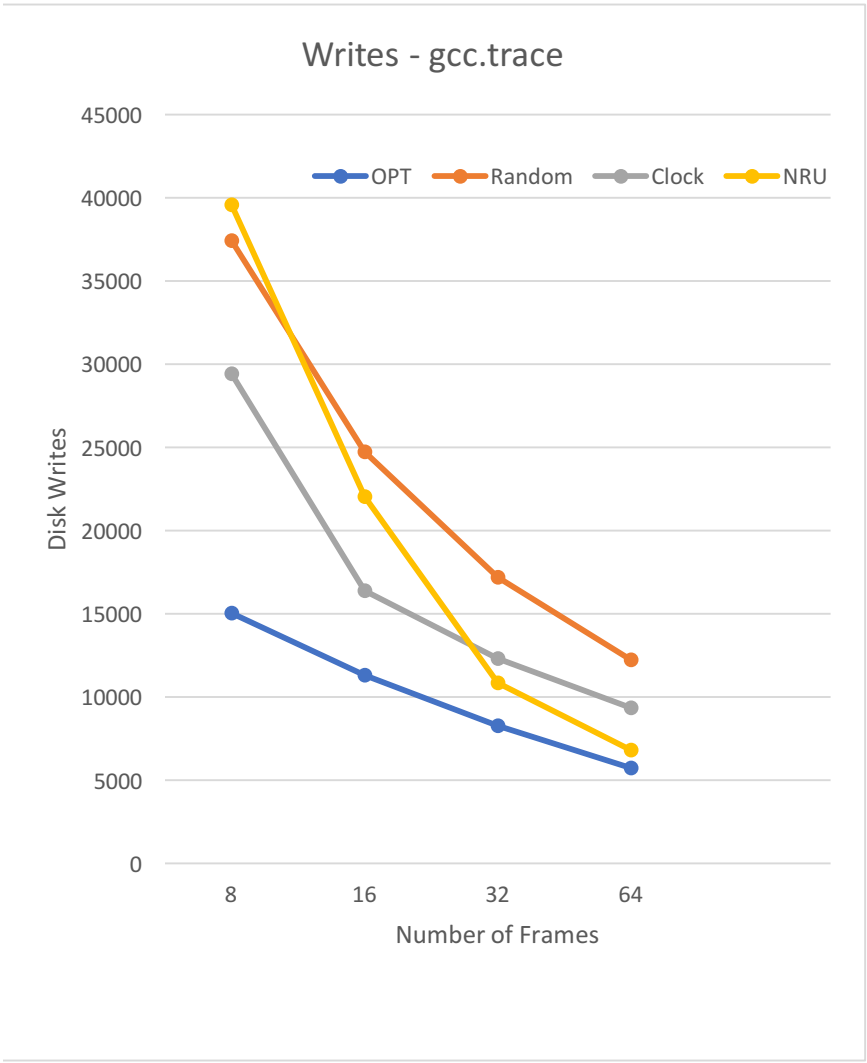
Lucas Brennan
CS1550
Misurda
17th November 2017

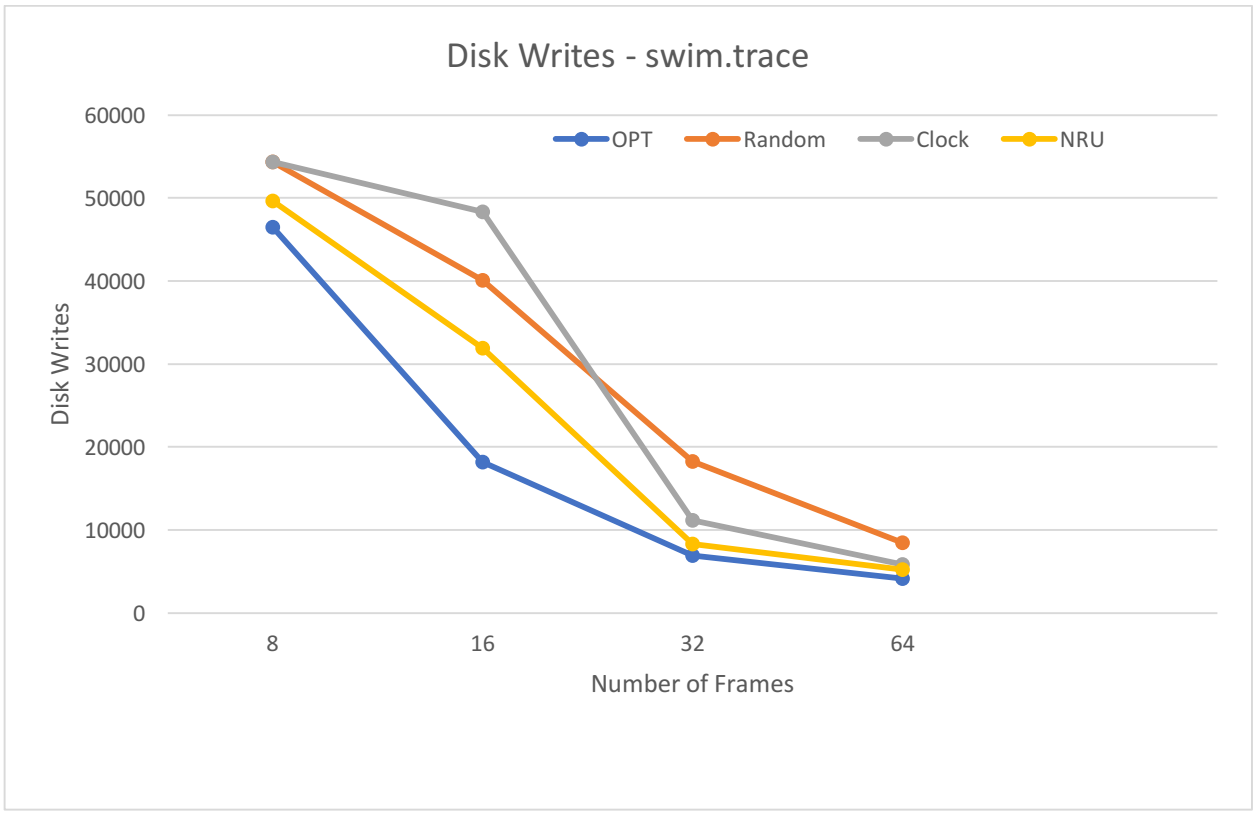
Project 3 Write-up

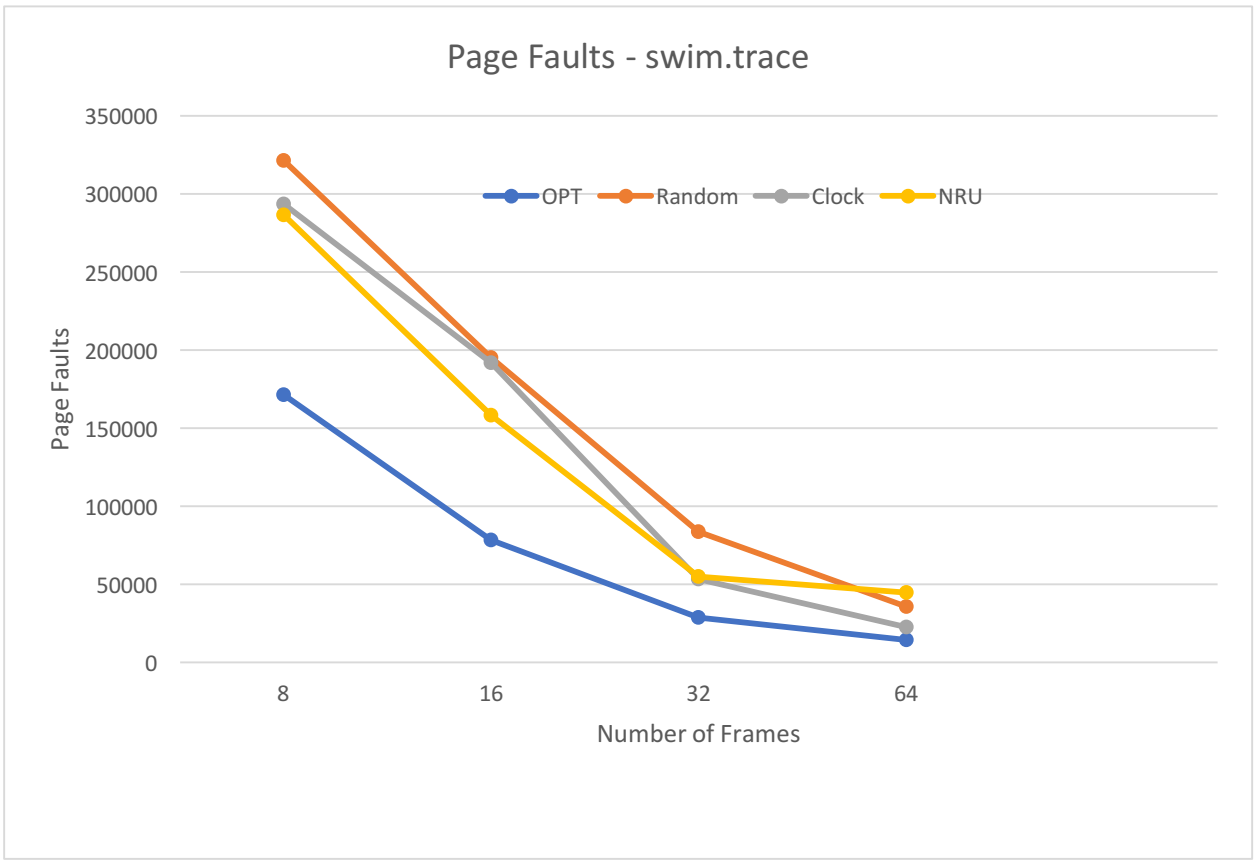


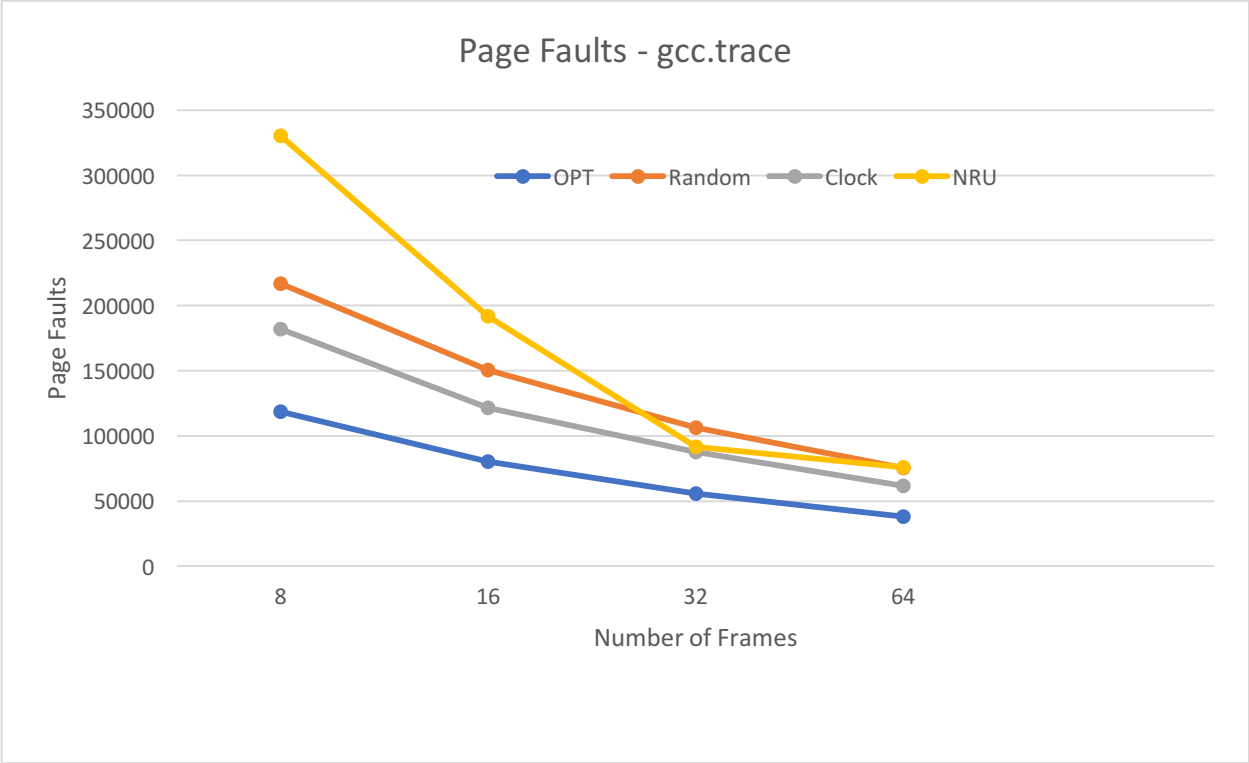


For the NRU algorithm one of our tasks was to pick a good time to refresh the page frames and deference things so we make sure to evict pages that haven't been used recently. I used a brute force approach to find a good refresh point. I picked 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100, 200, 500 and 1000 as refresh points and ran my program at 32 frames and recorded the number of disk writes and page faults. I kept the number of frames steady in order to decrease the possibility of a confounding variable. I then graphed my results. What I saw in these graphs, included above, is that the higher the refresh was the less page faults and disk writes I got. This was the trend until I went past 200 and I started to get a diminishing return. So with that information 200 seemed to be the best refresh parameter to my program and that is what I ended up using in order to compare all the algorithms.




Disk Writes - swim.trace
60000
50000
40000
30000
20000
10000
0
Disk Writes
OPT
Random
Clock
NRU
8
16
32
64
Number of Frames


Page Faults - swim.trace
350000
300000
250000
200000
150000
100000
50000
0
Page Faults
OPT
Random
Clock
NRU
8
16
32
64
Number of Frames



Algorithm Comparison

Disk Writes – After running the algorithms, I plotted all the data. From the graphs above, with gcc.trace NRU started at the most amount of disk writes followed by random, clock and of course optimal. Interestingly, as the frames were increased NRu had the least amount of page faults compared to Optimal and clock followed close behind leaving random with the most disk writes for that file. With swim.trace NRU started with the least amount of disk writes and also finished with the least amount compared to optimum. Clock and Random had similar performance but ultimately clock finished with the least amount of disk writes.

Page Faults – In terms of page faults the data revealed similar conclusions. With gcc.trace, for 8 frames NRU performed the worst, followed by random then clock compared to the OPT algorithm. AS the frame number increased NRU improved and finished and finished almost with the same page faults as random. Clock performed the best at 64 frames compared to the OPT algorithm. Swim.trace revealed that Random starting off with the most amount of page faults followed by clock and NRU. As the number of frames increased, however, Clock finished with the least amount of page faults, Random with slightly more and then NRU finishing with the most amount of page faults at 64 frames compared to the OPT algorithm.

Conclusion – In a modern operation system I am going to assume that we would have at least 64 frames so I am judging my conclusion off of that. My choice would be the clock algorithm. The clock algorithm gave us the least amount of page faults and finished with more disk writes than NRU but less than random. I am not that worried about a hit in the amount of disk writes especially with the speeds of solid state drives and the increasing speed of hard drives in general. I am more concerned with page faults do to the possible performance hit that increasing amounts of page faults could cause. Another benefit of the clock algorithm is the implementation simplicity. It was more difficult to implement than random but still relatively simple.