Matthew Lanus
Matthew Flanagan
April 7, 2017

Virtual Memory Lab Report

Overview:

The purpose of the lab is to understand and implement virtual memory. By implementing the virtual memory, we, the programmers, receive better insight on how a computer works internally. Our implementation not only tries to simulate previous algorithms already created, but also is aimed for us to create a new algorithm. For this project, we used the student00.cse.nd.edu machine to test our code. We tested various commands to make sure our code worked properly and as expected. We used the following commands:

./virtmem 100 3 rand sort ./virtmem 100 3 fifo sort ./virtmem 100 3 custom sort
./virtmem 100 10 rand sort ./virtmem 100 10 fifo sort ./virtmem 100 10 custom sort
./virtmem 100 20 rand sort ./virtmem 100 20 fifo sort ./virtmem 100 20 custom sort
./virtmem 100 50 rand sort ./virtmem 100 50 fifo sort ./virtmem 100 50 custom sort
./virtmem 100 75 rand sort ./virtmem 100 75 fifo sort ./virtmem 100 75 custom sort
./virtmem 100 100 rand sort ./virtmem 100 100 fifo sort ./virtmem 100 100 custom sort

./virtmem 100 3 rand scan ./virtmem 100 3 fifo scan ./virtmem 100 3 custom scan
./virtmem 100 10 rand scan ./virtmem 100 10 fifo scan ./virtmem 100 10 custom scan
./virtmem 100 20 rand scan ./virtmem 100 20 fifo scan ./virtmem 100 20 custom scan
./virtmem 100 50 rand scan ./virtmem 100 50 fifo scan ./virtmem 100 50 custom scan
./virtmem 100 75 rand scan ./virtmem 100 75 fifo scan ./virtmem 100 75 custom scan
./virtmem 100 100 rand scan ./virtmem 100 100 fifo scan ./virtmem 100 100 custom scan

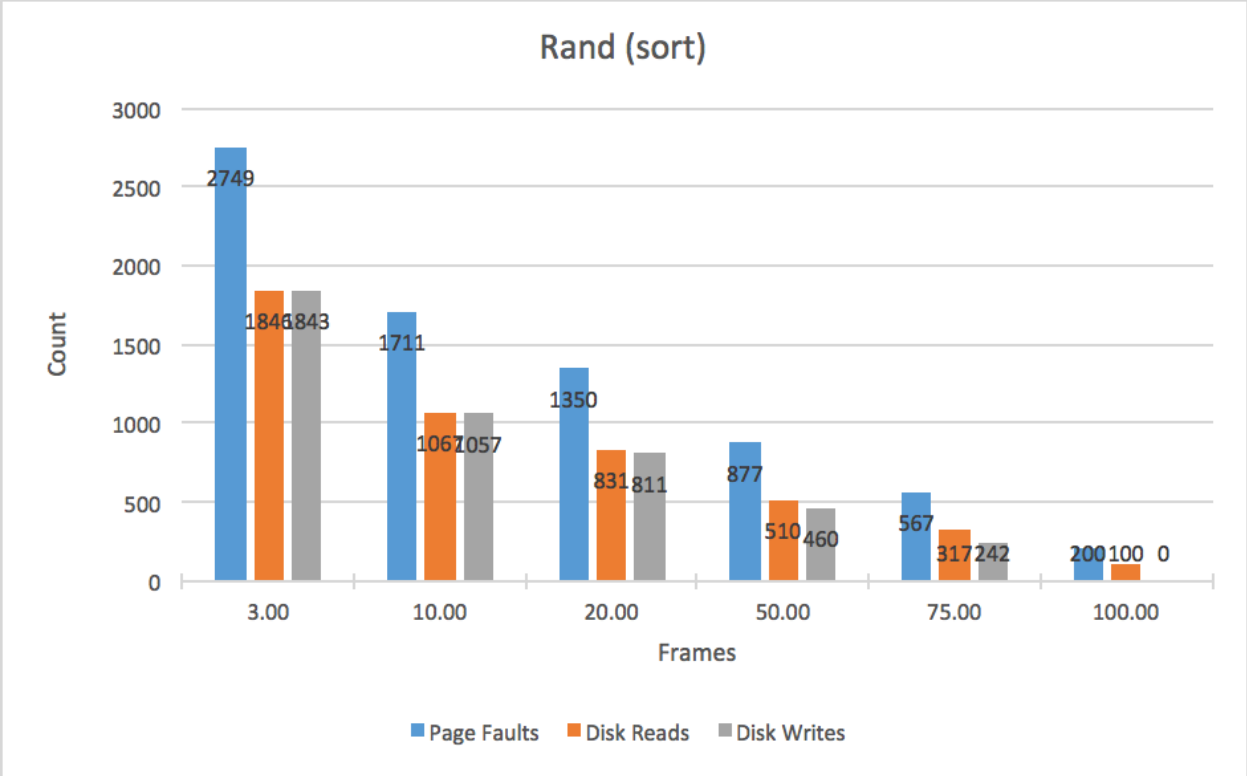
./virtmem 100 3 rand focus ./virtmem 100 3 fifo focus ./virtmem 100 3 custom focus
./virtmem 100 10 rand focus ./virtmem 100 10 fifo focus ./virtmem 100 10 custom focus
./virtmem 100 20 rand focus ./virtmem 100 20 fifo focus ./virtmem 100 20 custom focus
./virtmem 100 50 rand focus ./virtmem 100 50 fifo focus ./virtmem 100 50 custom focus
./virtmem 100 75 rand focus ./virtmem 100 75 fifo focus ./virtmem 100 75 custom focus
./virtmem 100 100 rand focus ./virtmem 100 100 fifo focus ./virtmem 100 100 custom focus

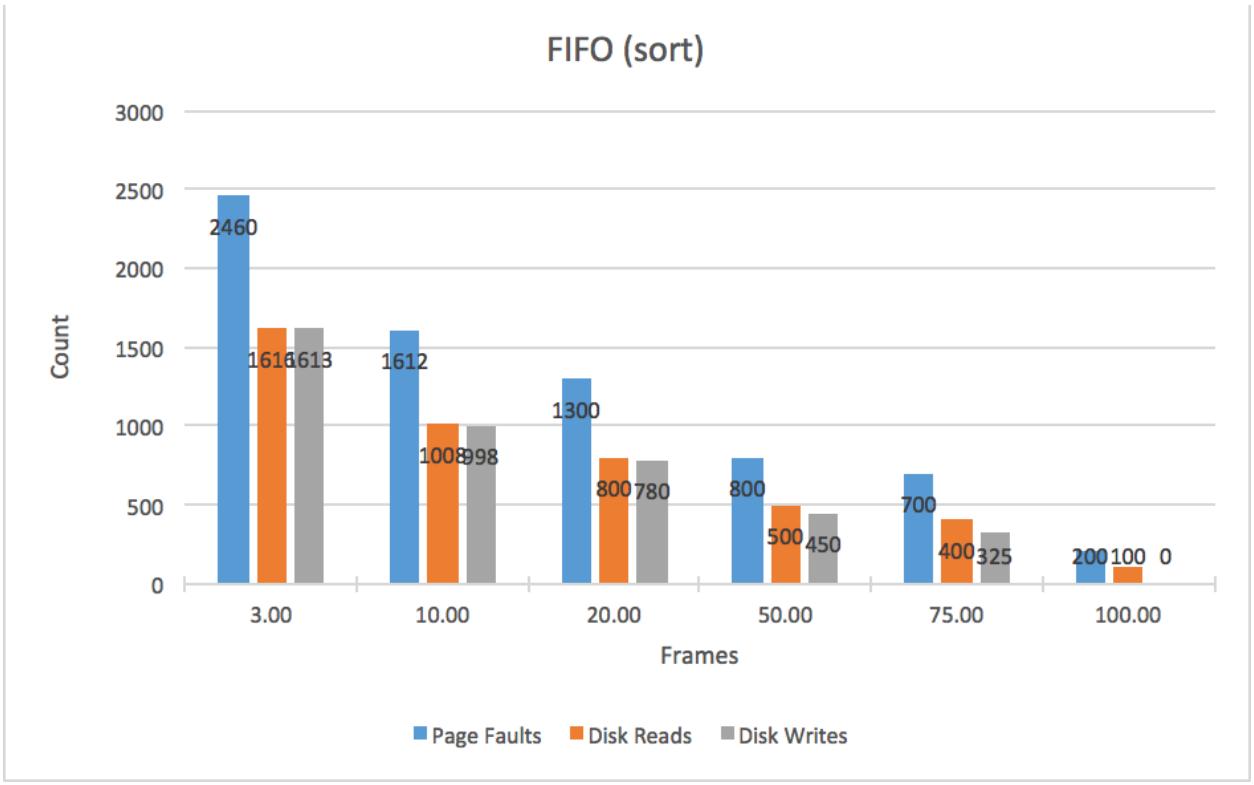
We wanted to make sure the results were consistent across various parameters. The output confirmed that our code produced the intended output.

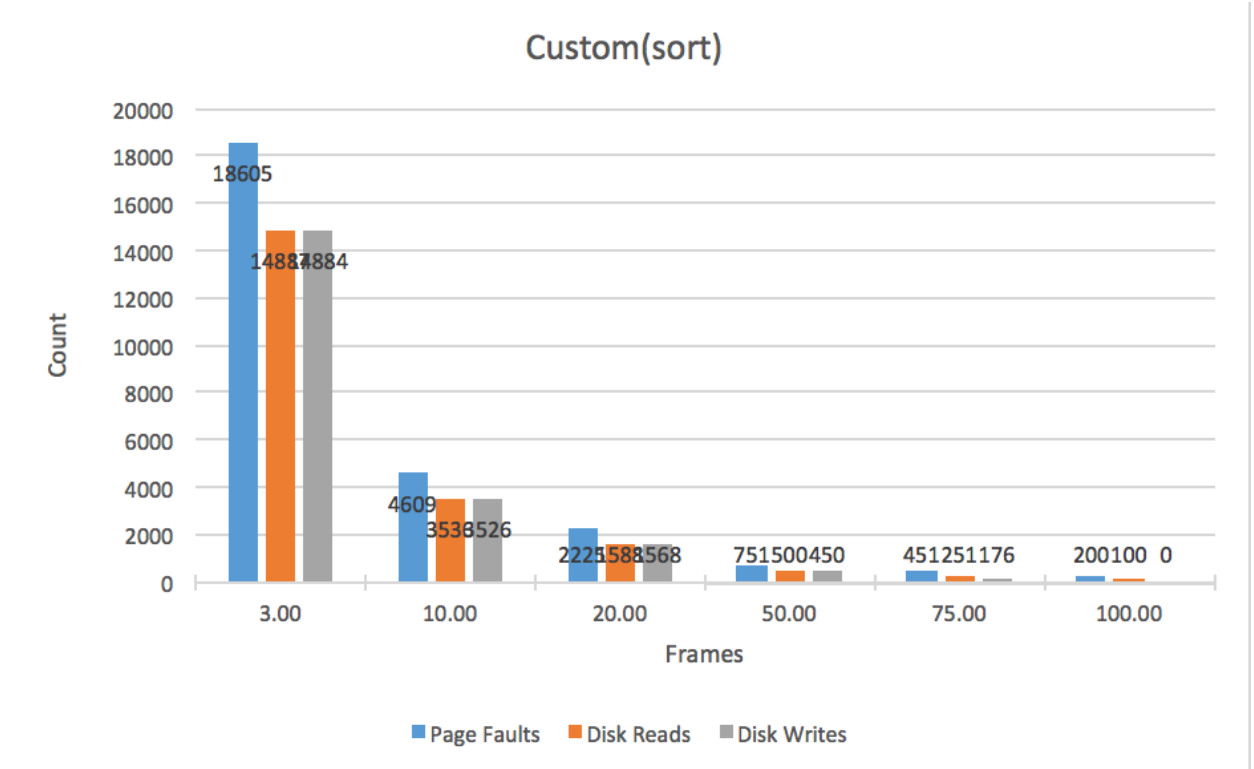
Custom:

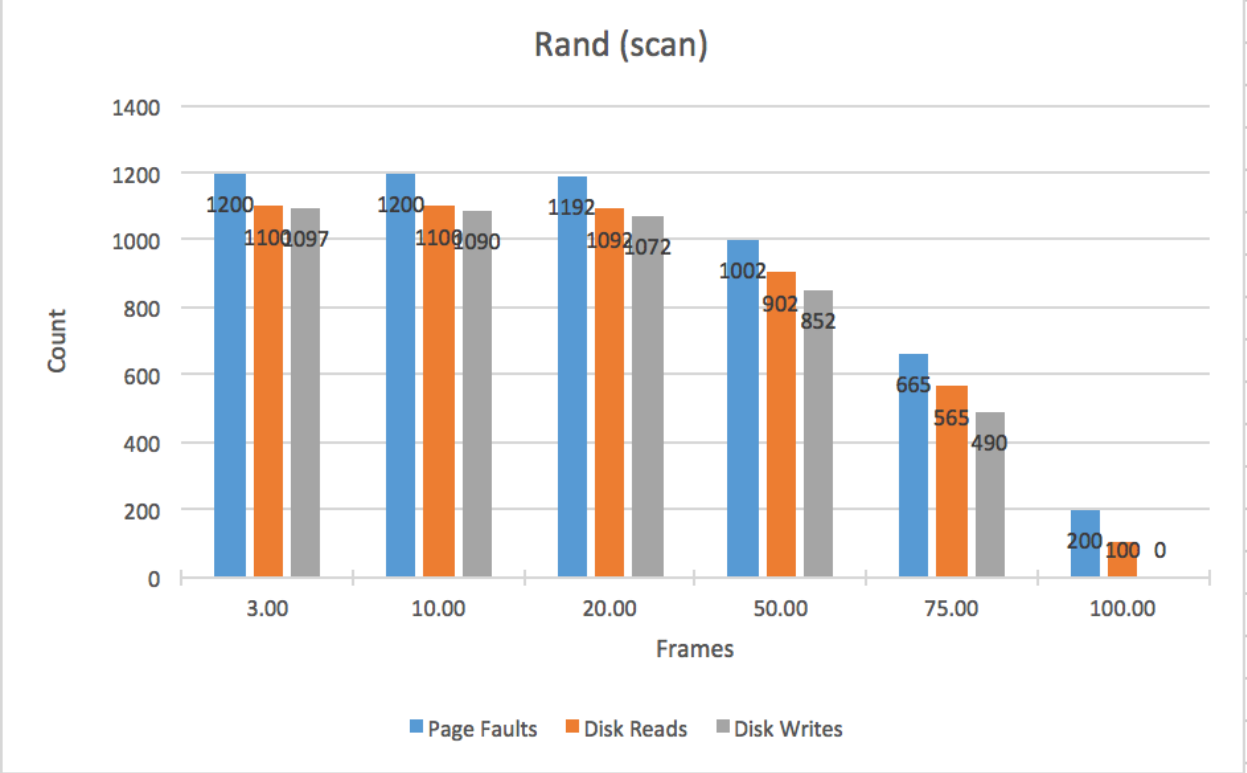
For our custom algorithm, from a high-level view, we attempted to simulate the least recently used (LRU) algorithm as closely as possible. To be more specific, we create an array called numFaultsOnPage to keep track of how times a certain page faulted. The page with the fewest faults will be replaced, suggesting that it is the least recently used. From an even more low-level description, we first increment the number of faults on that page. We also have a variable 'min' to keep track of which page we should replace and we loop through the page table and check the number of faults on that page (from the numFaultsOnPage) and compare it to min. If it is smaller than min, then we update min and record the page we will replace. Once we find the the item that we need to replace we go to the frame table and see if one of those items is the replaceable item. If the item that we are putting into the virtual memory has not read bits (not in the page table), then we write the frame we found to disk and set the item we were reading into the page table.

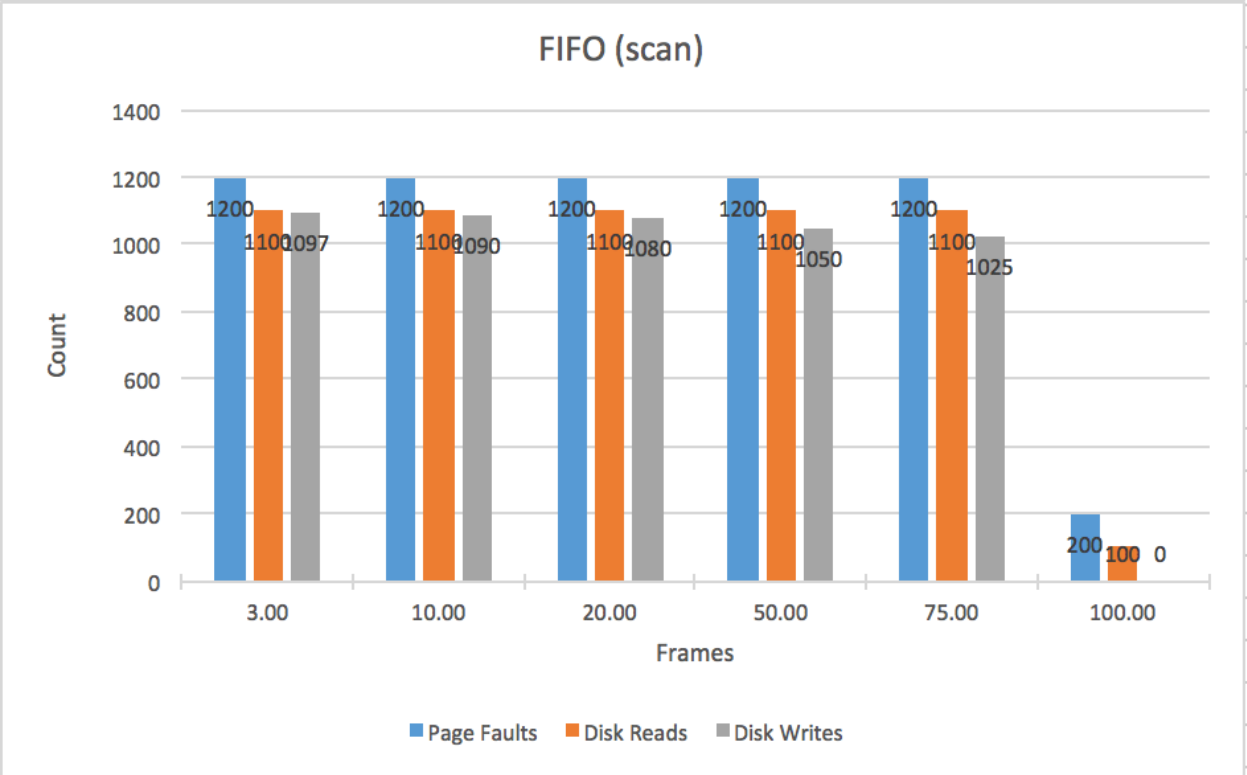
Results:

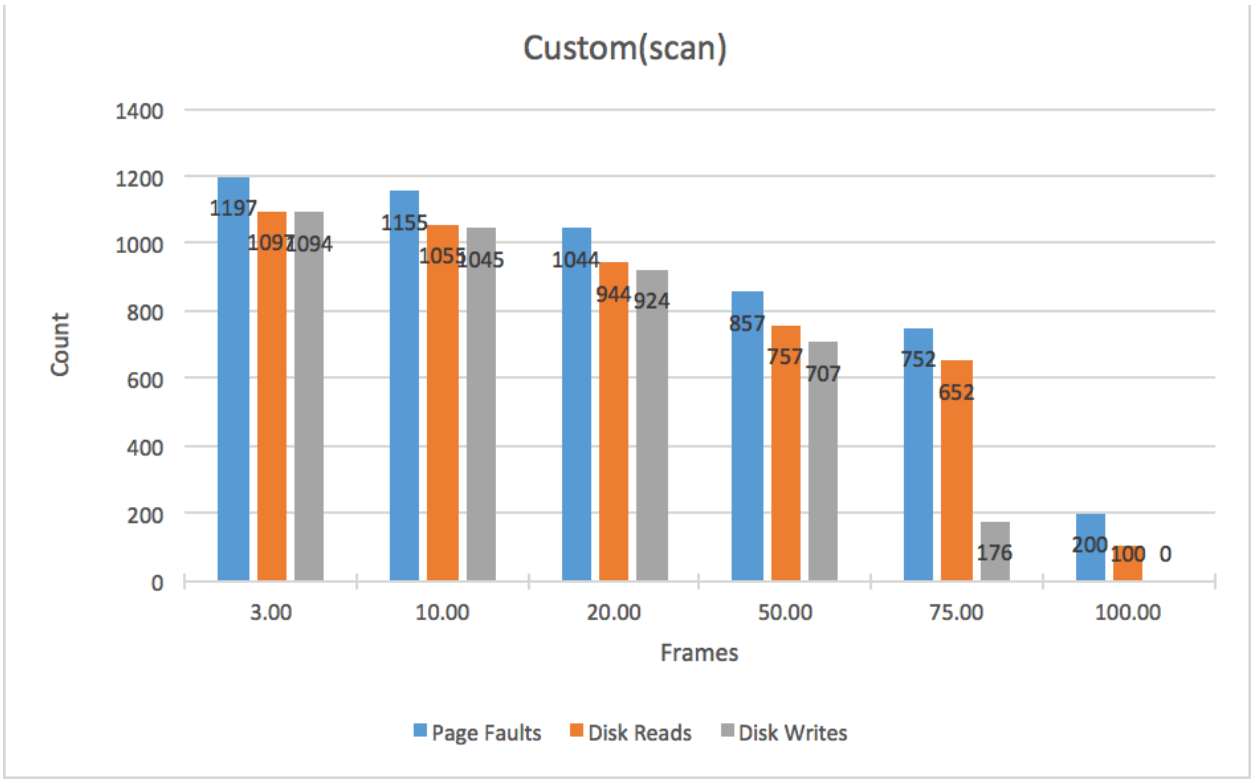


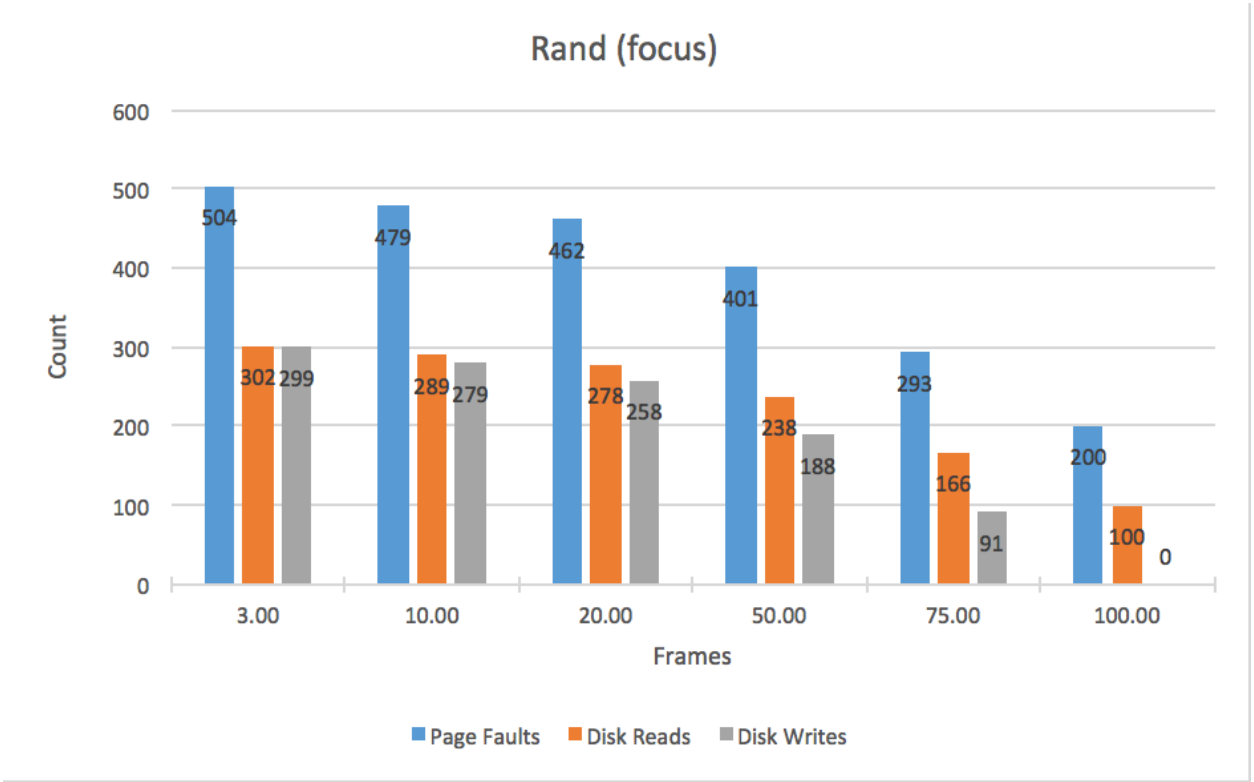

FIFO (sort)
3000
2500
2000
1500
1000
500
0
Count
2460
1616
1613
1612
1008
998
1300
800
780
800
500
450
700
400
325
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes

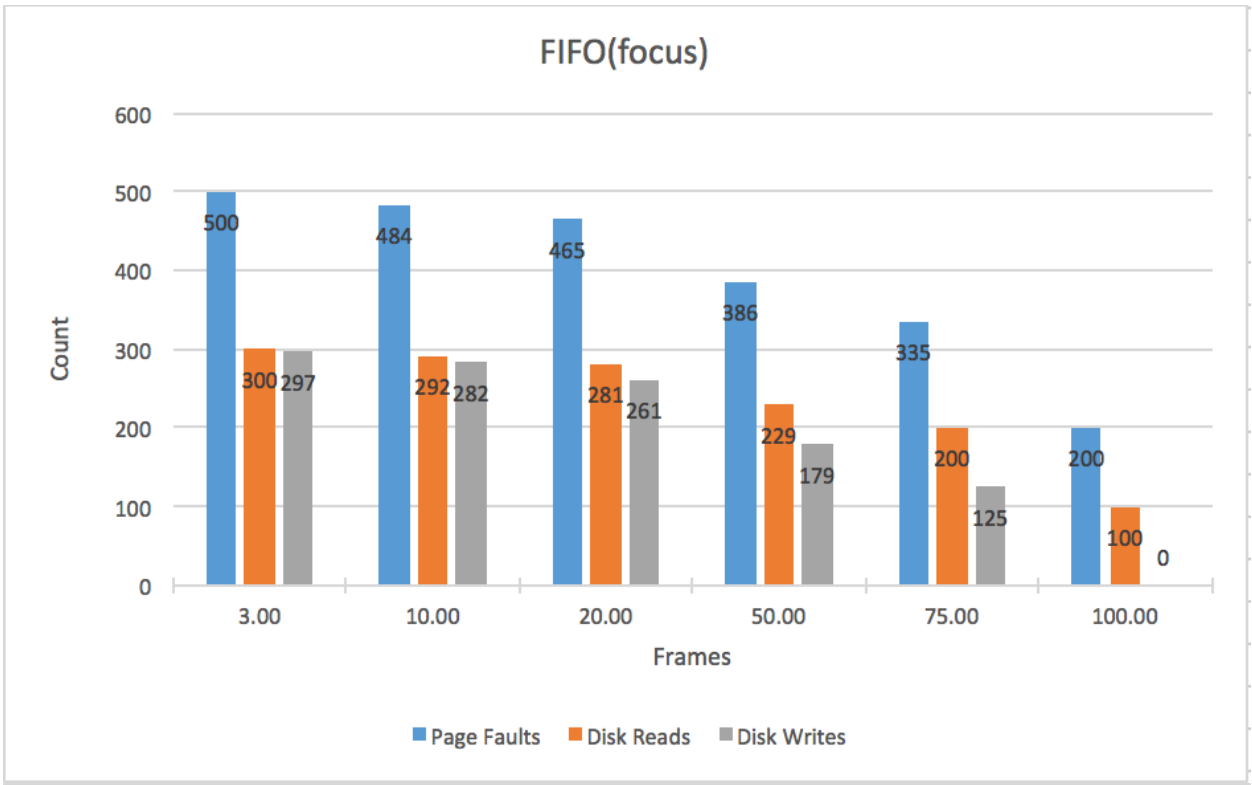

Custom(sort)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
Count
18605
14887
14884
4609
3538
3526
2225
1588
1568
751
500
450
451
251
176
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes

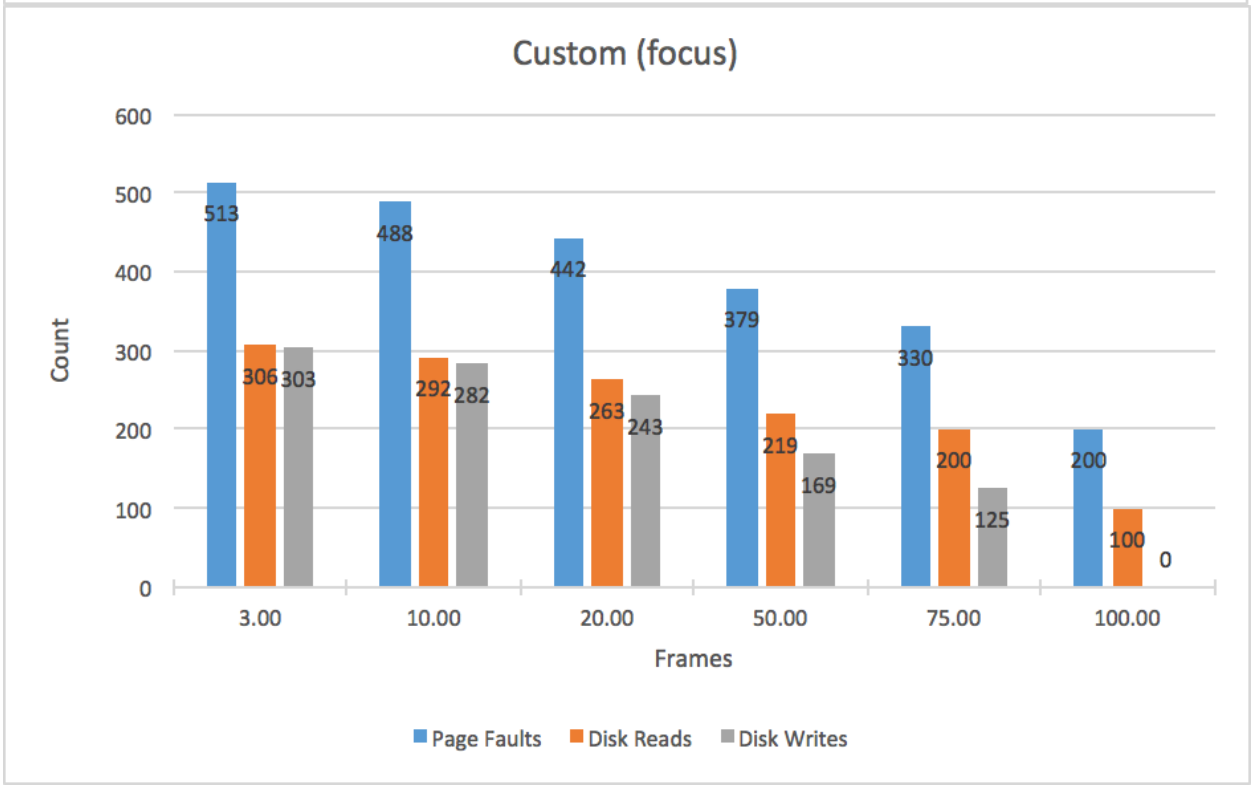

Rand (scan)
1400
1200
1000
800
600
400
200
0
Count
1200
1100
1097
1200
1100
1090
1192
1092
1072
1002
902
852
665
565
490
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes


FIFO (scan)
1400
1200
1000
800
600
400
200
0
Count
1200
1100
1097
1200
1100
1090
1200
1100
1080
1200
1100
1050
1200
1100
1025
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes


Custom(scan)
1400
1200
1000
800
600
400
200
0
Count
1197
1097
1094
1155
1055
1045
1044
944
924
857
757
707
752
652
176
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes


Rand (focus)
600
500
400
300
200
100
0
Count
504
302
299
479
289
279
462
278
258
401
238
188
293
166
91
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes


FIFO(focus)
600
500
400
300
200
100
0
Count
500
300
297
484
292
282
465
281
261
386
229
179
335
200
125
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes


Custom (focus)
600
500
400
300
200
100
0
Count
513
306
303
488
292
282
442
263
243
379
219
169
330
200
125
200
100
0
3.00
10.00
20.00
50.00
75.00
100.00
Frames
Page Faults
Disk Reads
Disk Writes

Results Explained:

We received the results we intended. Rand works alright, in the sense that it isn't terrible, but it is not the most efficient algorithm. As the number of frames becomes closer to the number of pages, the program results in fewer page faults and disk operations. This is because there is a smaller chance that the the page we faulted will not be in the page table (it is more likely to be found). This result is very similar for FIFO as well. In fact, FIFO works slightly better for a small number of frames. This is because it is no longer randomly randomly selecting a page to replace. It is instead replacing a frame that was used first. This algorithm makes it slightly more likely that the page we will need in the future will still be in the page table. And finally our custom algorithm also works better than either rand or FIFO. This is because it simulates LRU to the best extent we could conceive. The ideal scenario is one where we can see the future and kick out the entry that will not be used for the longest time. But since we cannot tell the future, we tried to catch how many times an entry faults. This means that even if the entry was the first entry entered, if the entry keeps getting faulted on then will kick out something has not been faulted on for a while.

All three of the algorithms perform the same when there are the same number of pages as there are frames. This is because no it is unnecessary to kick anything out if the pages and frames are equal, thus the zero writes to disk. Overall our algorithms performed as expected and we were pleased with out results.