Virtual Memory Analysis
CSE 30341: Project 5

Dylan Greene Borah Chong

06 April 2017

1 Introduction

The purpose of these experiments was to determine which eviction algorithm performed better under certain conditions and under different programs. Generally, this provides a good understanding of how a virtual memory system functions, as well as the trade-offs experienced depending on the implementation of eviction algorithm and workload. The eviction algorithms used were random, first-in-first-out (FIFO), and a custom algorithm which will be explained in the next section. To run our experiments, we used the student00.cse.nd.edu machine, and the command line arguments were:

```
./virtmem <npages> <nframes> <rand|fifo|custom> <sort|scan|focus>
```

where npages is the number of pages, nframes is the number of frames, <rand | fifo | custom> is the evicting algorithm, and <sort | scan | focus> is the function to run. When the command is run, it returns the value outputted by the function, the total number of page faults, the total number of disk reads, and the total number of disk writes.

For testing, we set npages to 100 and kept it constant for all of our tests. We then made 3 graphs for each program. One graph showed the total number of page faults; the next contained the total number of disk reads, and the last showed the total number of disk writes, all for 3 to 100 frames. Each graph also has 3 lines, one for each eviction algorithm. The data was obtained by creating a script that ran and reported the values to us.

2 Custom Eviction Algorithm

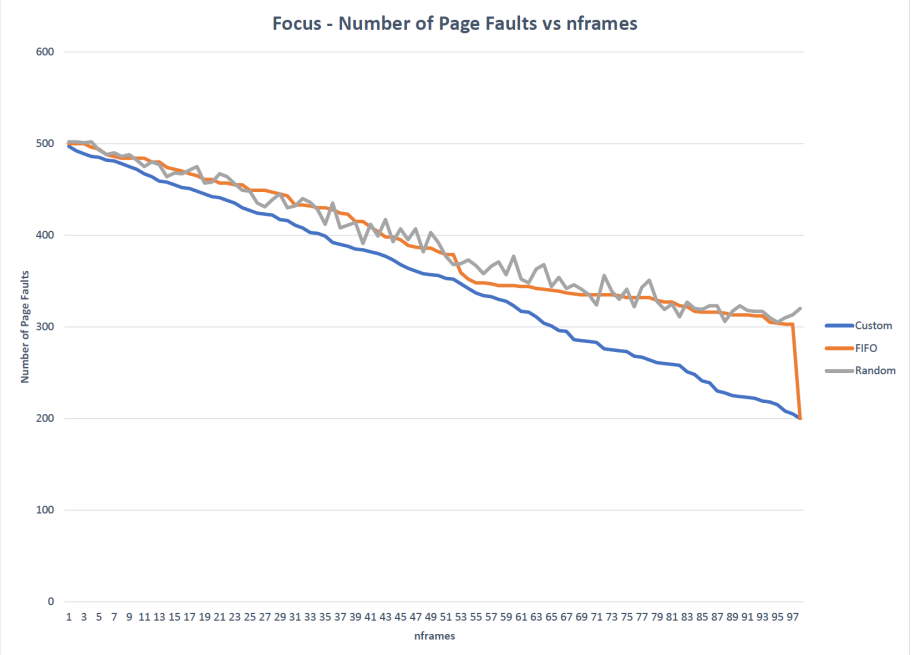
The custom eviction algorithm that we used was designed to mimic a type of most recently used eviction algorithm. Specifically, the custom algorithm fills the frames of physical memory first in order. For example, when the first page fault occurs, the data will be read and placed into frame 0 of physical memory

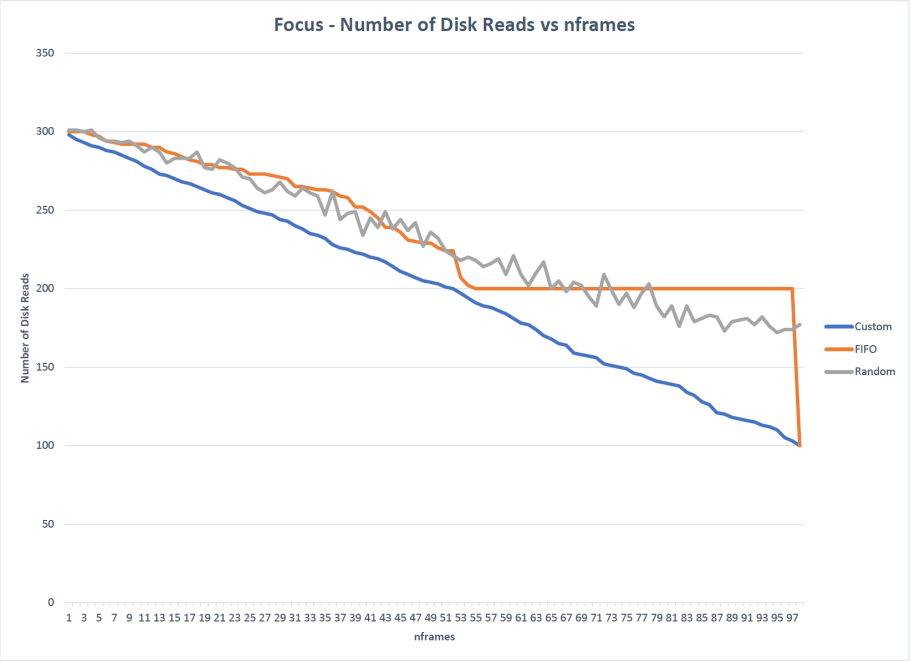
and then each next page fault will place the data into the next frame of physical memory until all frames are full. At this point, the algorithm begins to resemble the most recently used replacement strategy.

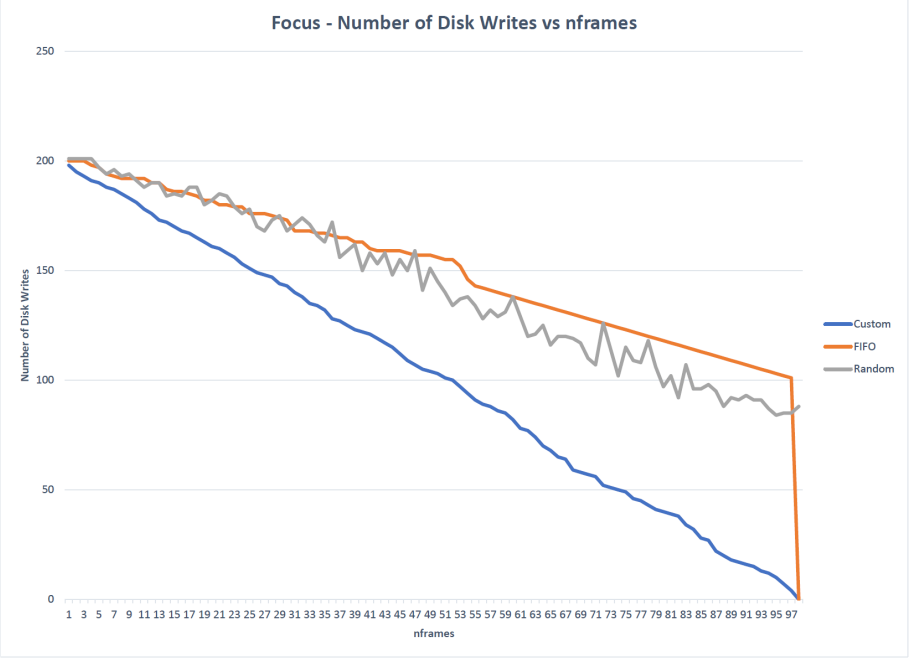
So, once all frames of physical memory are in use and a page fault occurs, the eviction algorithm will decide to evict the second to last frame. Then, when the next page fault a occurs, it will evict the last frame. It will then cycle between the second to last and last frame being evicted on each page fault. The reason for the cycling is that it allows for instructions that need to use two frames to work properly.

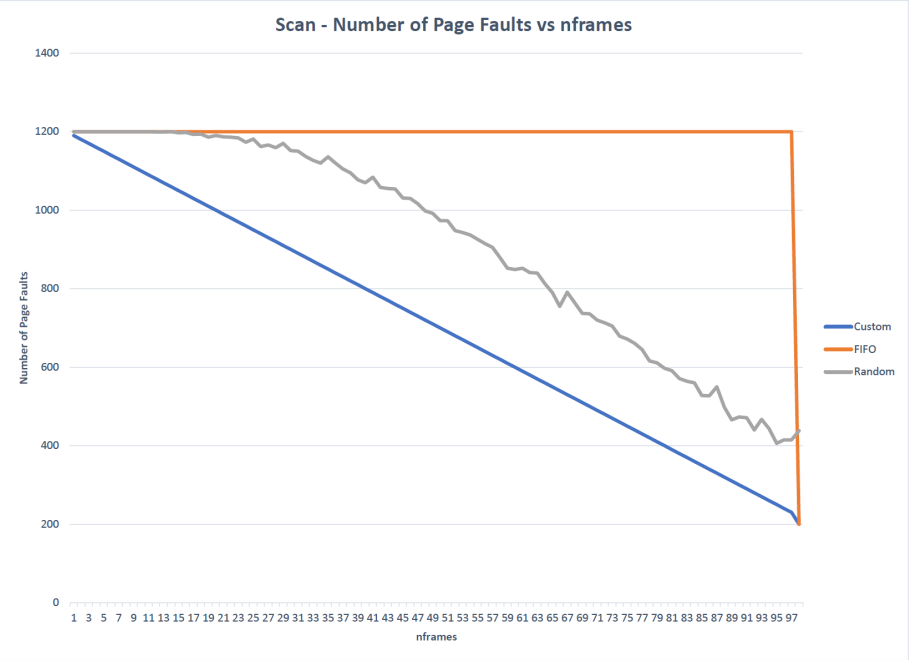
The overall idea of this algorithm was to implement an eviction policy similar to a most recently used (MRU) policy. The reason for this is that when data is being repeatedly scanned in a looping sequential reference pattern, MRU is the best replacement algorithm. Since the scan program matches this style, it was thought that this would perform very well. Additionally, the focus program has a similar to looping somewhat sequentially reference access pattern, so we thought it would also perform well there. However, since we did not have access to when a page was being accessed successfully (only when a page fault occurred), we used this method to approximate a MRU algorithm since it replaces one of the two most recently items placed into physical memory.

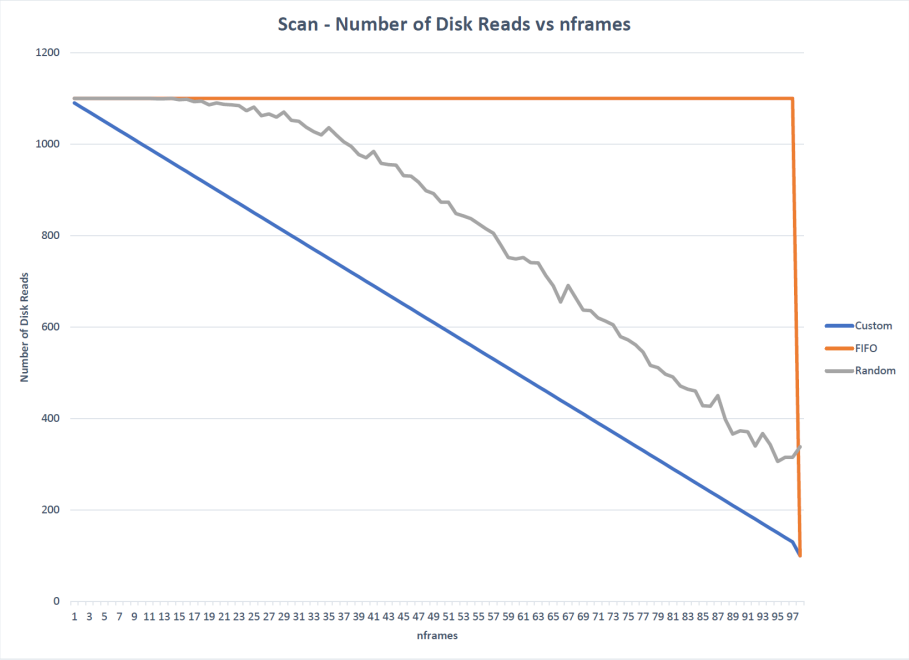
3 Results

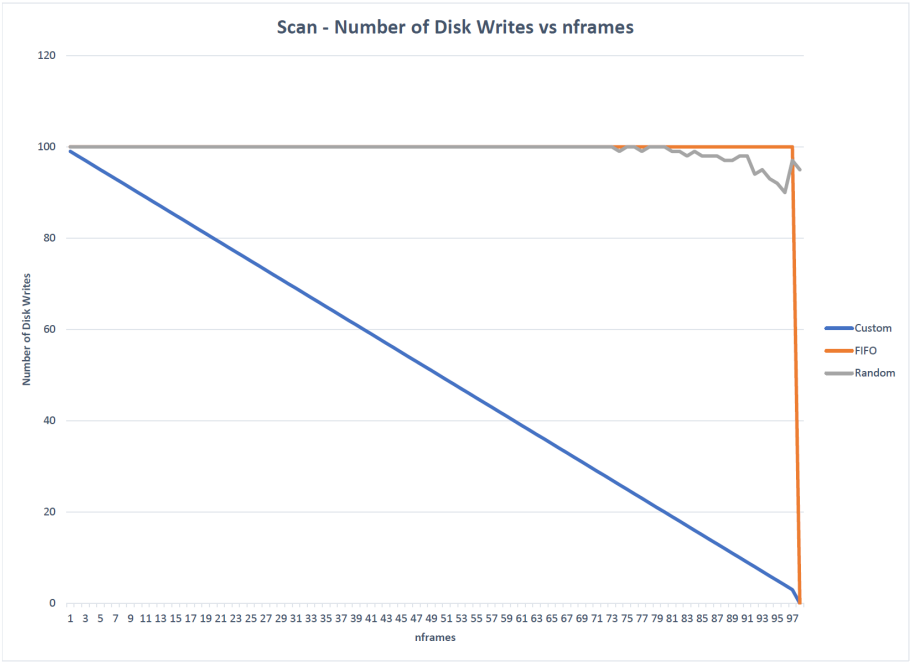


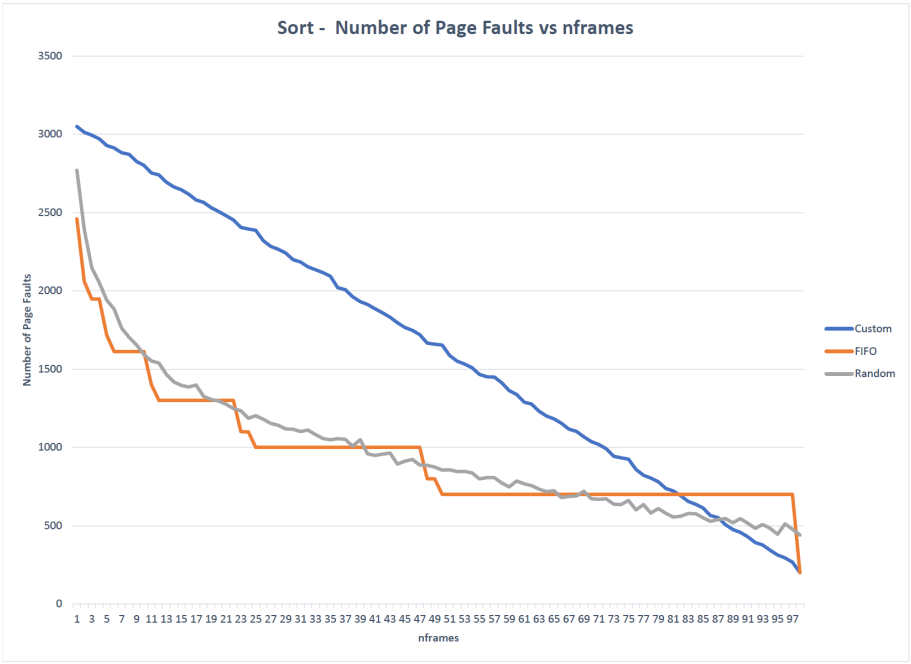

Focus - Number of Disk Reads vs nframes
Number of Disk Reads
nframes
Custom
FIFO
Random

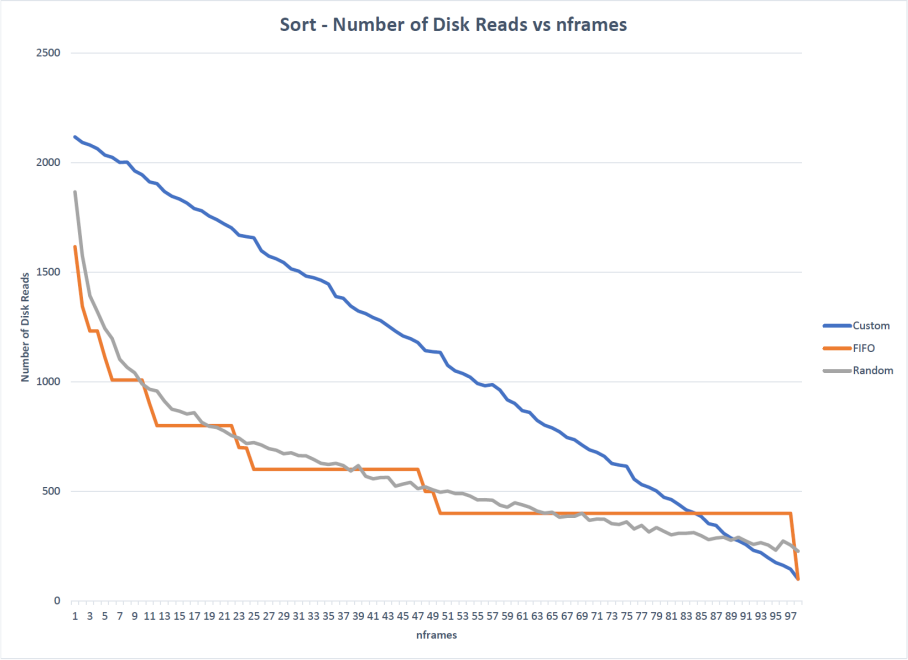

Focus - Number of Disk Writes vs nframes
Number of Disk Writes
nframes
Custom
FIFO
Random

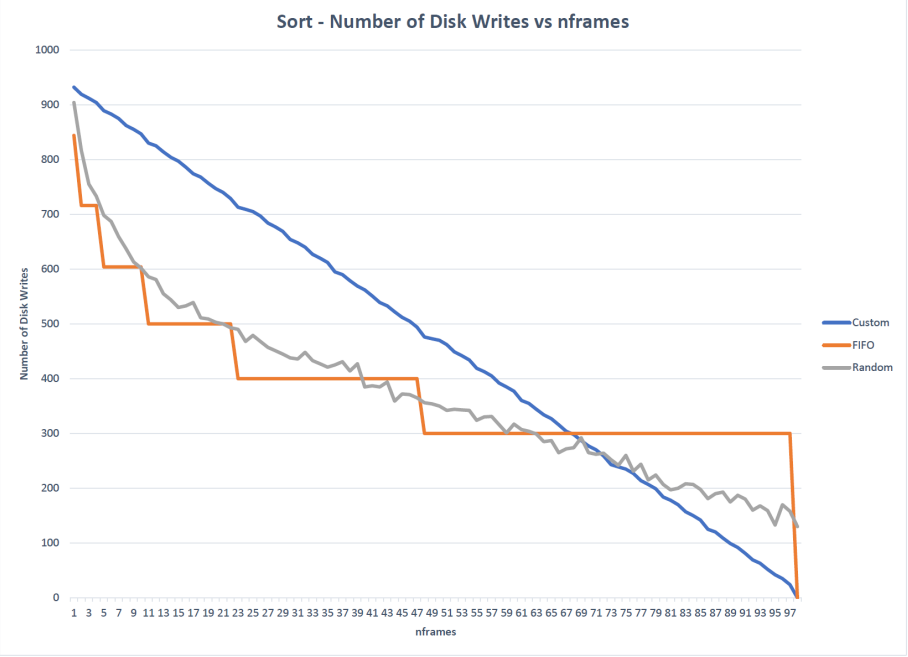

Scan - Number of Page Faults vs nframes
Number of Page Faults
nframes
Custom
FIFO
Random


Scan - Number of Disk Reads vs nframes
Number of Disk Reads
nframes
Custom
FIFO
Random


Scan - Number of Disk Writes vs nframes
Number of Disk Writes
nframes
Custom
FIFO
Random


Sort - Number of Page Faults vs nframes
Number of Page Faults
nframes
Custom
FIFO
Random


Sort - Number of Disk Reads vs nframes
Number of Disk Reads
2500
2000
1500
1000
500
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97
nframes
Custom
FIFO
Random


Sort - Number of Disk Writes vs nframes
Number of Disk Writes
1000
900
800
700
600
500
400
300
200
100
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97
nframes
Custom
FIFO
Random

4 Discussion

The results were obtained using a shell script to run over all of the different frame, algorithm, and program combinations. The raw numbers were then plotted using Excel. Looking at the graphs, it is easy to see which eviction algorithm performed best for each measurand. Basically, the best performing algorithm at a given value of nframes for each program will have the lowest value on the y axis (which is either the number of page faults, the number of disk reads, or the number of disk writes).

For the focus program, we can see that for all three measurands (page faults, disk reads, and disk writes) the custom MRU eviction algorithm performs the best for all values of nframes. The other two programs, rand and fifo, perform relatively similarly with one beating the other in certain ranges of nframes by small amounts. The focus program itself basically initializes all values of the data array to 0 in order, then focuses on a region of indices 100 times and puts a random integer into 100 random indices of the array for each time, then accesses and sums every value in the array in order. This explains the trends seen in these results. Specifically, this explains why the custom MRU algorithm outperforms the others. The custom MRU algorithm performs better do to the focusing and sequential nature of the focusing since it keeps around the older data which will result in more hits. Further, we can see that for fifo the disk reads level off, while the disk writes continue to decrease. This can be explained by the fact the number of page faults where the number of things with the write bit is decreasing while the number of evictions with the read only bit is increasing.

On the other hand, the results for the scan program are very different. In this case, we can see that the custom algorithm far outperforms the both rand and fifo. Additionally, the rand program also far outperforms the fifo eviction algorithm. Since the scan program writes a number to each array index sequentially in order and then accesses the each index (also sequentially in order) ten times, it is clear why fifo always has the same, very high number of page faults, disk reads, and disk writes until the number of frames equals the number of pages. Specifically this can be explained by the fact that fifo simply evicts the oldest page in memory. So, when the array is looped over sequentially, it will always evict the page that will be needed. This is because the only accesses to memory will always be sequential, so there will be a page fault every single time no matter what so long as the nframes is less than npages. On the other hand, custom keeps around the oldest frames so when the sequential access returns to the front, it will have hits, which is why it is so fast! Random on the other hand is somewhere in between, which makes sense since it sometimes it does not randomly evict a needed page.

Finally, the sort program puts a random value in each index, sorts the array, then sequentially sums the values in the array. In this benchmark, the custom MRU algorithm performs very poorly for all of but the highest number of frames

in physical memory. On the other hand, rand and fifo both perform increasingly well at a decreasing rate as the number of frames increases. Custom performs poorly relative to the other two because of the way that qsort operates. Specifically, qsort frequently needs the same array indices in a small time frame. Thus, recently used frames will be very important. However, custom evicts the most recent frames while fifo keeps them and rand has a pretty good chance of keeping them as well. Further, the stair-step pattern seen in the fifo benchmarks can also be attributed to the way qsort operates. This is because the stair-steps occur at the number of frames over a power of two and qsort operates using a pivot on chunks of the data so the structure of fifo does not take advantage of increased frames between those chunks.

Overall, it is clear that the best eviction algorithm depends on the work load as well as the number of frames of memory available in the system. Further, we can see that if you are able to have the same number of frames as pages, you will never need to evict resulting in the best performance.