Lauren Ferrara and Emily Obaditch
Operating Systems
Project 5
7 April 2017

The experiments were compiled by typing the command ‘make’ and run on student01. An example of running the program is:

```
$ ./virtmem 100 50 rand sort
```

Commands similar to this were used to obtain the results for sort, scan, and focus testing each algorithm (rand, fifo, and custom).

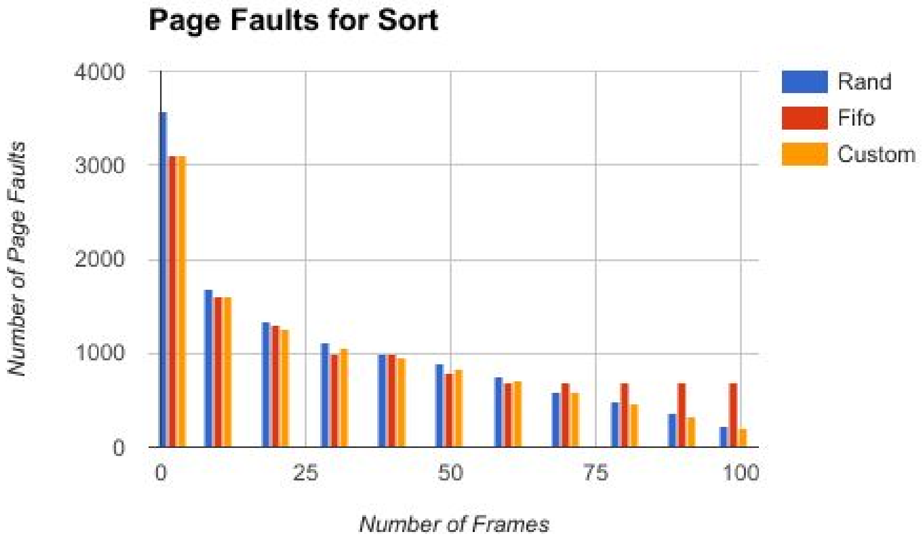
The purpose of the experiments is to compare each algorithm’s overhead of having to go to the disk to retrieve information from memory. Testing with multiple frames allows for a well-rounded data set which will make trends clear. The experiments reveal the pros and cons of rand, fifo, and the custom algorithm in terms of page faults, disk reads, and disk writes.

The custom page replacement algorithm we designed works similarly to the clock algorithm for Least Recently Used (LRU) page replacement with dirty bit consideration discussed in class and in the textbook. We started by making a global array of use bits and a global array of dirty bits, both initialized to 0 for each of the pages. Then, anytime a page fault occurs, the use bit is changed to 1 for the page sent to the page fault handler. When a page must be replaced, the algorithm iterates through the frame table starting at a random frame number. It first tries to find a frame with a page that has both use bit and dirty bit set to 0. If it checks all frames and none like that exist, it then ignores the dirty bit and only checks the use bits, setting the use bits to 0 along the way until it reaches a use bit already set to 0. That page number is then replaced, as it was not faulted on recently.

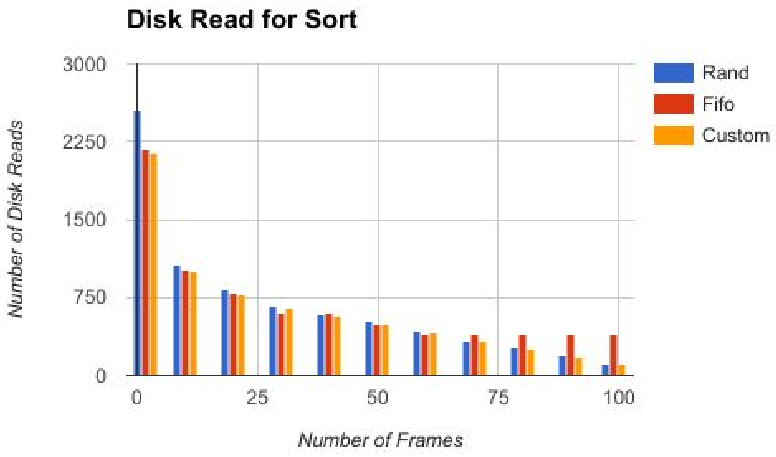
Sort:

nframes	Rand Page Faults	Rand Disk Reads	Rand Disk Writes	Fifo Page Faults	Fifo Disk Reads	Fifo Disk Writes	Custom Page Faults	Custom Disk Reads	Custom Disk Writes
2	3573	2563	1010	3116	2176	940	3103	2140	963
10	1683	1057	626	1612	1008	604	1599	1002	597
20	1348	830	518	1300	800	500	1259	778	481
30	1109	673	435	1000	600	400	1051	645	404
40	1001	592	406	1000	600	400	953	568	380
50	889	520	359	800	500	300	823	489	324
60	745	431	300	700	400	300	719	417	280
70	591	331	227	700	400	300	595	337	222

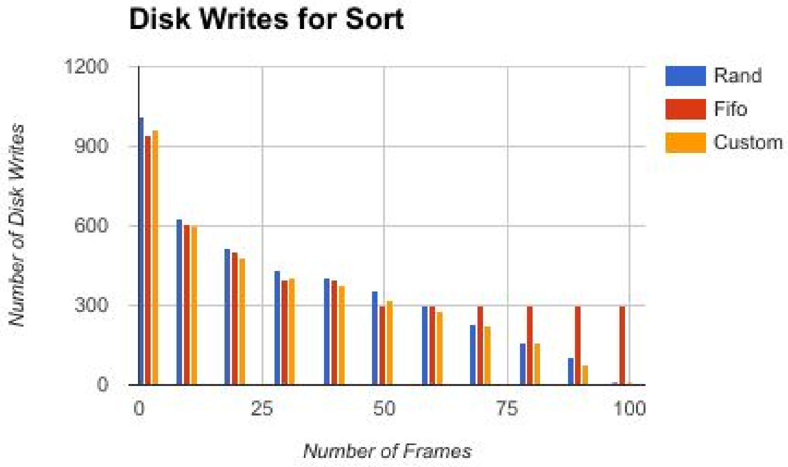
80	478	262	161	700	400	300	469	259	161
90	370	196	104	700	400	300	323	172	76
99	217	109	10	700	400	300	212	107	8



Page Faults for Sort



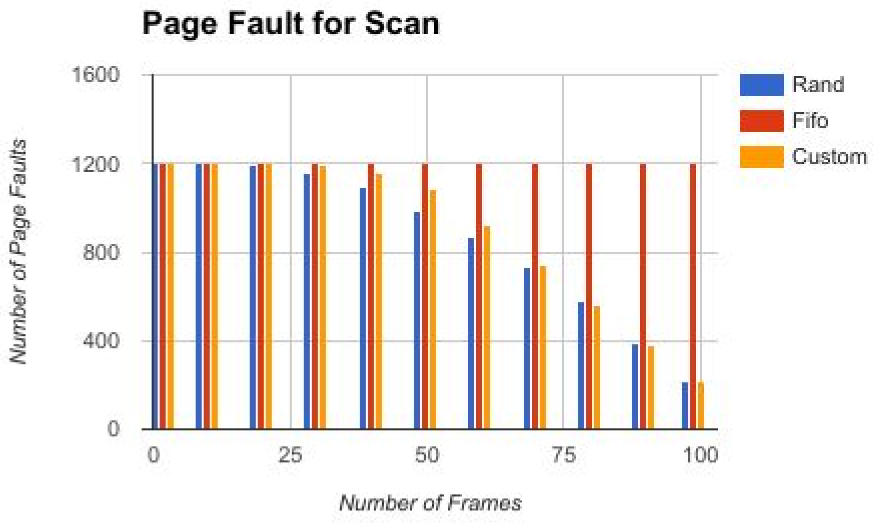
Disk Read for Sort



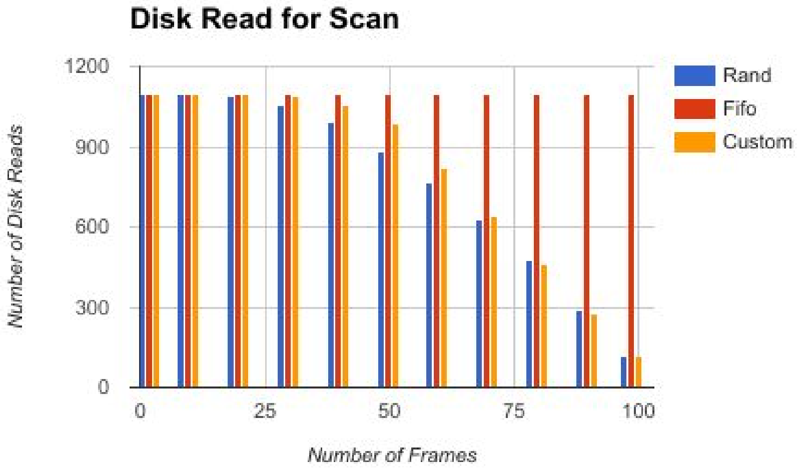
Disk Writes for Sort

Scan:

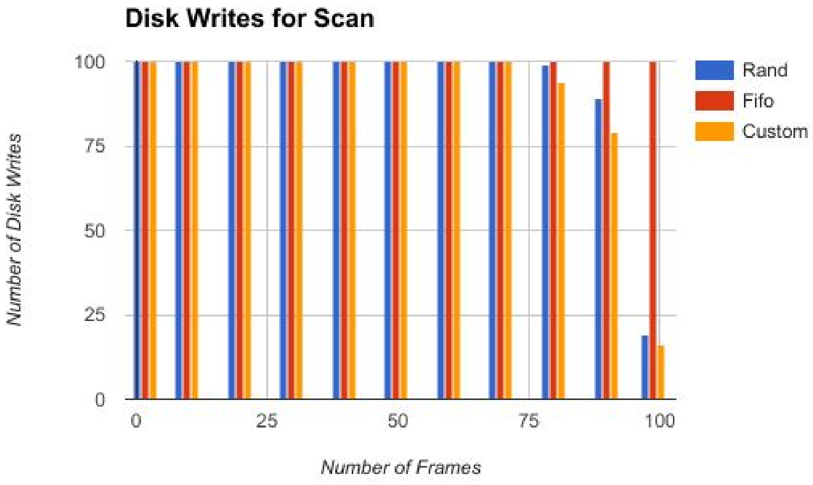
nFrames	Rand Page Faults	Rand Disk Reads	Rand Disk Writes	Fifo Page Faults	Fifo Disk Reads	Fifo Disk Writes	Custom Page Faults	Custom Disk Reads	Custom Disk Writes
2	1200	1100	100	1200	1100	100	1200	1100	100
10	1200	1100	100	1200	1100	100	1200	1100	100
20	1192	1092	100	1200	1100	100	1200	1100	100
30	1157	1057	100	1200	1100	100	1189	1089	100
40	1095	995	100	1200	1100	100	1155	1055	100
50	984	884	100	1200	1100	100	1086	986	100
60	867	767	100	1200	1100	100	924	824	100
70	729	629	100	1200	1100	100	740	640	100
80	576	476	99	1200	1100	100	560	460	94
90	391	291	89	1200	1100	100	377	277	79
99	219	119	19	1200	1100	100	218	118	16



Page Fault for Scan



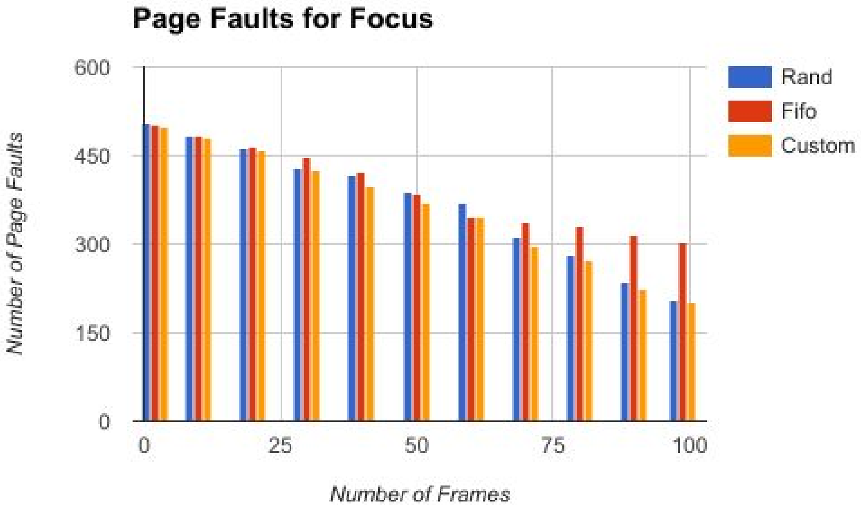
Disk Read for Scan

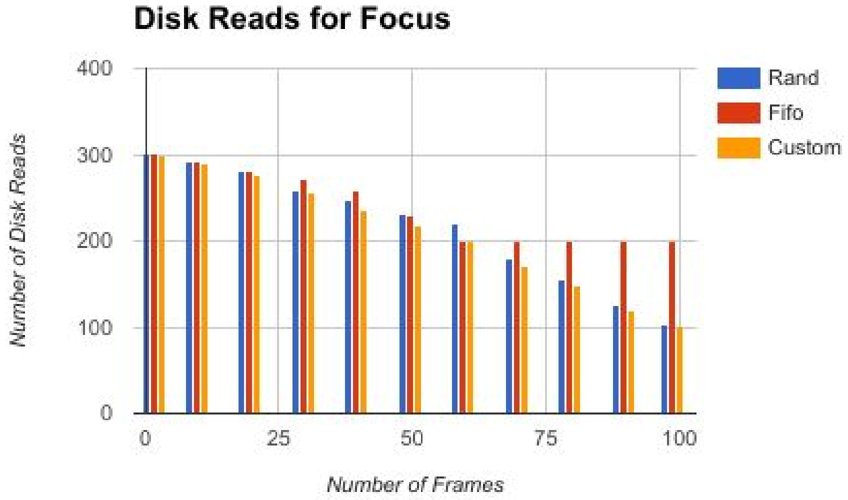


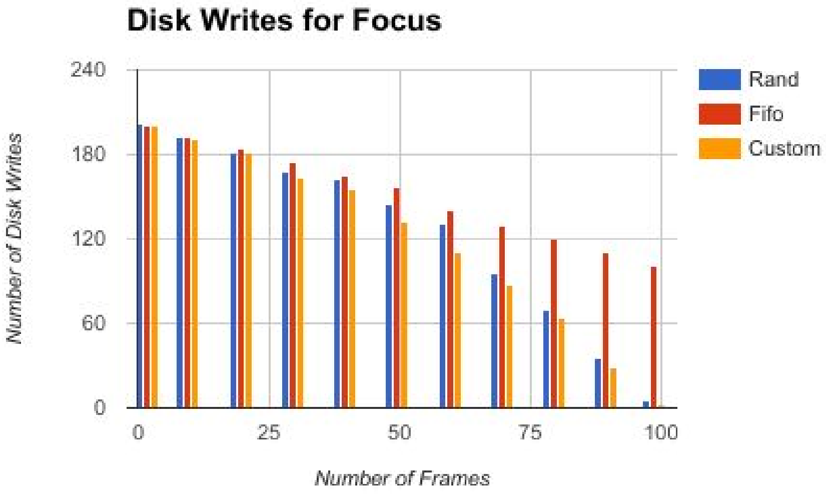
Disk Writes for Scan

Focus:

Nframes	Rand Page Faults	Rand Disk Reads	Rand Disk Writes	Fifo Page Faults	Fifo Disk Reads	Fifo Disk Writes	Custom Page Faults	Custom Disk Reads	Custom Disk Writes
2	504	302	202	502	301	201	500	300	200
10	484	292	192	484	292	192	482	291	191
20	462	280	182	465	281	184	459	277	182
30	428	258	168	447	272	175	427	257	164
40	416	248	162	423	258	165	398	237	155
50	390	231	144	386	229	157	371	219	132
60	371	220	131	345	200	140	345	199	111
70	313	180	95	336	200	130	297	171	87
80	282	155	69	332	200	120	271	149	64
90	236	126	36	315	200	110	222	119	28
99	201	101	2	303	200	101	201	101	2


Page Faults for Focus
600
450
300
150
0
Number of Page Faults
0
25
50
75
100
Number of Frames
Rand
Fifo
Custom


Disk Reads for Focus
400
300
200
100
0
Number of Disk Reads
0
25
50
75
100
Number of Frames
Rand
Fifo
Custom



The custom algorithm performed better for all three programs in terms of Disk Writes. For programs sort and focus, it was mostly better in Page Faults and Disk Reads than both rand and fifo. For sort, near 50 frames, it was very close, sometimes a little better and sometimes a little worse, to the performance of fifo for Page Faults and Disk Reads. It outperformed fifo though for frames at either end of the range. However, for scan, in Page Faults and Disk Read, it performed better than fifo but worse than rand.

The custom algorithm performed the best in each program for Disk Writes because it is the only algorithm to take into consideration whether a page is dirty or not before choosing it for replacement.

Explanation of sort program results: The sort program works by first marching through the pages, in order, reading and writing a random number to each one. Then it sorts the data through using the compate_bytes function and continues to march through the sorted pages, reading each one. In terms of page faults, rand performs the worst until number of frames is greater than 50, then performs similarly in terms of page faults to the custom algorithm, both of which perform better than fifo. When the number of frames is less than 50, fifo and the custom algorithm perform similarly in terms of page faults, both better than rand. Overall, the custom algorithm had the least amount of page faults for sort, which is due to the fact that it keeps track of both the dirty bit and used bits. Rand and the custom algorithm had similar behavior for disk reads and writes for all numbers of frames. As the number of frames increased the amount of page faults decreased due to the larger size of available memory. Fifo performed similarly to these two algorithms until the number of frames was larger than 50, fifo then plateaued at the same value no matter what the number of frames was. This could be due to the fact that the sorted memory is traversed going forward whereas the fifo algorithm pops of the beginning of the queue. This inconsistency in direction causes a larger amount of page faults than rand or the custom algorithm.

Explanation of scan program results: The scan program works by first marching through the pages in order, reading and writing each one. Then it continues to march through the pages, reading each one. This explains why fifo gets the exact same results for each number of frames when running this program. Page numbers are pushed into the queue and taken out of the queue in numerical order throughout the program. Since the program is going through memory forward and fifo is taking pages from the beginning of the queue there is no way to avoid page faulting frequently. When full, the frames will always have a series of page numbers that does not include the page number you are looking for to read. That means that it will have to write to the disk all 100 pages that were written to, because each will be evicted and read from disk every other time a page fault occurs. One way to fix this poor performance would be to implement a lifo algorithm instead of fifo. This way, the program would go through memory in the same direction that pages are being popped off the queue.

Explanation of focus program results: The focus program begins by marching through the pages, reading each and writing a 0 to each page. Then, the program looks at a certain chunk of memory and accesses random pages within that chunk 100 times, and then moves to another chunk of memory and repeats the random accesses. This program does this 100 times. After, the program marches through each page and reads from it. This results in both spatial and temporal locality increases as the program is only accessing pages within a certain range each time. This explains why the number of page faults, disk reads, and disk writes for the focus program were significantly lower than the results from sort and scan. For example, the maximum number of page faults that occur when focus is run is around 500 where as sort is around 3000 and scan is 1200. As the number of frames increased the resulting page faults, disk reads, and disk writes decreased for each algorithm. For each measured quantity (page faults, disk reads, and disk writes) fifo performed the worst, followed by rand, followed by custom. The custom algorithm performs the best because it makes the most use of temporal locality with LRU approximation.