Nicholas LaRosa
CSE 30341 Project 5
Evaluation Report

Experimental Purpose

The purpose of this experiment was to determine the most successful page replacement algorithm based on the number of page faults, disk reads, and disk writes that occurred when each of three algorithms (random, first-in first-out, and custom) were applied to each of three programs (focus, sort, and scan). Comparing algorithm performance across the three programs, we can also determine the effect of program behavior on algorithm performance and whether or not one page replacement algorithm is better suited for one program, while a different algorithm is better suited for another. In other words, one algorithm may not be the overall best at minimizing page faults, disk reads, and disk writes, but rather, one algorithm may result in the best performance of the scan program, while another algorithm may result in the best performance of the sort program, for example.

Experimental Setup

Each of the three page replacement algorithms were run with each of the three programs, resulting in nine data sets. Using a Python script (test.py), the nine combinations of commands were executed on student00.cse.nd.edu with page table sizes 2 to 100, inclusive. In all cases, the disk size was 100. This means that the command in each test case was "./virtmem 100 <page_table_size> <algorithm> <program>". Each of the nine combinations was written to its own CSV file, named <algorithm>_<program>.csv, and were stored in the test_results/ folder. With each of the nine combinations executed with 99 different page table sizes, there were a total of 891 test cases. All nine graphs (one for each combination) were graphed using Microsoft Excel.

Custom Algorithm

Because the given API does not allow us to track page accesses during program execution, but only page faults, I decided to implement a page replacement algorithm in which frames are given a second chance in the frame table after being read from disk and after being modified. Building onto the frame table I built to keep track of free frames and the pages mapped to each frame, another variable "givenSecondChance" was initially set to false for each frame. In order to select the next frame to be replaced, this algorithm loops through the frame table (as in FIFO) and selects the first free frame or the first frame that has already gotten a second chance (ie. the first frame whose "givenSecondChance" frame table struct variable is set to true). If a frame is passed by without replacement, its "givenSecondChance" property is set to true and thus will be a candidate for replacement during the next page fault. Because the frame table begins as empty, this algorithm by itself would be no different

than FIFO. However, I added functionality that gives a frame a second chance before replacement if it is modified. In other words, if the page fault handler is called and the respective page has already been read from disk, we know that we are actually handling a page modification. Thus, within my algorithm, the frame mapped to this page is given yet another chance at survival. This algorithm thus relies on the idea that highly-modified frames are more likely to be accessed by the program and thus should not be replaced right away. Once a frame is modified, this algorithm ensures that it remains mapped to the page table (ie. able to be accessed/read) by the program until at least two more page faults occur (at which point it has used up its second chance if it has not been read again).

Execution Results

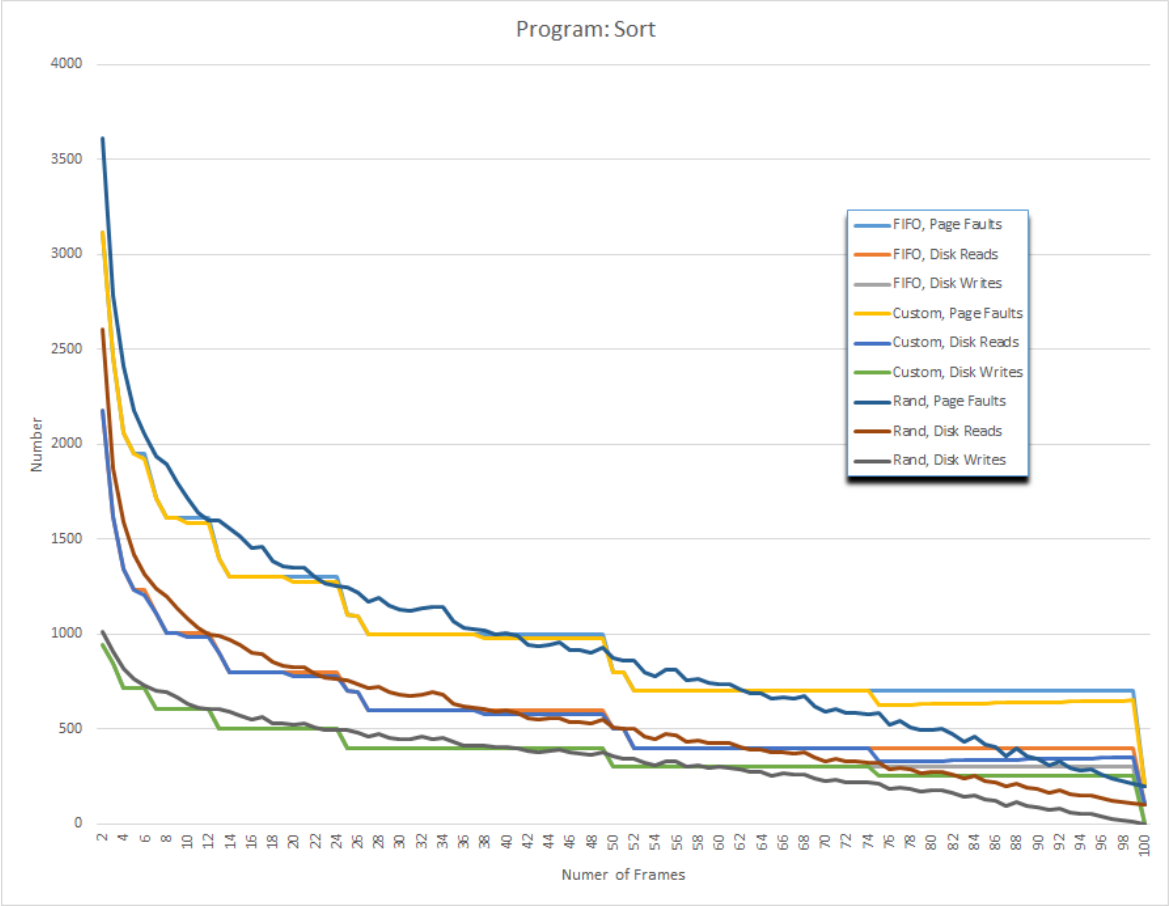

Figure 1: Page Faults, Disk Reads, and Disk Writes during execution of Sort

As we can see in Figure 1, during the execution of the Sort program, the Custom algorithm resulted in fewer page faults, disk reads, and disk writes than FIFO. However, both the Custom and FIFO algorithms performed worse than the Random page-replacement algorithm, but only during certain frame number intervals. The Custom algorithm performed better than FIFO because modified

pages were given another chance at survival during the frame table eviction process, allowing heavily modified frames to remain paged and preventing accesses to these pages from causing page faults. During some intervals, however, the Random replacement performed better. This is likely due to the fact that many times the Rand algorithm will, by chance, choose to evict frames that are least recently used. However, we were unable to count the page accesses to provide LRU support.

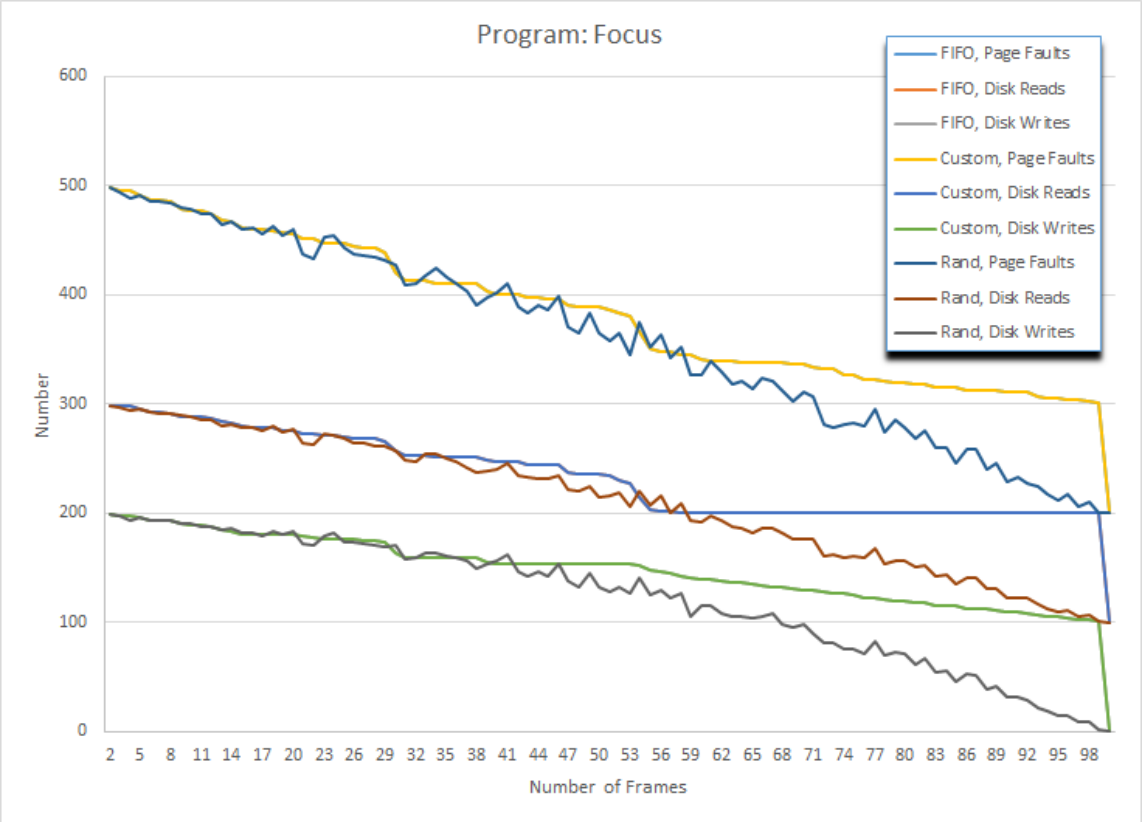

Figure 2: Page Faults, Disk Reads, and Disk Writes during execution of Focus

As we can see in Figure 2, during the execution of the Focus program, the Custom algorithm resulted in the same number of page faults, disk reads, and disk writes as did FIFO. Both the Custom and FIFO algorithms performed worse than the Random algorithm, but only at a high number of frames. The Custom and FIFO algorithms perform similarly in the case where all frames in the frame table are modified one after the other, leaving all frames in the frame table with a second chance at survival. Subsequently, we know that when the entire frame table is given a second chance, the Custom algorithm degenerates into FIFO. Again, the Random algorithm likely performed better due to its by-chance ability to choose least-recently-used frames for replacement better than FIFO. We can see that the Random algorithm performed better (relative to FIFO) as the number of frames increased; this means that least-recently-used frames are more quickly evicted if randomly chosen, as opposed to being the next frame in a larger FIFO queue.

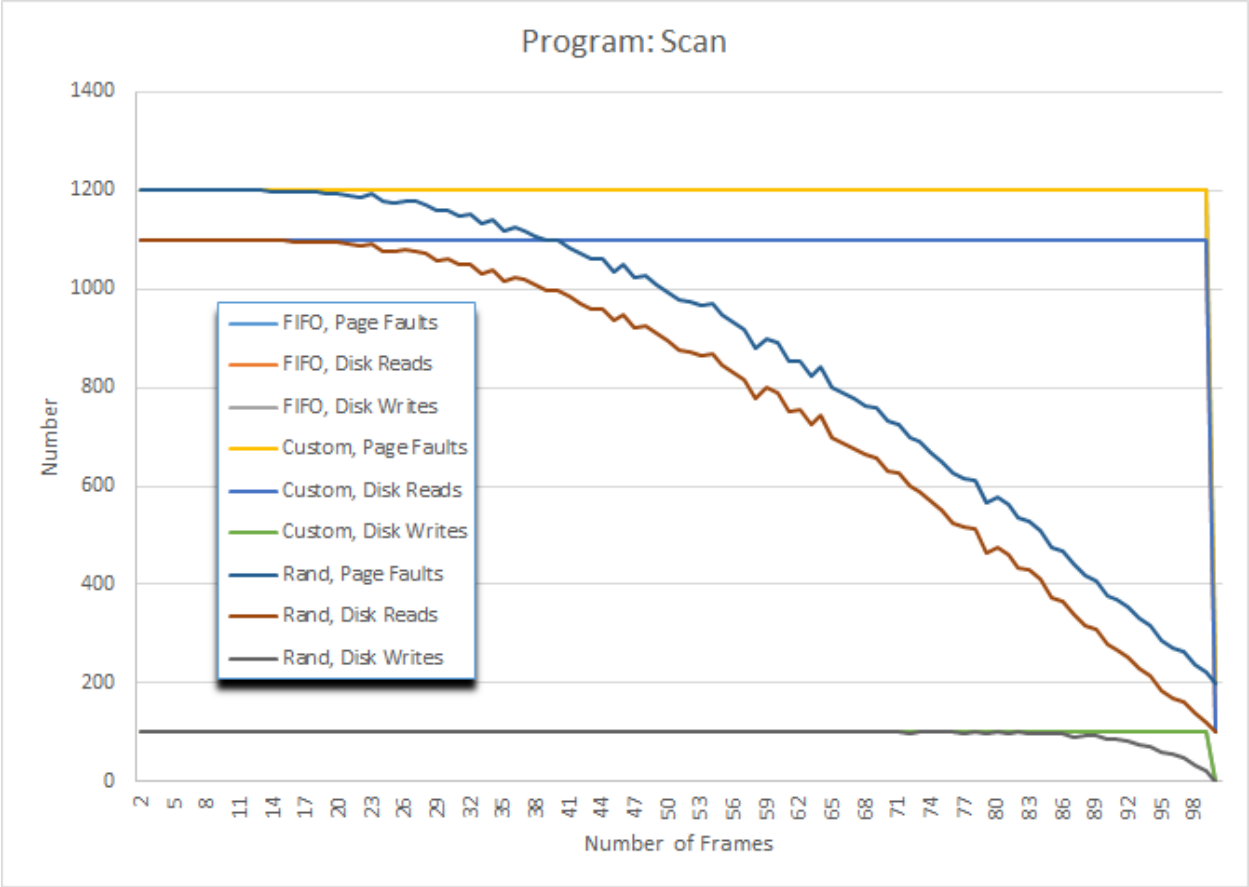

Figure 3: Page Faults, Disk Reads, and Disk Writes during execution of Scan

As we can see in Figure 3, during the execution of the Scan program, the Custom algorithm resulted in the same number of page faults, disk reads, and disk writes as did FIFO. Both the Custom and FIFO algorithms performed worse than the Random algorithm, but only at a high number of frames. The Custom and FIFO algorithms perform similarly in the case where all frames in the frame table are modified one after the other, leaving all frames in the frame table with a second chance at survival. Subsequently, we know that when the entire frame table is given a second chance, the Custom algorithm degenerates into FIFO. Again, the Random algorithm likely performed better due to its by-chance ability to choose least-recently-used frames for replacement better than FIFO. We can see that the Random algorithm performed better (relative to FIFO) as the number of frames increased; this means that least-recently-used frames are more quickly evicted if randomly chosen, as opposed to being the next frame in a larger FIFO queue.