Madeline Gleason
Reilly Kearney
Project 5 Report
10 April 2017

Project 5 Report

Purpose

For this project, we measure the number of page faults, disk reads, and disk writes for each program and each page replacement algorithm. As explained in the project document, we start with a linear search to find an open frame in our frame table. If there is no open frame, our page_fault_handler selects the appropriate algorithm to use to determine which element is evicted. We collect our results for each program and replacement algorithm by running our virtmem executable. We used 100 pages and frame numbers varying from 3 to 100 on the student machines on campus. Our command line arguments were as follows:

1. <./virtmem 100 3-100 rand scan>
2. <./virtmem 100 3-100 rand sort>
3. <./virtmem 100 3-100 rand focus>
4. < ./virtmem 100 3-100 fifo scan>
5. <./virtmem 100 3-100 fifo sort>
6. <./virtmem 100 3-100 fifo focus>
7. <./virtmem 100 3-100 custom scan>
8. <./virtmem 100 3-100 custom sort>
9. <./virtmem 100 3-100 custom focus>

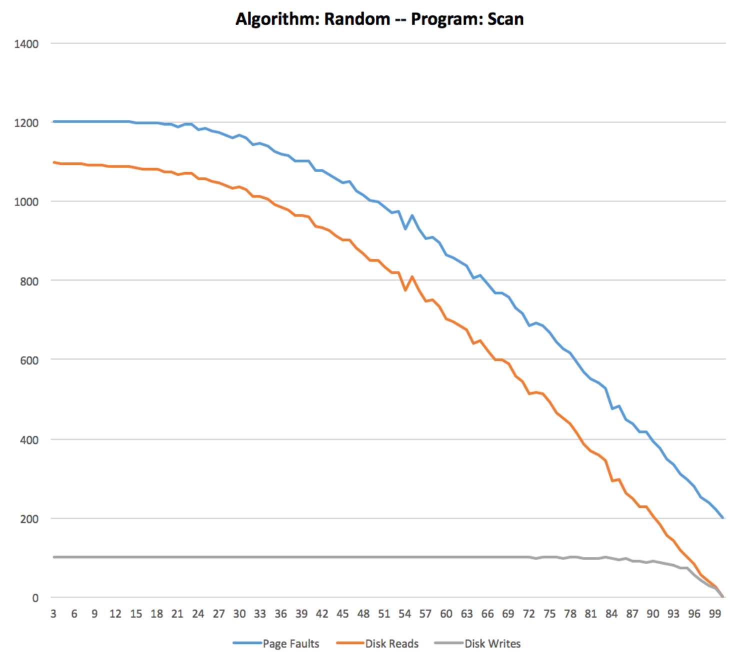
Each of these commands returns the corresponding number of page faults, disk reads, and disk writes for the command, which we measure and analyze in subsequent sections of this report.

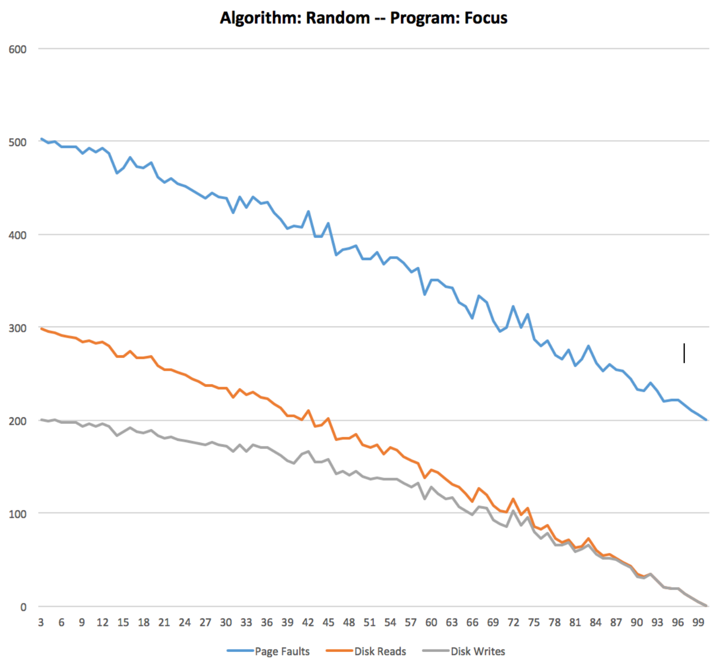
Custom Replacement Algorithm

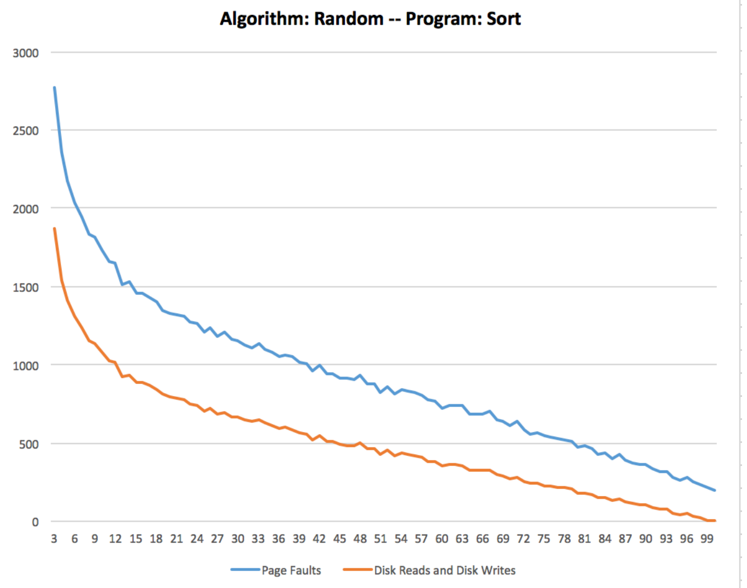
We used a Second Chance Page Replacement policy for our custom algorithm. To do so we created a second frame table, FRAME_TABLE_CHANCES, indexed by the frame number with a corresponding “second chance” bit value of 0 or 1. Each time a frame is referenced the second chance bit is set to 1, giving the frame a “second chance” before eviction. When a new page is read into a memory frame the second chance bit is set to 0. To complete an eviction, we iterate through the frames in a manner similar to our FIFO algorithm. If the second chance bit in FRAME_TABLE_CHANCES is equal to 1, we reset the bit to 0 and continue. If the second chance bit is 0, we evict and replace that page.

Measurement and Graphs

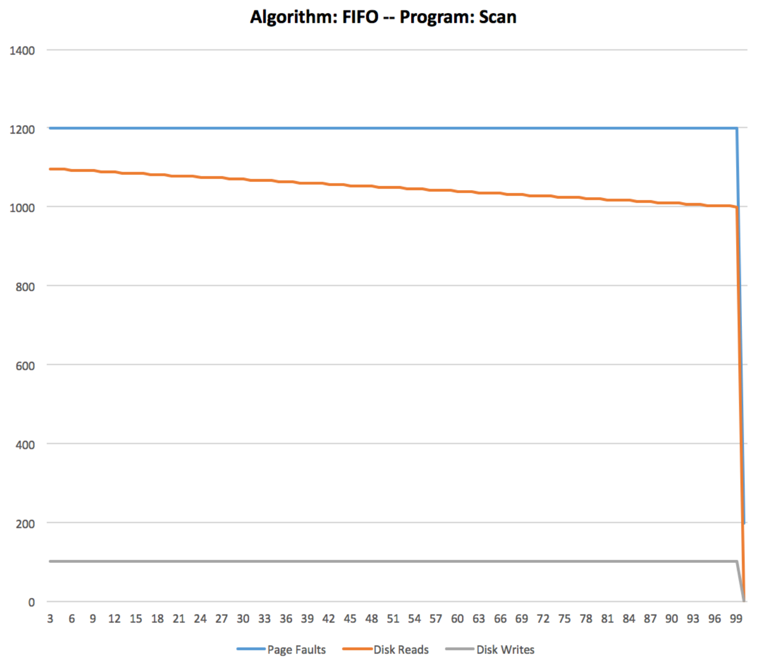
Algorithm: Random

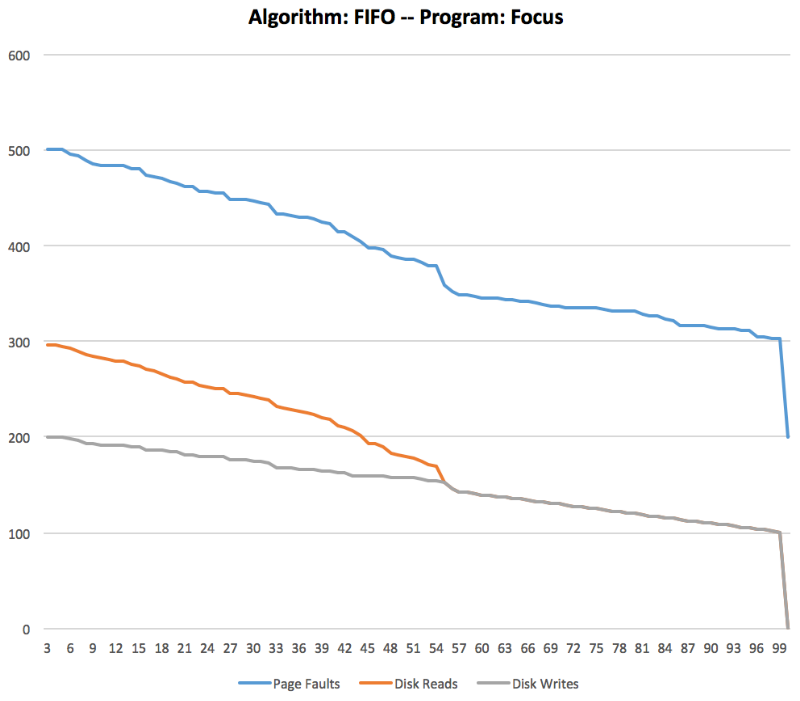


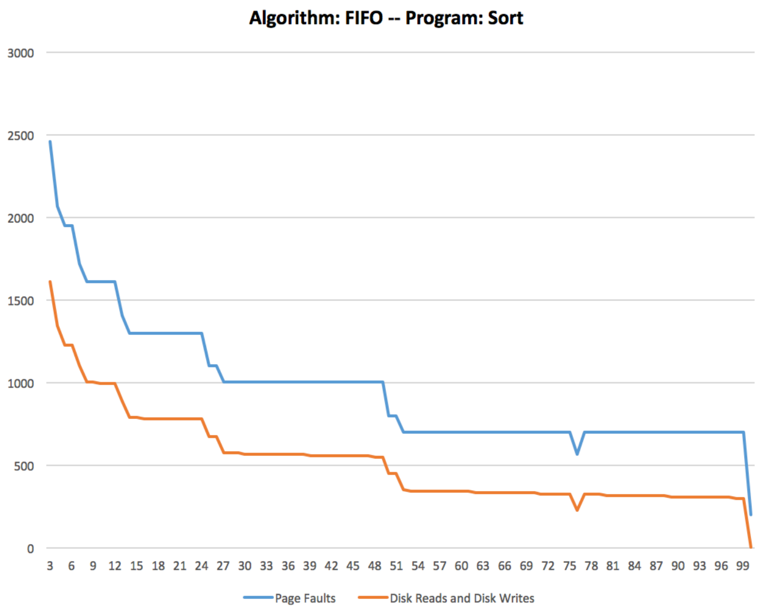




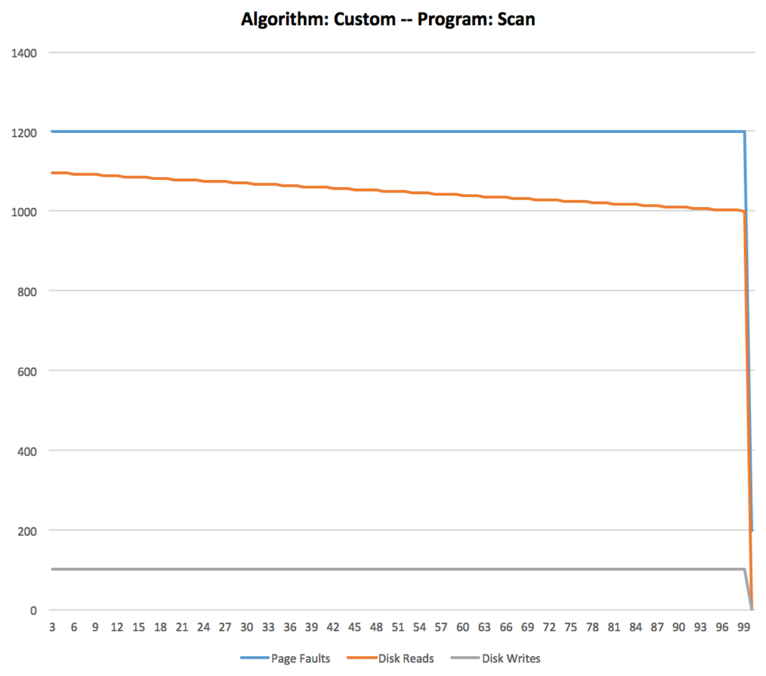
Algorithm: FIFO

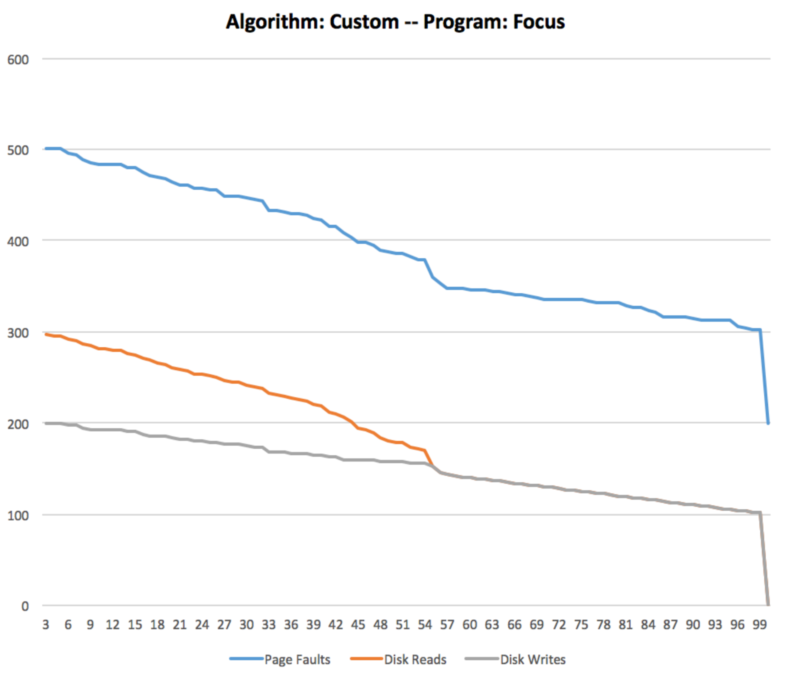


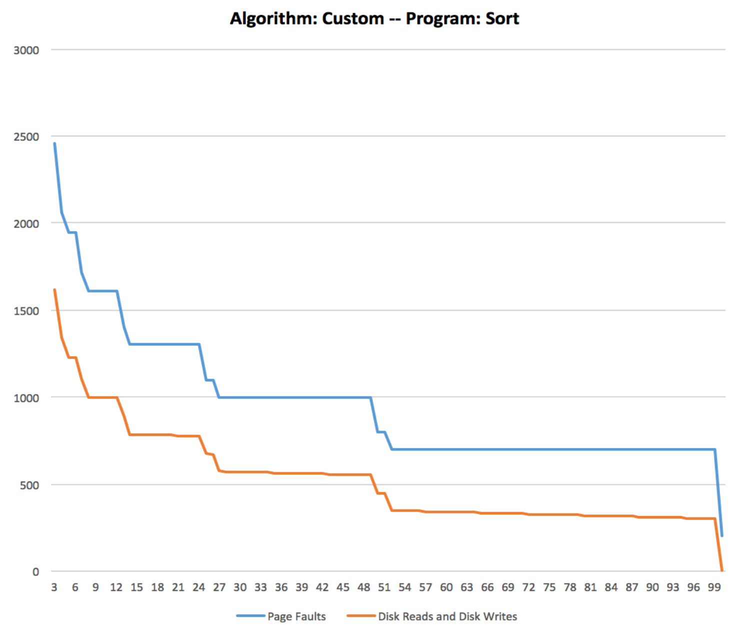




Algorithm: Custom







Results

Given the measured results above, it is obvious our measurements are incorrect for each algorithm. However, we did our best to implement our FIFO algorithm and Custom algorithms correctly for this project. For our results analysis we will compare our actual results to our assumed theoretical results.

Given our actual results, it would appear the random algorithm performs the best in evicting elements from our frame table. It performs most consistently and page faults, disk reads, and disk writes decrease as the number of frames increases. Theoretically we believed our custom algorithm, second chance page replacement, would perform the best under all circumstances as it prioritizes elements that are constantly accessed unlike random or FIFO.