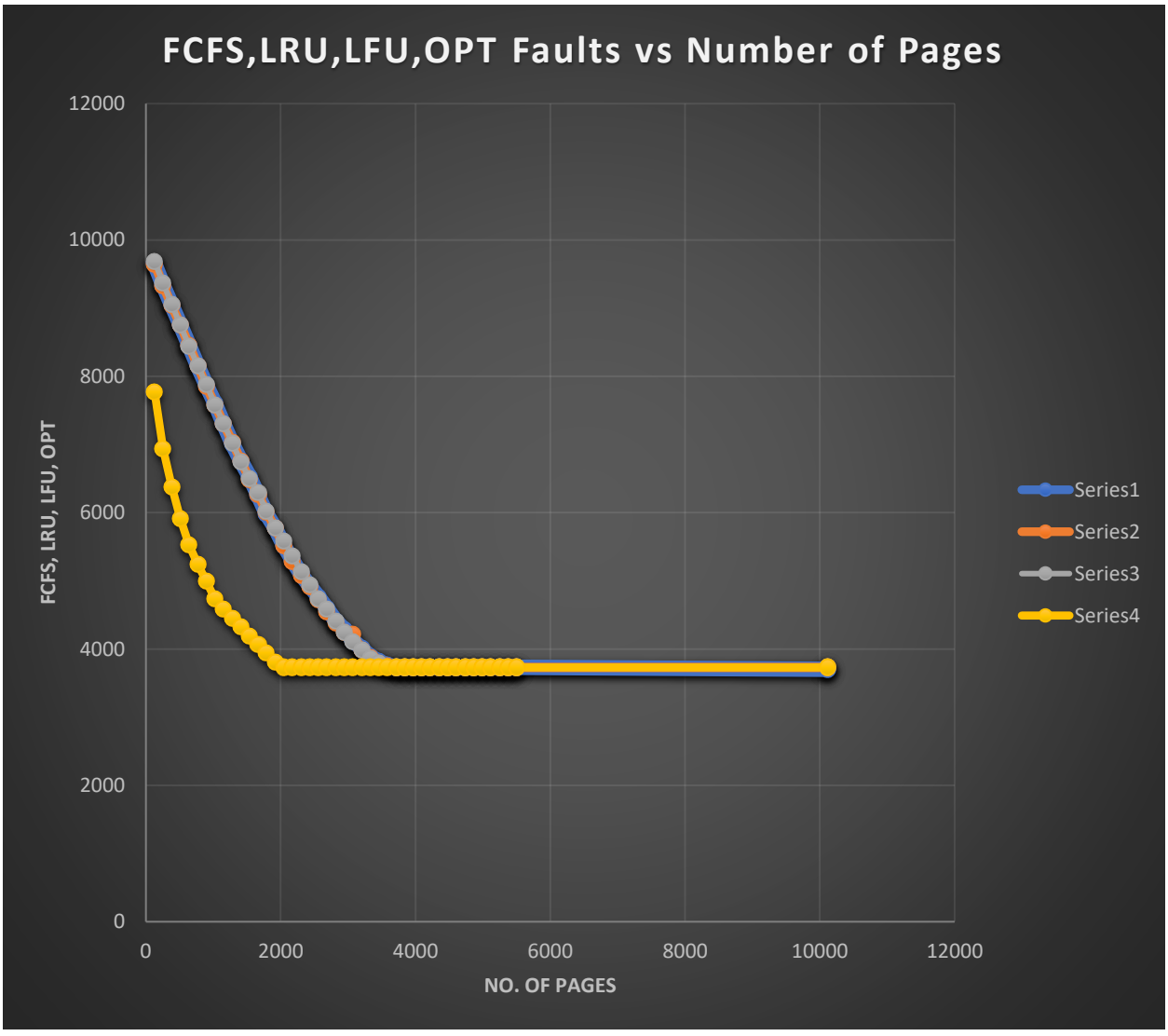
Chase Brown

Operating System Concepts Lab-03

Simulating Page Replacement Algorithms

The primary goal of this lab was to program four algorithms (First Come First Server, Least Recently Used, Least Frequently Used, and Optimal), and compare their performance against each other. Performance is based on the memory management by the number of page faults for a given size of memory.



The graph data was generated by taking the page faults of each algorithm every 128 pages from 128->5504. I stopped collecting data at 5504 pages because the algorithms had all become linear at this point. Optimal can become linear much quicker because it looks to see if a process is ever used again in the future. If it is not, then that process is replaced for the new one coming in. For FCFS, LRU, and LFU because of how many unique processes are in the stream of data, if there are 10,000 process references but only 3730 unique processes then when you hit 3730 pages all the processes are in the memory, but

all processes are unique and therefore create a page fault when entered in the memory. These are unavoidable faults created the first time the process is loaded into memory.

When comparing the data points based on the data and graph FCFS, LRU, and LFU appear to have the close to same data points with Optimal being the only algorithm breaking away from the trend. Eventually as the page numbers increase the other algorithms increasingly close the difference between their faults and Optimals. This is simply due to there being more open memory pages for 'new' processes to be loaded into, which will eventually get large enough to hold all the unique process in the data. When comparing the data, we see that eventually it doesn't matter what algorithm we use we will still have the same number of page faults. If we don't care about memory and we can have all the processes for a program loaded into memory that would be a perfect world. This would increase the program speed because there would be no waiting for deciding which process to kick out and fetching data and it would be ready to go in memory when called. The reality is that we are dealing with a limited memory size for most cases. Being able to decrease the number of page faults is a key to increasing speed and efficiency. Optimal in most cases besides on paper is not realistic. The difference between the other three algorithms when ran on a large data set is minimal. The programmer would need to consider the time it takes to search the memory to find which process to replace for the LFU and LRU algorithms while FCFS can simply kick out the first process and replace.

In conclusion the Optimal algorithm beats out FCFS, LRU, and LFU. The difference between FCFS, LRU, and LFU are so close its hard to determine a real 'winner', and the results can be determined by the data set that is being ran. Eventually all the algorithms become linear if the memory size is increased to a size large enough to hold all the processes creating only the initial page fault.