**Program 3 Report- LRU, FIFO, Clock Average Page Fault Comparison**

Name: Adwaith Moothezhath Rajesh

Date: April 25, 2021

NetID: axm180043

CE 4348.001

**Problem Statement**

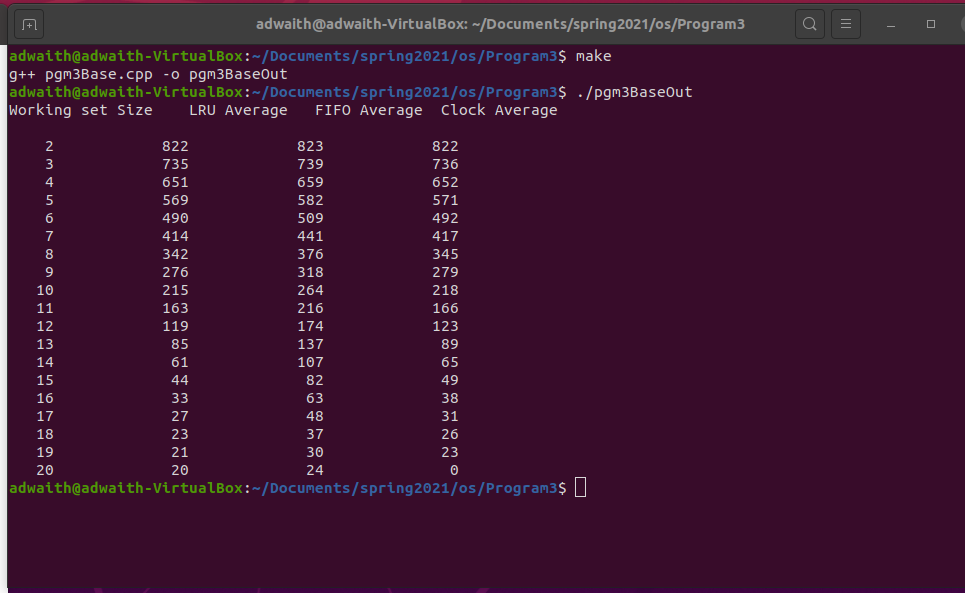
The assigned task was to compare the number of page faults for LRU, FIFO, and Clock algorithms. For all three algorithms, the working set sizes of 2 to 20 should be displayed. This experiment should be run 1000 times using a Poisson distribution and a lambda value of 10. The code should output the average number of page faults for each of the working set sizes. At the end plot, the average number of page faults for each of the working set sizes on a graph using an external program to compare the algorithms.

**Approach**

The primary approach was first using a Poisson distribution with a lambda value of 10 to get 1000-page traces and store them in an array. Next for each working size set between 2-20 inclusive call the three functions LRU, FIFO, and Clock. This should return the number of page faults for each working set size 1000 times. The page faults for each working set size for the three algorithms LRU, FIFO, and Clock can then be outputted at the end. Finally, by using excel I can graph the data of the average number of page faults that occurred for each of the working set sizes for the three algorithms. The language used for this program was C++ and the environment I ran the code was on Ubuntu. I made a make file to run the C++ program on Ubuntu 20.04 LTS environment.

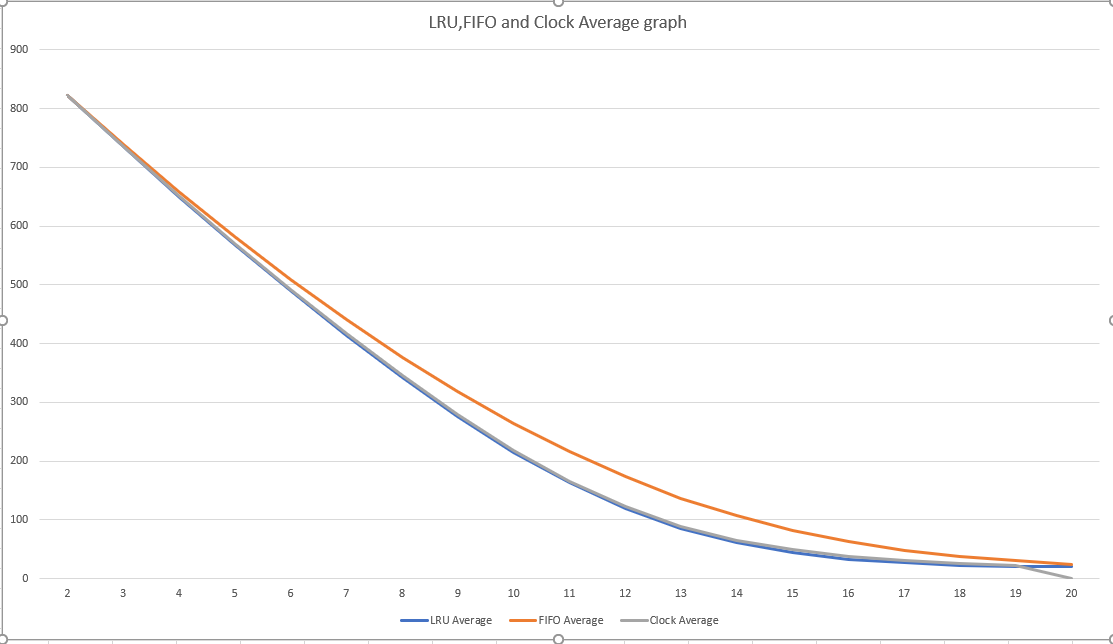
**Solution**

I wrote the program in C++ in an Ubuntu 20.04 LTS environment. First, I generated the random Poisson numbers using lambda = 10. Then I stored all those pages in an array and made counters for LRU, FIFO, and Clock also I ran the experiment 1000 times. I then ran the three counters for the working set size of 2 to 20 and called the functions, respectively. For the LRU function, I used a vector to do the least recently used algorithm. When a new number is added to the vector a page fault counter is incremented. If the same page is in the working set then no page fault occurs but if the working set is full and a new page is added the least recently used page is deleted and a new page is added to the end of the vector. Also, the page fault is incremented. Also for the LRU function if the same page is next to be added to the working set then it deletes the LRU of the page and puts that page at the end of the vector which is the same as swapping them. In the FIFO function, the same thing happens as LRU except that this algorithm is First in First out and it does not need to delete if there is the same element in the working set and add to the end of the vector. The clock algorithm has two arrays one is for storing the pages and one is for having a Rbit set for each page. The RBit represents a second chance as if the pointer points to a page with an RBit of 0 then it replaces that page if the working set is full. If the Rbit to a page is set to 1 and the pointer is pointing to it and wants to be replaced it sets the Rbit to 0 and skips that page and replaces the next page which has RBit set to 0. All these functions return the page faults and at the end, I output the average page fault for each working set size and the three algorithms as shown in Figure 1 below. I also graphed the average of page faults of the three algorithms for all the working sizes between 2-20 like Figure 2 below. This C++ program was compiled and ran on Ubuntu 20.04 LTS environment using a make file like shown in Figure 1 below.



**Figure1: Output Screenshot**

The program output displays the working set size from 2-20 , LRU averages, FIFO averages and Clock Averages. To compile the program, I used g++ and put it in a make file. To run the code, I typed the command **./pgm3BaseOut**.



**Figure2:** **LRU,FIFO and Clock Average graph**

The program graphs the LRU,FIFO and Clock Average line graph for each of the working set sizes from 2 to 20.

**Appendix**

To run the program first make sure to have Ubuntu 20.04 LTS environment. Next, have the build-essential package downloaded. Next, download the pgm3Base.cpp and Makefile and put them in the same directory, and run the command **make** in the terminal. Next, run the command **./pgm3BaseOut** and now the output will be displayed just like the output screenshot above.