/\* Quoc Lien (816097211)

\* Masc1434

\* CS570, Summer 2014

\* Assignment #2, Page Replacement Simulation.

\* README.txt

\*/

Included files:

A2.h

A2.cpp

Makefile

README.txt

pages.txt

::Compile Instructions::

To compile this project, go to the directory where the source code is located. It is located in a folder called a2. Change directory to this a2 folder. Once there, type Make to compile the source code to output an executable called a2

Example.

% Make

Make clean will remove all object files.

::Operating Instruction::

This program requires one command line argument. Frame Size, Where Frame size is how many frames a memory system has. To run this program, type a2 followed by the size of frame.

Example.

a2 3

:: Design decisions ::

When programming this program initially, I wanted to use multiple different data structure for each algorithm, but that seems inefficient. Instead I used a single vector of structs. The struct I’ll be using has 3 variables; Page number, Use bit(for clock), and A pointer to the next page(for clock). When running the LRU, the algorithm only uses the page number. During Clock, I built a linked list using the same data structure, instantiating its pointers and use bits. OPT algorithm uses the same data structure and calculates the displacement from its current position and it determines which page should be swapped.

:: Extra ::

Print\_frame(), prints the current frame.

Print\_list(), prints the string of number.

Create(), creates a file name pages.txt with randomly generated integers.

:: Deficiencies/Bugs ::

I didn’t run into any bugs while testing the project. Everything seems to be working correctly.

:: Lessons Learned ::

What I learned during this project was how different each algorithm acted differently when given different frame size and input strings. And what I learned most importantly is memory management systems, how each memory system can use different types of frames and algorithm. The effects that each frame size and algorithm has on the memory system may hinder or increase the memory systems performance.

**Summary Analysis**

**Control –**

To start my analysis, I started off with a small control or base test. This was conducted using a list of 100 numbers from 0-9, and 0-24. I’ll be using this data to compare the data set in a scaled up test.

Variable—

The variable used in this small scale test was 3 frames, 5 frames, and 10 frames for the memory management system. Eventually leading up to a scaled up version of this small test using 15 frames, 20 frames, 25 frames, and 50 frames with 2 different data sets.

Outcome—

The outcome of this small scale test shows that with the LRU algorithm, it starts off slow with more faults than the clock algorithm. Eventually beating the clock algorithm in the higher frame test. With the clock algorithm, it starts out faster than LRU algorithm (although minor) beating it very slightly in the 3 frame test by throwing less faults then the LRU. It eventually loses in performance after the 5 frame test and 10 frame test. In this small scale test, the Optimal algorithm seems to perform the best, and its performance can be increased linearly as we increase the frame size.

Now to our large scale testing. I conducted the test 2 different times with two different data sets. The data set consisted of 100 numbers ranging from 0-99. All the numbers were randomly generated using a function built into my program that creates a file of 100 numbers. As the test shows, the LRU and Clock algorithm was very similar to the small scale testing. With the lower amount of frames the program simulate, the similar in performance the LRU and Clock algorithm performs. But as we get higher in frames, the LRU performs much better with less faults than the Clock algorithm. With higher simulated frames, the LRU throws less faults then the clock after the 20 frame test. This would suggest that while both algorithm LRU and Clock are very similar in performance in low frame memory systems, the LRU out performs the Clock algorithm in higher frame memory system. LRU throws less faults than the Clock algorithm does in my higher frame test.

The anomaly here is the Optimal algorithm. In the small scale test, it shows that the algorithm performs much better linearly as we increase the frame size. But that’s not the case in the large scale test. As we compare both test, it seems that as we increase the frame size the algorithm beings to even out and plateaus. This would suggest that as frame size approaches infinity the performance of the Optimal algorithm hits a lower bound for that data set and stops increasing its performance.

To conclude this analysis, The testing of the algorithm shows that with both the small scale and large scale test, the LRU and Clock algorithm performs very similar to each test (small and large). The LRU page replacement algorithm performs better in terms of throwing less faults in higher frame test, while both algorithm performs similarly with the Clock algorithm outperforming the LRU in the lower frame test (throwing equal or lower faults). Now the Optimal algorithm out performs both the LRU and Clock algorithm, but in the large scale testing it seems to level out as we approach higher frame size. It can be suggested that in higher frame, anything above what I have tested ( greater than 50 frames ), the LRU and the Clock Algorithm my approach the performance of the optimal algorithm or even outperforming it (Throwing less faults, and swapping pages).