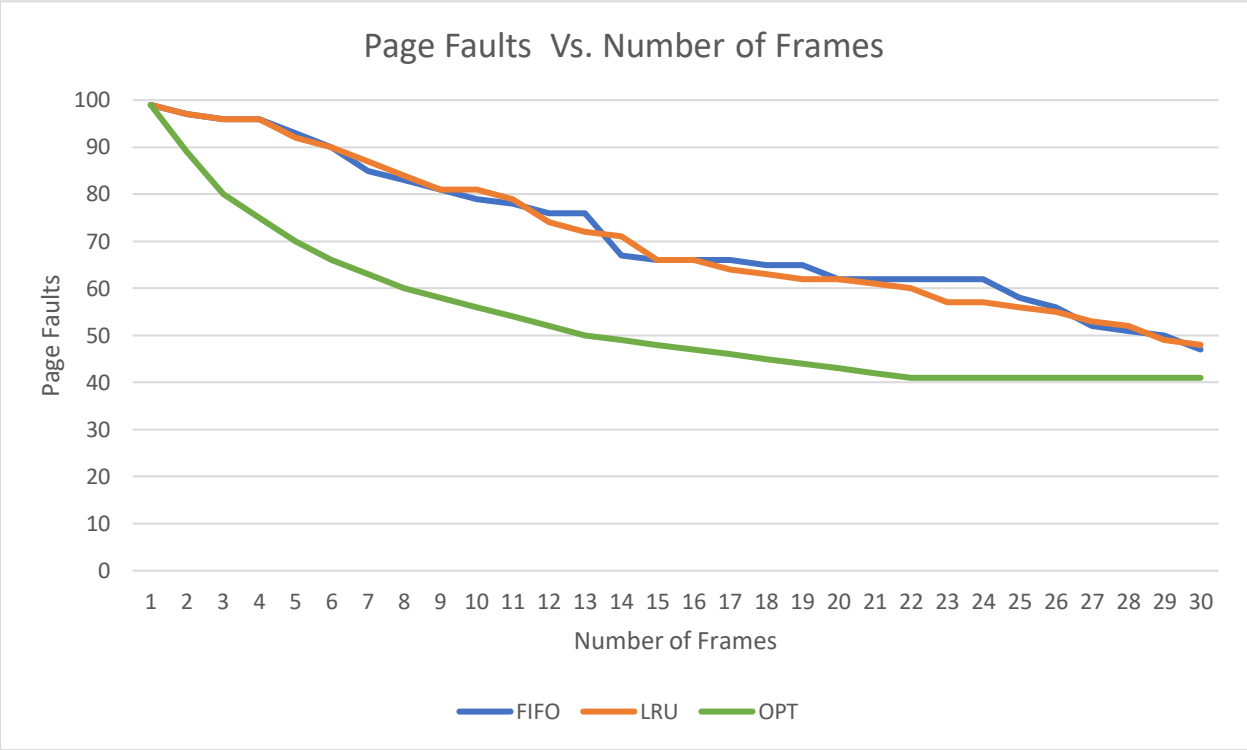
Dan Lepe
11/26/20
Program 3

Overview

This program was simple. The Program is designed to simulate three separate Page Replacement algorithms. This program would have been possible with standard C++ dequeues. However, there were several actions that needed to be done several times. Furthermore, limits needed to be placed on the size of the frame. Because of these limitations a new data structure was created using dequeues. Several of the standard dequeue functions were duplicated in the **pFrame Class**. Other functions were added to facilitate the implementation of each of the algorithms. Once the data structure was in place it was a simple matter to create the algorithms. Each algorithm was created as a separate function within the main.cpp file for easy access to the main function.
FIFO algorithm simply replaces each page fault as it comes along by pushing it from the back and popping the front entry.

LRU algorithm was almost as simple. LRU pushes a page fault entry and pops the front entry. However, when a 'Hit' is registered the stack is re-ordered so that the entry which was a hit is popped from the stack and pushed back to the back of the stack. This keeps the stack with the oldest entries in the front.
OPT algorithm required a dedicated function, furthestIndex() which returns the index of the page frame which contains the page which is either furthest down the list of upcoming pages, or not present; with priority given to pages which are not on the list. This index is then used by page faults to replace the page at that index.

The results



The results were in line with the expected results. Both FIFO and LRU performed similarly with LRU having more controlled movement. I did run FIFO over 100 times with different random seeds for the Page Numbers trying to see if I could force Belady's anomaly to show up with the FIFO algorithm. However, in all 100 cases not one exhibited the anomaly. I was able to find a string online which guaranteed the anomaly would show up with page frames sizes of 3 and 4 in FIFO. I tested that string and was able to get 9 and 10 page faults respectively. This means the algorithm worked perfectly but the sample strings just did lend themselves to show off the anomaly. The example string and the results were left within the main function commented out as a reference to what was tested to see the anomaly.

I did perform several other tests with different random strings to see other results. In a couple, LRU was found to perform worse than FIFO. In this test LRU actually performs worse than FIFO at the 30 page frame size. With FIFO showing 47 page faults to LRU's 48.

Optimal performed just as expected. A steady and sharp decline in page faults as the number of frames increases followed by a flattening once the optimal amount of page frames is reached.

Compile and Run

The bash file provided should be capable of compiling the program and running over every simulation of algorithm and page frame count. No executable is included in this folder. To compile you may try “bash run.sh” if there is any issues with this you can also try “./run.sh” or “sh run.sh”

The bash file will automatically create a data folder for you to dump the run data into(note the run data is in .csv format for easy transfer to Excel for the charts). However you may need to create a directory where the data is moved to. Simply create a new folder named ‘data’

you might need to chmod the bash file. On the terminal do ‘chmod u+x run.sh’

Finally. If you just want to compile and run the program once the Makefile included should take care of the compilation. Simply type “make” in the terminal to compile. To run the file you need to add parameters here’s an example of how to run the file ‘ ./pageSim 1 30 1 ‘