Page Replacement Algorithm Implementation

By: Luis Castro
University of South Florida
College of Engineering
U19068559

I. Abstract

As the title states, the focus of this project is the implementation of some of the page replacement algorithms that exist in an operating system. Though actual memory and address spaces are not being used here, it is a close simulation that is made to get the basic understanding of how these algorithms work. In this report, you will find more information on what the goal of the project is, the procedures and methods used, and some of the results and conclusions that were achieved throughout the project. Included with this project will also be the source code along with instructions on how to compile, run, and execute said program.

II. Introduction

Let us start with the basics of page replacement algorithms. These functions are under the memory management section of operating systems topics. The “pages” refer to blocks of virtual memory that is allocated within an operating system. Within the pages, we have “page frames” which coincide with physical addresses. We use these in order to transfer pages between main memory and storage (for example, hard disk drives, or solid state drives).

Here we will describe four of these algorithms. There will be a small explanation (nothing to abstract or in-depth) for the sake of simplicity, in order to grasp the concept. First In, First Out (FIFO) keeps track of which pages have been in memory the longest, and those that are oldest will be the ones that will be replaced. In short, look back in the table to see what will be removed.

Secondly, we have Optimal page replacement where the OS looks ahead in the page reference string to see which (if any) page will not be used for the longest amount of time. The last page that would be used - or if that page does not appear again in the string - will be the page that is inserted into the frames.

Next, there is Least Recently Used (LRU). Here, the OS looks at which of the pages in frames has been inserted least recently. One thing to look out for here are page hits, because having a hit could reset a page that has been used.

Lastly, there is Least Frequently Used (LFU). This is a little more complicated because we have to keep track of how many times a page is referred to in the reference string. In addition, any time a page is added into frames, we increment the frequency. On the other hand, when a page is replaced, we have to decrement its frequency.

III. Methodology

First of all, this project was built in a Linux environment using Ubuntu's most recent stable release (version 18.04). It was written in C++ for a bit more versatility since there wasn't actual memory being tossed around. Had that been the cased, C would have been used. Anyways, this was a developed just using a text editor and the command line and the g++ compiler for UNIX.

Though these algorithms can be easy to understand visually, translating this into text and program language, I started by just writing pseudocode on paper in order to write out the steps needed to make these algorithm simulations work. After that, it was just a matter of knowing how to use the tools from C++ to aid in creating the algorithm.

Another reason C++ was used was for the libraries included. Note: these libraries in no way solve the problem or create the algorithms, they just help with some of the tedious details. Moreover, the only libraries implemented were <vector> which is for arrays, <deque> which helped with shifting the arrays (since I treated the reference string as a queue from which I would push/pop elements), and <algorithm> which sorted arrays.

IV. Results

As the program was being built, I made the console output each line of the table, just like the examples we had done in class and on exams. This made it easier to see the entire process just as if were being done on paper. Assuming we are using a size of three frames, the user will input ten different page references as integers.

After the string is entered, the program will automatically take the input, read it back to the user, and then format it into a table (similar to examples done in class) so that you can see the algorithm working as each page is replaced.

V. Conclusions

Though this project was a lot of tedious coding, such as iterating through loops, sorting arrays, and doing comparisons, I did enjoy learning how to use some of the classes in C++. It was nice to be able to build the project from scratch because it made me feel that I had a much better understanding and greater control over the project.

Even though I was using programming skills learned prior, I noticed that was I completed one algorithm and moved on to the next, I was able to make the code more readable, and efficient. As with any project, you always gain more experience because with any new task, you're sure to find new bugs, which of course need to be solved or your program will not compile.