Samuel Scalf
CMSC 412 6380 Operating Systems (2212)
University of Maryland Global Campus
Dr. Suzanna Schmeelk

Final Project Report

Table of Contents

Introduction

The program is a **Demand Paging** virtual memory simulator!

The program is able to simulate the FIFO, OPT, LRU, and LFU algorithms for a notional system with only N physical frames (numbered from 0 to N-1, N < 8), determined by the user. The notional process running has a virtual memory of ten frames (numbered from 0 to 9).

The user will be presented with a menu of options at the start of the program. Please see Appendix A for a description of these options.

Design and Implementation

I started out by creating the menu and empty methods for each of the menu options. In order to verify that methods were being called, I added print statements stating the method was called. Once all of the menu options were verified, I built-in input checking using regular expressions.

While I may normally have built this with a more object-oriented approach by using multiple classes, I tried to fall back on the more traditional approach I learned with PASCAL. As such, the primary variables and constants used within the program are all declared as static before any of the methods. This ensures the space requirements are kept at a minimum, since none of the arrays need to be re-allocated.

When starting to build out the meat of the algorithm methods, I created a separate source file for testing in isolation. This allowed me to play around with the algorithm without unintentionally damaging the primary code. While I could have left them as separate files, I opted to include them in the primary file to share variables without “#including” a separate “variable” file.

When reading over the algorithms, a lack of queues is easily noticed. This was intentional. Since the FIFO algorithm always follows the same order – starting at 0 and going to N-1 physical frame, where N is the number of physical frames – I simply used an integer, initialized at 0, and increasing it by one and getting modulo N. This effectively created the FIFO queue for physical frames and only moved to the next frame if a page fault occured.

With the other algorithms, I created an array of size N to keep track of priority. At the start of each loop, I check the priorities, starting at 0 and moving to N to determine the highest priority, depending on the algorithm. This kept the choice in line with the lesson modules, which checked for highest priority “top down,” in case of equal priorities. The only time I needed to adjust this was for the OPT algorithm.

In the OPT algorith, I needed an additional check for the initial N frames. Since I decreased all priorities by 1 each loop and calculated the priority for the newly inserted frame to 99 if not present again, I found that sometimes an earlier frame might get replaced despite not all frames being used. I overcame this by setting an initial variable to N and decreasing it each loop until it was -1, after which no further decreases were done (to prevent underflow) and page faults could proceed normally. Additionally, in order to keep the “top down” approach, only priorities less than 99 were reduced. This ensured that if a virtual frame was inserted but never again used, even in physical frame 0, it would get selected “top down”.

With both the LRU and LFU algorithms, the priorities started at 0. The LRU algorithm would increase each loop, including the empty frames. This worked out well, because the priority would be set to 0 when the frame was filled. This means that for the third virtual frame, physical frame (PF) 0 had a priority of 2, PF 1 had a priority of 1, and PF 3 on had priorities of 3. With the fourth virtual frame all of these would be increased by 1 (assuming a different virtual frame is needed) and PF 3 would now have a priority of 1, with later physical frames having priorities of 4.

The LFU algorithm was unique in that in did not “increase” priority with each loop. Instead, the priority was a count of how many times a certain virtual frame in physical memory was used. In other words, it counted the frequency. The first “top down” physical frame with the lowest count, the “least frequently used” frame was the target frame.

I tried my best to replicate the table shown in the lesson modules. Since the application is purely command line driven without non-standard libraries, the output table was not updated directly. Rather, the output table is displayed repeatedly with the next virtual frame “column” added, pausing for the user to press Enter.

My original plan was to use the ncurses library to create an updating table, but porting it to Windows was not as straightforward as it seemed at first. Also, I lacked confidence in learning the ncurses library by the due date. I still plan to rebuild the application using ncurses, because I feel it would be a good exercise to learn about the library.

Appendix A: Menu Options

The program should be menu-based and the menu will keep the user in a loop containing

the following options:

0 – Exit

Will exit the program

1 – Read reference string

A reference string will be read from the keyboard and stored in a buffer. Each value of the reference string will be verified and validated (or rejected).

Using option 1 again will result in overwriting the old reference string.

2 – Generate reference string

A reference string will be randomly generated; the length of the reference string will be given by the user interactively. The string will be stored in a buffer.

Using option 2 more than once will result in overwriting the old reference string.

3 – Display current reference string

Will display the stored reference string; if there is no reference string stored yet, an error message will be displayed.

4 – Simulate FIFO

Will simulate the step by step execution of the FIFO algorithm using the stored reference string; if there is no reference string stored yet, an error message must be displayed.

The user will press a key after each step of the simulation to continue the simulation.

The total number of faults will be displayed at the end of the simulation.

5 – Simulate OPT

Will simulate the step by step execution of the OPT algorithm using the stored reference string; if there is no reference string stored yet, an error message must be displayed.

The user will press a key after each step of the simulation to continue the simulation.

The total number of faults will be displayed at the end of the simulation.

6 – Simulate LRU

Will simulate the step by step execution of the LRU algorithm using the stored reference string; if there is no reference string stored yet, an error message must be displayed.

The user will press a key after each step of the simulation to continue the simulation.

The total number of faults will be displayed at the end of the simulation.

7 – Simulate LFU

Will simulate the step by step execution of the LFU algorithm using the stored reference string; if there is no reference string stored yet, an error message must be displayed.

The user will press a key after each step of the simulation to continue the simulation.

The total number of faults will be displayed at the end of the simulation.