**CS 4310 Operating Systems**

**Project #2 Simulating Page Replacement Manager and Performance Analysis**

**Due: 11/21**

(Total: 100 points)

**Student Name: James Lansdown**

**Date: 11/19/19**

***Important:   
-*** *Please read this document completely before you start coding.   
- Also, please read the submission instructions (provided at the end of this document) carefully before submitting the project.*

***Project #2 Description:***

Simulating Page Replacement Manager of the Operating Systems by programming the following three page replacement algorithms that we covered in the class:

1. First In First Out (FIFO)
2. Least Recently Used (LRU)
3. Optimal Algorithm

You can use either Java, or C++ for the implementation. The objective of this project is to help student understand how above three page replacement algorithms operates by implementing the algorithms, and conducting a performance analysis of them based on the performance measure of ***page faults*** for each page replacement algorithm using multiple inputs. Output the details of each algorithm’s execution. You need to show what pages are inside the page frames along with the reference string and mark it when a page fault occurred. You can choose your display format, for examples, you can display the results for each reference string in a table format as shown in the class notes. The project will be divided into three phases to help you to accomplish above tasks in in a systematic and scientific fashion: Design and Testing, Implementation, and Performance Analysis.

The program will read in a reference string a file (ReferenceString.txt) – this file will be generated by you. In this project, assume that

1. the length of the reference string is always 30, e.g. 361724720354720146353214567012
2. there are 8 pages, from 0 to 7.

Note that you need to generate 50 testing cases, or 50 reference strings of length 30. You are required to run each reference string using three algorithms with *NumberOfPageFrame* (3, 4, 5 and 6) page frames.

A sample input file of having 5 page frames and a reference string 361724720354720146353214567012 (always with length 30) is given as follows:

[Begin of ReferenceString.txt]

*NumberOfPageFrame* value:   
5

Reference String:

361724720354720146353214567012

[End of ReferenceString.txt]

You can implement the algorithms in your choice of data structures based on the program language of your choice. Note that you always try your best to give the most efficient program for each problem.

***Submission Instructions:***

* ***turn in the following @blackboard.cpp.edu after the completion of all three parts, part 1, part 2 and part 3***
  + 1. ***(1) three program files (your choice of programming language with proper   
        documentation)***

1. ***this document (complete all the answers)***

**Part 1**

**Design & Testing (30 points)**

* 1. Design the program by providing pseudocode or flowchart for each page replacement algorithm.

First in First Out

Loop through reference pages

If a set of frames is not full

Add a reference page to a set of frames until full

Add the page added to the frame to a queue called referenceQueue

Increment pageFaults

Else

If the current page is not in the set of frames

Remove the first page in the queue

Replace the page in the set with the first page in the queue

Add the current page to the queue

Incement pageFaults

Return pageFaults

Least Recently Used

Loop through reference pages

If a set of frames is not full

Add a reference page to a set of frames until full

Add the page added to the frame to a map called referenceMap to keep track of recently used

Increment pageFaults

Else

If the current page is not in the set of frames

Find the least recently used page in the frame from referenceMap

Replace the page in the frame set with the found least recently used page

Update the current page to the referenceMap

Incement pageFaults

Return pageFaults

Optimal Page Replacement

Loop through reference pages

If a set of frames is not full

Add a reference page to a set of frames until full

Increment pageFaults

Else

If the current page is not in the set of frames

Predict which page of the current frame set will be used the least often of the reference pages

Remove predicted page from frames set

Add the current page to the frames set

Incement pageFaults

Return pageFaults

* 1. Design the program correctness testing cases. Give at least 4 testing cases (with 3, 4, 5, or 6, page frames) to test your program, and give the expected correct output (# of page faults) of the program for each case in order to test the correctness of each algorithm.

<complete the following table>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testing case # | Input  (Input file with a number of page frames value and a reference string) | Expected # of page faults for FIFO  (√ if Correct after testing in Part ) | Expected # of page faults for LRU  (√ if Correct after testing in Part 3) | Expected # of page faults for Optimal Algorithm (√ if Correct after testing in Part 3) |
| 1  (3 page frames) | *NumberOfPageFrame* value: 3  Reference String:  361724720354201463532145  67012 | <27>  √ | <27>  √ | <20>  √ |
| 2 (4 page frames) | *NumberOfPageFrame* value: 4  Reference String:  361724720354201463532145  67012 | <25>  √ | <27>  √ | <16>  √ |
| 3  (5 page frames) | *NumberOfPageFrame* value: 5  Reference String:  361724720354201463532145  67012 | <23>  √ | <25>  √ | <14>  √ |
| 4  (6 page frames) | *NumberOfPageFrame* value: 6  Reference String:  361724720354201463532145  67012 | <15>  √ | <17>  √ | <12>  √ |

* 1. Design testing strategy for the programs. Discuss about how to generate and structure the randomly generated inputs for experimental study later in Part 3.

*Hint 1: To study the performance evaluation of the three page replacement algorithms, it is the easiest to use a random number generator for generating the inputs. However, student should store each data set of 50 trails (reference strings) for each of the four frame sizes (3, 4, 5, 6) and use the same data set for running each of the three page replacement algorithms.*

*Hint 2: The average performance result (number of page faults) of each input data size can be calculated after an experiment is conducted in 50 trails (reference strings) of each page frame size (3, 4, 5, 6). We can denote the run time results (number of page faults) as the set X = {x1, x2, x3 … x50} which contains 50 results (number of page faults) for each page frame size, and each xi is the number of page faults of each trail (reference string number i).  
  
The Average time =*

For 3, 4, 5 and 6 frames

For an amount of trials (50 trials)

For an amount of reference pages (30) Generate random reference and add them to a string or array

Sum the page faults for each replacement algorithm

FIFOsum += findPageFault(reference pages)

LRUsum += findPageFault(reference pages)

OPTsum += findPageFault(reference pages)

Average times = FIFOsum / 50 trials

LRUsum / 50 trials

OPTsum / 50 trials

**Part 2**

**Implementation (30 points)**

1. Code each program based on the design (pseudocode or flow chart) in Part 1(a).

<generate three programs and stored them in three files, needed to be submitted>

1. Document the program appropriately.

<generate documentation inside the three program files>

1. Test you program using the designed testing input data given in the table in Part 1(b), Make sure each program generates the correct answer by marking a “√” if it is correct for each testing case for each program column in the table. Repeat the process of debugging if necessary.

<Complete the three columns of the three algorithms in the table @Part 1(b)>

1. For each program, capture a screen shot of the execution (Compile&Run) using one testing case to show how this program works properly

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

Example of 50 trial test:

A close up of text on a black background

Description automatically generated

By now, three working programs are created and ready for experimental study in the next part, Part 3.

**Part 3   
Performance Analysis (100 points)**

1. Run each program with the designed randomly generated input data given in Part 1(c). Generate a table for all the experimental results for performance analysis as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| number of page frames | Average of the total completion time  (FIFO Program) | Average of the total completion time  (LRU Program) | Average of the total completion time  (Optimal Algorithm) |
| 3 page frames | *19.56* | *19.56* | *14.14* |
| 4 page frames | *16.04* | *15.96* | *11.38* |
| 5 page frames | *13.82* | *13.96* | *9.86* |
| 6 page frames | *11.62* | *11.1* | *8.74* |

1. Plot a graph of each algorithm, average page fault vs. page frame size (3, 4, 5, 6) and summarize the performance of each algorithm based on its own graph.

This graph shows a negative slope which indicates that the FIFO algorithm’s average page faults decrease with an increase in frame size. The slope of this graph is -2.6, which I will be comparing with the other algorithms.

Similarly, to the FIFO algorithm this graph also has a negative slope indicating that the number of page faults decreases with an increase to the frame size. The slope of this graph is -2.7.

The slope of this graph is -1.8 which indicates that the page faults decrease with an increase to the frame size, which behaves similarly to the other two algorithms. I will compare the slopes of all three algorithms in the next section.

Plot all three graphs on the same graph and compare the performance (page faults) of all three algorithms. Rank three page replacement algorithms. Try giving the reasons for the findings.  

Here you can see that the optimal algorithm performs much better than the other two in general. However, the slope of the optimal at -1.8 is smaller than the other two, the FIFO being -2.6 and the LRU being -2.7. This shows that the LRU has the best performance increase with more frames, which is almost the same as the FIFO and the OPT showing the lease performance increase with more frames. Given these results, I can rank the optimal algorithm as the best option because of its overall lower fault rate, then the least recently used algorithm just because of its slight increase in performance with an increase in page frames and the first in first out at last.

1. Conclude your report with the strength and constraints of your work. At least 100 words.

(Note: It is reflection of this project. If you have a change to re-do this project again, what you like to keep and what you like to do differently in order get a better quality of results.)

The strengths of my project are that I used classes for each algorithm an to process the files for the reference strings and to create the random reference string file. This made making changes that would affect each algorithm used in the project a lot easier and helped to organize and focus on specific sections of my code. Some changes that I would make would be to further compartmentalize longer runs of code into smaller methods, which would allow me and a reader to more easily track variables and to understand what each method is doing. I also think that I can further reduce the time complexities of the algorithms, especially the prediction method for the optimal algorithm and would like to focus more time into producing better pseudo code before starting the implementation of the code.