بسم الله الرحمن الرحيم



**Operating Systems Project 2 Report**

**Instructor:** Dr. Adnan Yahya

**Name:** Mohammed Sehwail, **ID**: 1131253

**Name:** Mahmoud Abdelkareem, **ID**: 1130786

**Name:** Hussein Dahir, **ID**: 1131138

**Abstract:**

In this project, we implemented a simulator for experimenting with page replacement algorithms, which were, Second Chance and Clock, the even ones in the algorithms list in project description (2,4) based on the max value of the least significant digit of the team ID numbers. As a bonus, we designed a very simple and friendly use interface.

**Theory\*:**

The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page has been selected, however, we inspect its reference bit. If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances). In addition, if a page is used often enough to keep its reference bit set, it will never be replaced.

One way to implement the second-chance algorithm (sometimes referred to as the clock algorithm) is as a circular queue. A pointer (that is, a hand on the clock) indicates which page is to be replaced next. When a frame is needed, the pointer advances until it finds a page with a 0 reference bit. As it advances, it clears the reference bits. Once a victim page is found, the page is replaced, and the new page is inserted in the circular queue in that position. Notice that, in the worst case, when all bits are set, the pointer cycles through the whole queue, giving each page a second chance. It clears all the reference bits before selecting the next page for replacement. Second-chance replacement degenerates to FIFO replacement if all bits are set.

**Program Implementation:**

In our implementation, we used java language. We made four java classes only, Page Class, Frame Class, Process Class and PageReplacement Class. Page Class which simulates a memory page was consisting of the page number and the pid of the process that own this page, Frame Class was just like a single memory cell, which consists of a page and its bit reference, and so the memory was an array of Frames, Process class which simulates a process was consisting of the process attributes (burst time, arrival time, start time, finish time, etc…), and so, processes were java objects from this class, finally, PageReplacement Class was for the GUI and for our main program which consists of all algorithms codes and its functions.

**How to Run Our Program?**

We attached an executable jar file for our program, so that you can run it with no need for an IDE, and we also attached the code and a soft copy of this report.

Our interface is very easy to use, as shown in the following screenshot, you just need to input memory size and time quantum, select an algorithm and click “Browse and Run” to choose and input file and run the program.

Notes:

1) The main input file for our program is the scheduling data file, memory trace files must be in the same folder that contains the scheduling data file so that our program can recognize them and it will load them automatically.

2) If the user didn’t enter any value for memory size or time quantum, then memory size will be 10 and time quantum will be 2, by default.

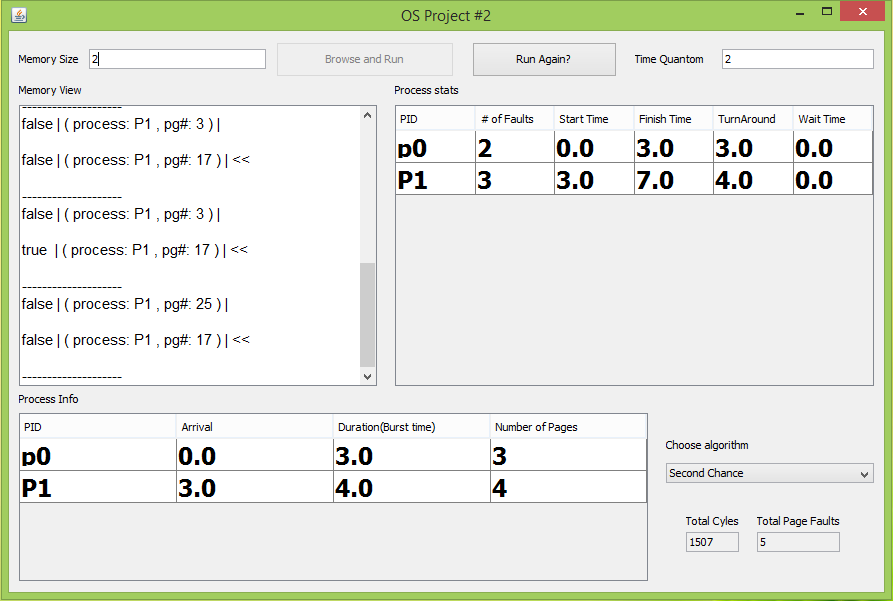


Figure 1

**Input Files Structure:**

The structure of the input file (scheduling data file) for our program should be like the screenshot in figure 2:

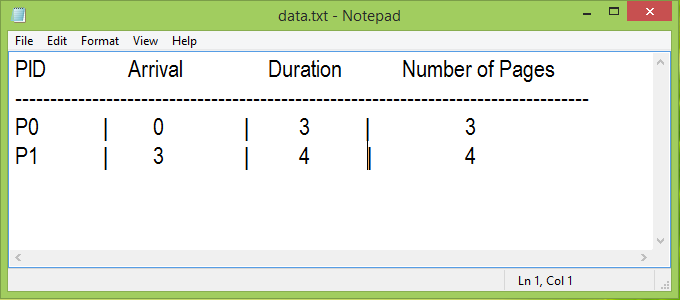


Figure 2

First two lines will be ignored (but they are necessary for our code, since we always skip the first two lines in any input file), they are just to make it easier to enter and read data in the file. The “|” separator is necessary between the numbers in each line (but the spaces are not, you can enter data without spaces).

For memory trace files, they should be like the screenshot in figure 3:

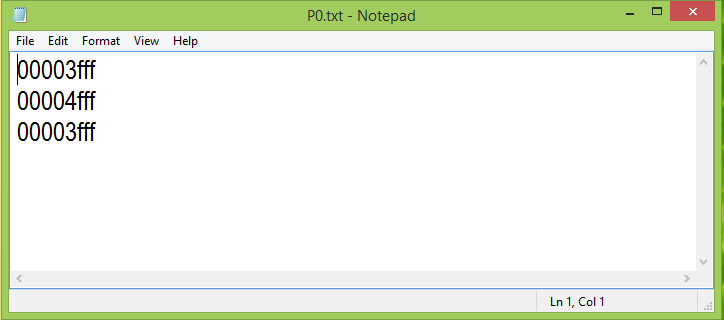


Figure 3

Memory trace files contains the memory addresses in hexadecimal. Note that each memory trace file should be named with the same name of the process that owns this memory trace file, i.e. if we have two processes, p0 and p1, then their memory trace files should be named “p0.txt” and “p1.txt”, respectively (keeping in mind that file name isn’t case sensitive). And it is very important to make sure that the memory trace file for each process contains the same number of pages that has mentioned in the scheduling data file earlier.

**Conclusion:**

From this project, one can see and test different page replacement algorithms, compare between them, and so know what is better for a specific purpose, since some algorithms may give less page faults than some other algorithms and other algorithms may be better for other cases. Simulating Algorithms is good, since we can see their efficiency before applying them in real, and it was very nice to have a project to train us to do something like this.

**References:**

\*: Operating System Concepts, 9th Edition.