Operating Systems (CSE223)

Project 2 Report

Submitted By:

Ahmed Alaa Ahmed	**16P6071**
Robear Wagih Nabih Selwans	**16P8207**

Project Description

This is a simple C++ program that compares the efficiency of widely known page replacement algorithms. It takes the length of the page reference string and the number of page frames in the system as arguments. It then randomly generates the reference string with page numbers (0-99) and applies the algorithms to that string. The algorithms included are: First-In-First-Out(FIFO), Least Recently Used(LRU), Least Frequently Used(LFU), Second Chance Algorithm, Enhanced Second Chance Algorithm, and the Optimal Algorithm.

The algorithm is implemented in C++(11) and is set to be running in terminal as a console application.

However, the project is implemented as a web service where the user interface is a website powered by python on the backend using flask framework, the python server invokes the C++ program with the appropriate arguments provided by the user through the website and then returns back the output in a user friendly form.

This allows for fast computation using C++ as a driver for simulating the algorithms and suffer no losses in the computing speed while still having the program work with a nice user friendly interface.

The project displays and compares the output of the algorithms in a table structure and using a bar-chart to compare visually between their number of page faults (using CanvasJS).

How the algorithms were implemented

FIFO

1. We pass the array to the function.
2. We go through the array while keeping an integer holding the index of the first element
3. If we find a page that is not in the frames, we add it then check the length of the page frames
4. If the frames set is larger than the specified frame count, we start removing the element that the starting_index is pointing towards and iterating it until the frames set size is the same as the frame count.

LRU:

1. We walk through the reference string with an imaginary range
2. We also keep track of how many times every page appears in this range
3. We start iterating and increasing the size of the range until we meet a page that is not in it
4. Once that happens we test if the range is more than the frame_count.
5. If that is the case, we decrement its size and remove the elements until an element gets its count down to 0.
6. When that happens, we stop decreasing the length of the range and start walking through the reference string again

LFU:

1. While walking through the reference string, we keep track of how many times every page has appeared by incrementing its value in a hashmap
2. When adding a new element to the set, it automatically reorders itself in ascending order by how many times an element appeared
3. So when we remove the first element from the set, it is guaranteed that the element with least appearances was the one removed
4. On every iteration, we refresh the set, so that it reorders itself to make sure that the most recent element is in the right place.

Second Chance Algorithm:

1. We iterate through the reference string.
2. When a page is not present, we add it to the frames set with an initialized R bit of 0.
3. If a page is present, we set its R bit to 1.
4. If a page is to be removed, we iterate through the frames set. We remove the first page with an R bit of 0 and set the R bits of all the pages before it to 0.

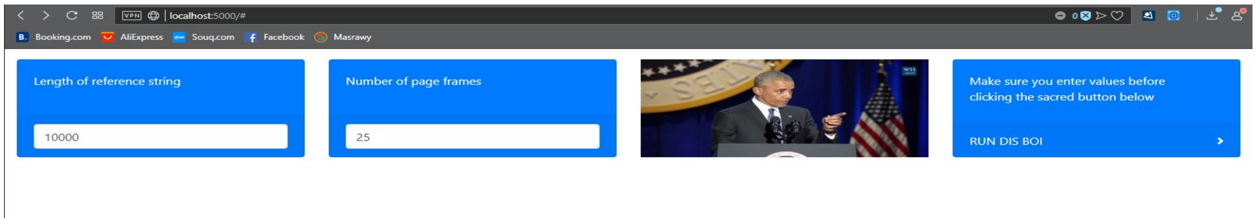
Enhanced Second Chance Algorithm:

1. A randomly-generated bit is added to every page in the reference string (M bit).
2. New pages are added to set 10 or 11 based on their M bit.
3. When a page needs to be removed, set 00 is searched first followed by 01.
4. If no elements are present in those 2 sets, all pages in 10 and 11 get their R bit flipped and are moved to 00 and 01 respectively.

Optimal Algorithm:

1. When an element needs to be removed, we start iterating through the reference string and stop at the end or when pages in the frames are found.
2. If we stopped because of them being found, then we stopped at the page appearing last. So we remove it.
3. If we stopped at the end of the string, we compare the "found" set with the "frames" set to find the missing pages and remove the first one we find.

Screenshots of the project and examples

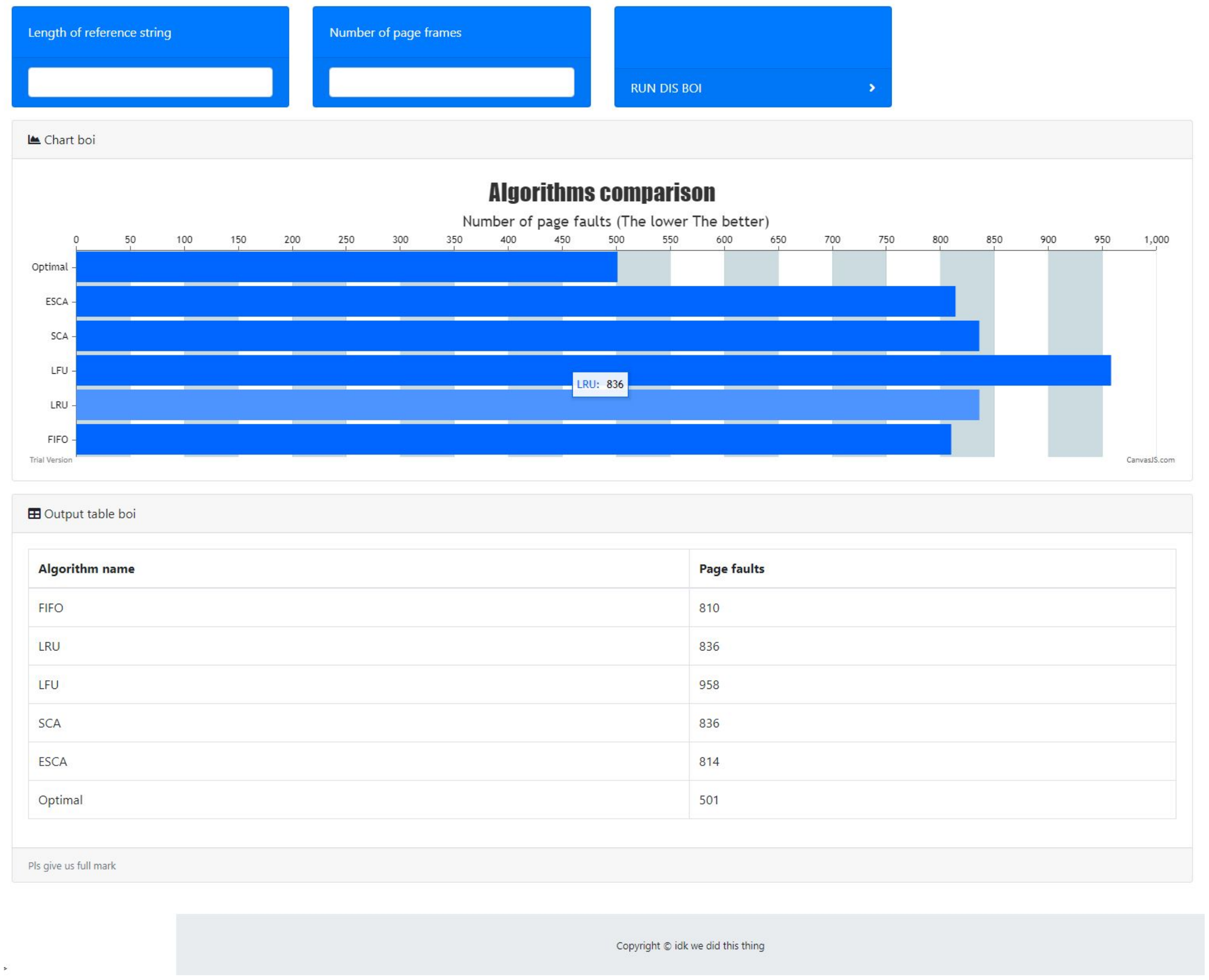


Length of reference string: 10000

Number of page frames: 25

Make sure you enter values before clicking the sacred button below

Figure(1): Input of the program



Algorithm name	Page faults
FIFO	810
LRU	836
LFU	958
SCA	836
ESCA	814
Optimal	501

Pls give us full mark

Copyright © idk we did this thing

Figure(2): The structured output of the program.