Page Replacement

Azaan Nabi Khan 21K-3208

Muhammad Rehan Khan 21K-3172

Syed Muhammad Shayan Hussain 21K-3386

BCS-4A

Instructor: Dr. Gufran Ahmed

Department of Computer Science BS(CS)

FAST-NUCES Karachi

Objective

The objective of testing multiple page replacement algorithms in computer operating systems is to find the most effective algorithm for managing memory pages. These algorithms aim to minimize page faults and maximize page hits. To achieve this, simulations are conducted with varying process loads (workloads) and the number of available memory frames. The process load represents the demand on the system, while the number of frames represents the available memory capacity.

During the simulation, sample workloads consisting of memory references made by processes are used. The algorithms track and determine which pages to keep in memory and which to replace when a page fault occurs. The performance of the algorithms is measured by counting the number of page hits and page faults.

By comparing the results of different algorithms under various conditions, such as different process loads and numbers of frames, the most efficient algorithm can be identified. This information assists in selecting or designing the appropriate page replacement algorithm for a specific system or workload, leading to improved memory management and overall system performance.

Introduction

This project focuses on evaluating and comparing multiple page replacement algorithms used in operating systems to optimize memory management. The objective is to identify the most effective algorithm under varying conditions. Page replacement algorithms are crucial for efficiently allocating and deallocating memory pages, aiming to minimize page faults and maximize page hits.

The project involves simulating different process loads and adjusting the number of available memory frames. Process loads represent the workload placed on the system by running processes, while the number of frames represents the memory capacity. By varying these factors, the performance of the page replacement algorithms can be assessed.

The project measures the number of page hits and page faults for each algorithm to determine their efficiency. The goal is to gain insights into the performance characteristics of the algorithms and identify the best-performing one across different conditions. The findings will contribute to improving memory management techniques in operating systems, leading to enhanced system responsiveness, efficient resource utilization, and an improved user experience.

Background

Efficient memory management is crucial for optimal system performance in computer operating systems. As processes and applications run, they require memory pages for data storage. However, available physical memory is limited, necessitating effective memory allocation and deallocation strategies.

Page replacement algorithms are employed to manage memory pages when the available memory is full. These algorithms determine which pages should be replaced to accommodate new pages, aiming to minimize page faults and the associated performance overhead. Several popular algorithms, such as Optimal, LRU, FIFO, and Second Chance, have been developed with different strategies for page replacement.

The performance of these algorithms depends on factors like memory access patterns, process workloads, and available memory capacity. Evaluating and comparing these algorithms under varying conditions is essential to identify the most suitable algorithm for a given system or workload.

This project builds upon existing knowledge by evaluating and comparing multiple page replacement algorithms. Through simulations with diverse process loads and varying memory frames, the project aims to assess algorithm performance and determine the most efficient one for effective memory management. The project's findings will contribute to advancing memory management techniques, optimizing system performance, and enhancing the user experience.

Methodology

❖ First in First Out

Our implementation of the FIFO algorithm uses arrays and a variable to track the oldest page in the page frame. To insert a new page, we use the modulo function to ensure that the variable stays within the bounds of the array. When a page fault occurs, the new page is inserted at the index of the oldest page, and the variable is incremented. If a page hit occurs, the variable remains unchanged. This approach provides an efficient and straightforward way to manage page frames using FIFO.

❖ Least Recently Used

To implement the LRU algorithm, we use a linked list data structure. The list is in the order of least recently used to most recently used(head to tail). Our approach involves deleting the head of the list and inserting it at the tail whenever a page is accessed. If a page hit occurs, the corresponding node is moved to the tail of the list (i.e., from the oldest to newest order). If a page fault occurs, we delete the head (i.e., the least recently used node) and insert the new data at the tail to maintain the order.

❖ Optimal

Our implementation of the Optimal page replacement algorithm utilizes a linked list data structure. When a page fault occurs, the program searches the inputted data stream for the least used page in the future. The identified least used page is then replaced with the new page.In the event of a page hit, the linked list remains unchanged. This approach optimizes the use of page frames by replacing pages that are unlikely to be used in the future, reducing the number of page faults and improving performance.

❖ Second Chance

Our implementation of the Second Chance page replacement algorithm uses a circular linked list. We insert a new page at the tail of the linked list and use a variable called "chance" to represent a pointer (a Boolean type variable). Additionally, we use another pointer that is incremented each time a new page arrives in the data frame. With the help of this pointer and the chance variable, we determine if we can insert a new page at a given node in the linked list.

After implementing our algorithms, we made functions that would execute all four algorithms for the provided data sets. The data sets(processes and frames) are to be provided by the user. Our function will then print the pages at every iteration of every algorithm; Also mentioning if it was a hit or a fault. Then the number of page hits and faults are displayed for the algorithm that just completed its execution. After all of the algorithms have finished execution, we print a table that shows the names as well as the hits and faults for all of the page replacement algorithms for the user to compare.

Platform

Platforms used are as follows:

- Windows/Linux
- Visual Studio Code

Visual Studio Code

Languages

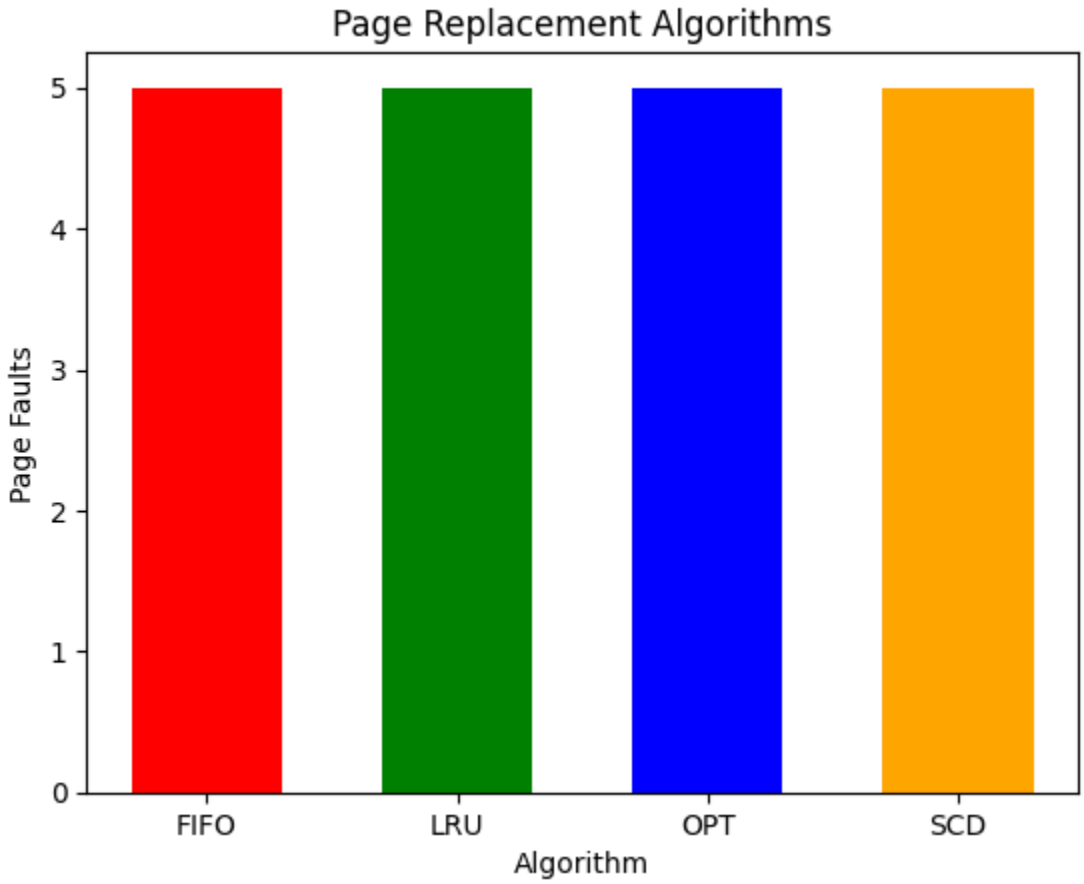
- C++.

Comparison

We tested different data sets on all of the algorithms and then compiled the results into grapes

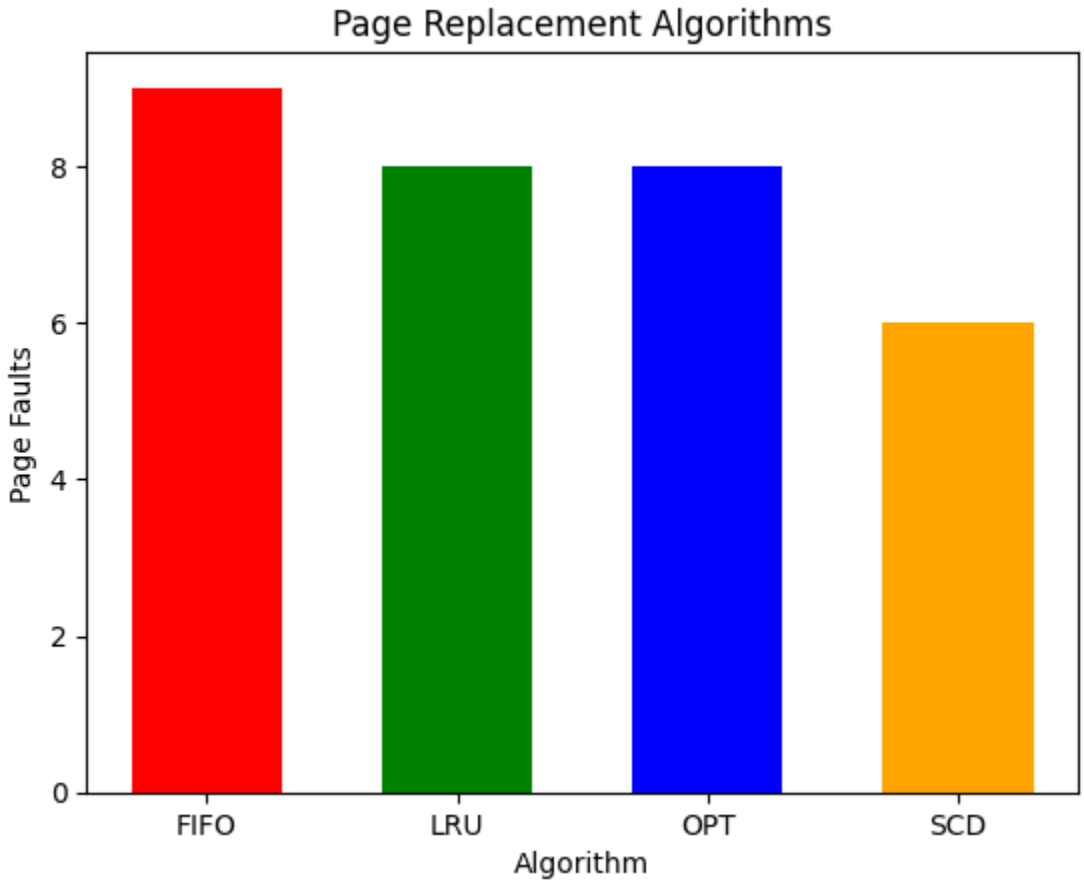
Frames 3

Data stream 1 3 0 3 5 6



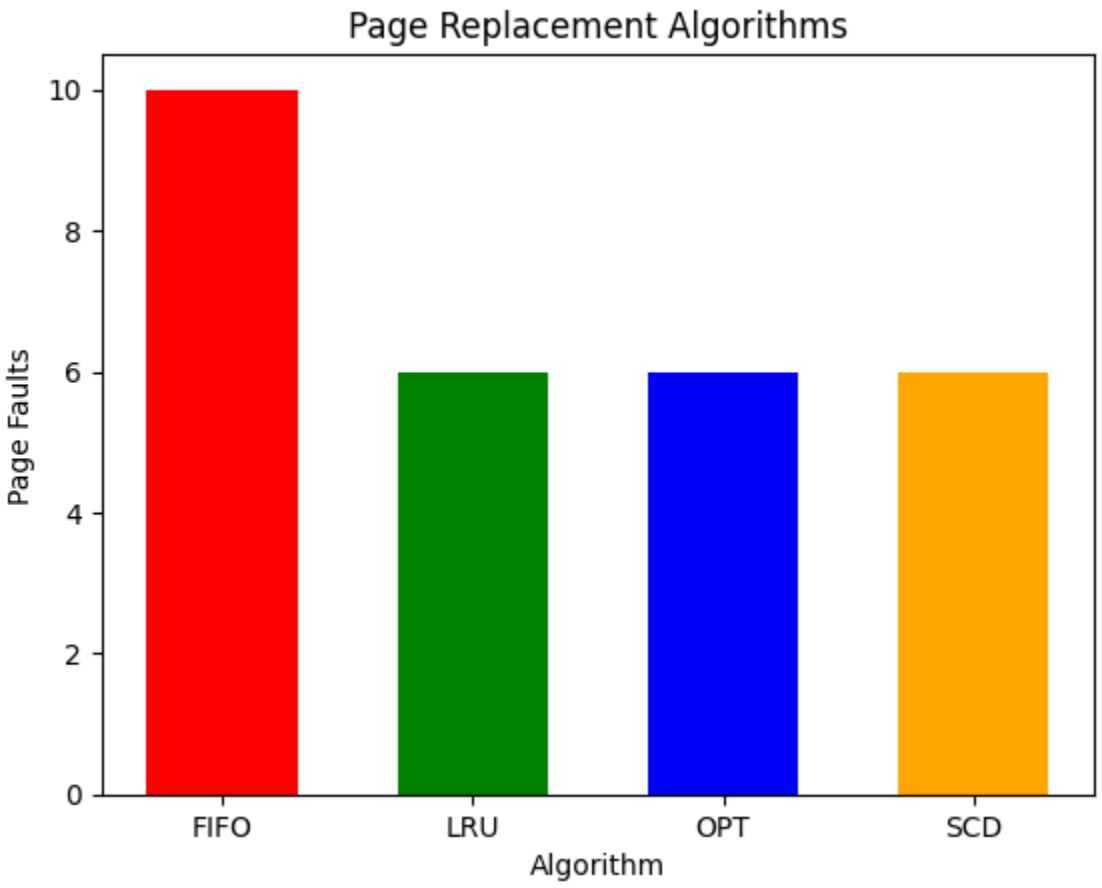
Frames 4

Data stream 0 2 1 6 4 0 1 0 3 1 2 1



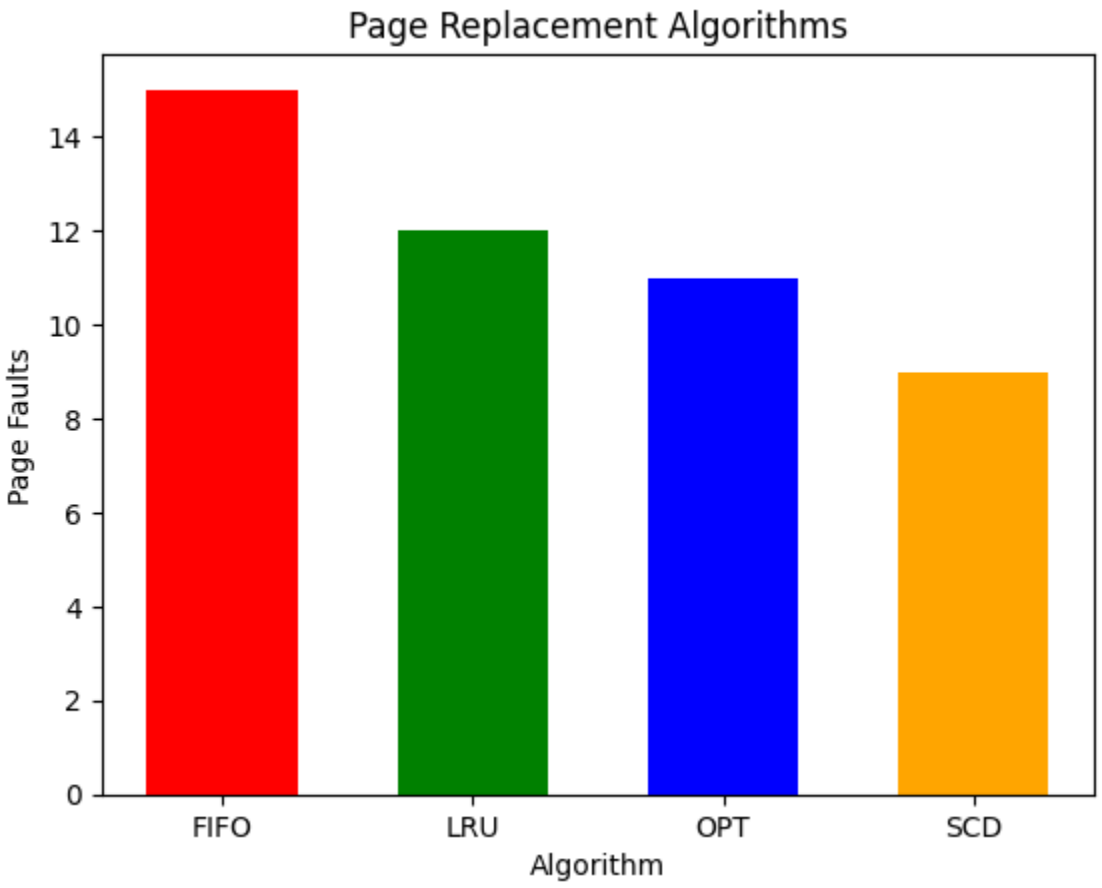
Frames 4

Data stream 7 0 1 2 0 3 0 4 2 3 0 3 2



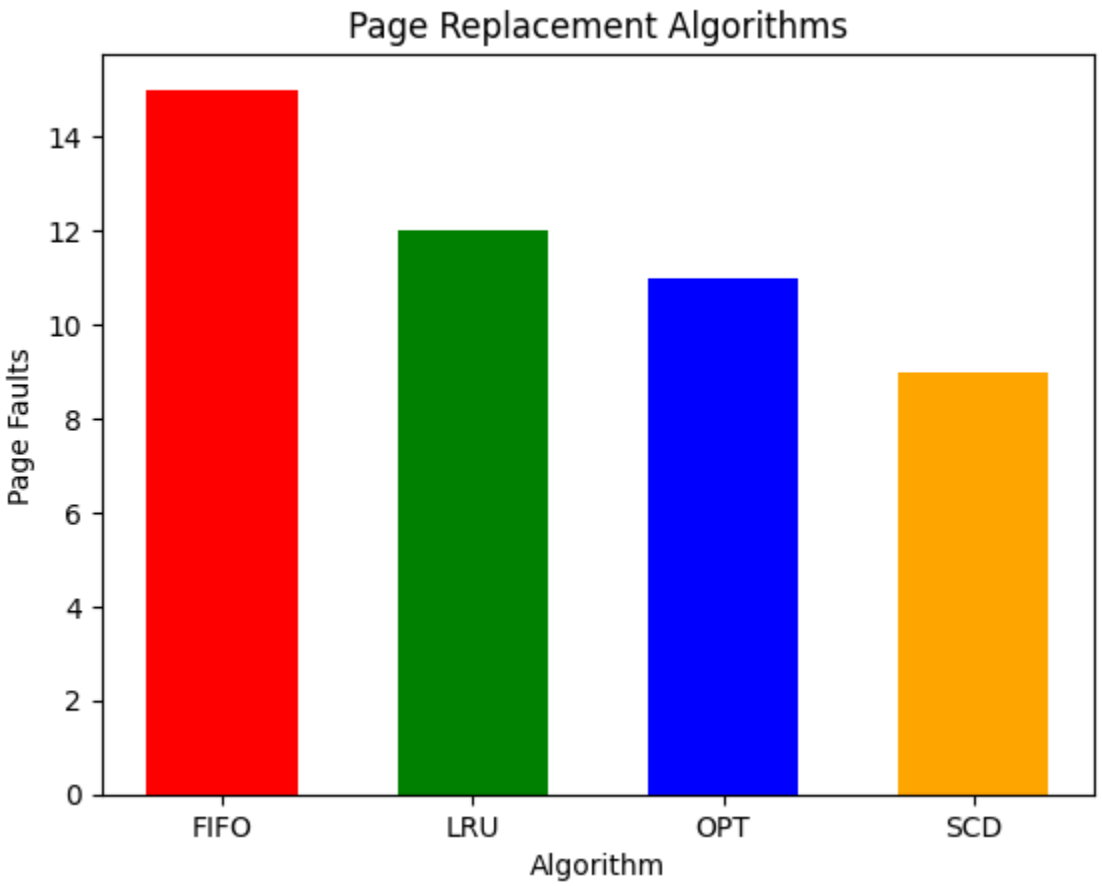
Frames 3

Data stream 7 0 1 2 0 3 0 4 3 0 3 2 1 2 0 1 7 0 1



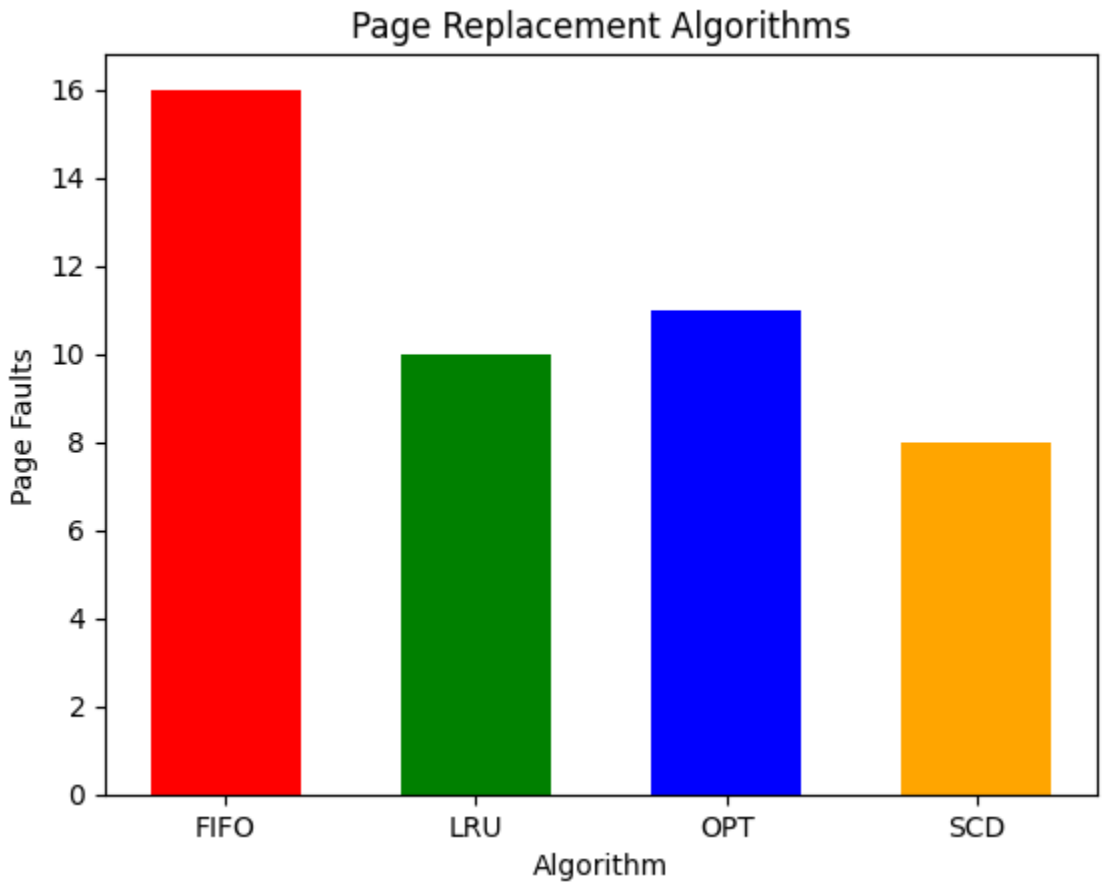
Frames 4

Data stream 7 0 1 2 0 3 0 4 2 3 0 3 2



Frames 4

Data stream 2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1



Analysis

After running various types of data sets, we can conclude that on averages, the worst performing algorithm is FIFO. It has the highest number of page faults compared to other algorithms.

On the other hand, SCD is the best performing algorithm with the minimum number of page faults on average.