ACADEMIC TASK - 2

CSE316

(Operating Systems)

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Name: Ayush Kumar

Registration number: 12322101

Roll No.: 09

Section: K23BM

Submitted to:

Hardeep Kaur Mam

LOVELY PROFESSIONAL UNIVERSITY

Efficient Page Replacement Simulator

1) Project Overview:

Efficient memory management plays a key role in enhancing the performance of operating systems, and page replacement algorithms are at the heart of this process. These algorithms determine how the system manages memory when a page fault occurs, ensuring that RAM is used effectively. The goal of this project is to simulate and compare three popular page replacement strategies:

- **FIFO (First In, First Out)**
- **LRU (Least Recently Used)**
- **Optimal Page Replacement**

Through this simulator, users can input a reference string and define the number of frames to observe how each algorithm handles page faults in real time. The project calculates the number of page faults for each algorithm and displays the results through interactive graphs. This visual representation allows users to easily compare the algorithms and understand their efficiency. Additionally, the project includes step-by-step tracking of memory allocations, providing an engaging and intuitive way for users to learn about the underlying mechanics of page replacement.

2) Module-Wise Breakdown:

Module 1: FIFO Page Replacement Algorithm

- Implements the FIFO page replacement strategy, which replaces the oldest page in memory when a new page needs to be loaded, and there is no available space in memory.

Module 2: LRU Page Replacement Algorithm

- Implements the LRU page replacement strategy, which keeps track of the most recently used pages and replaces the least recently used page when needed.

Module 3: Optimal Page Replacement Algorithm

- Implements the Optimal page replacement strategy, which predicts the future usage of pages and replaces the page that will be needed the farthest in the future.

Module 4: Input Handling

- Handles user input for the reference string (the sequence of page requests) and the number of frames (the capacity of memory).

Module 5: Output and Graphical Representation

- Displays the number of page faults for each algorithm and provides a graphical comparison using bar charts for easier visualization.

3) Functionalities:

 Page **Replacement Algorithms**: Implements FIFO, LRU, and Optimal algorithms.

 Page **Fault Calculation**: Computes and displays the number of page faults for each algorithm.

 User **Input**: Accepts page reference strings and frame count from the user.

 Graphical **Output**: Displays a bar graph comparing the page faults of the three algorithms.

4) Technologies Used:

Programming Languages:

- Python

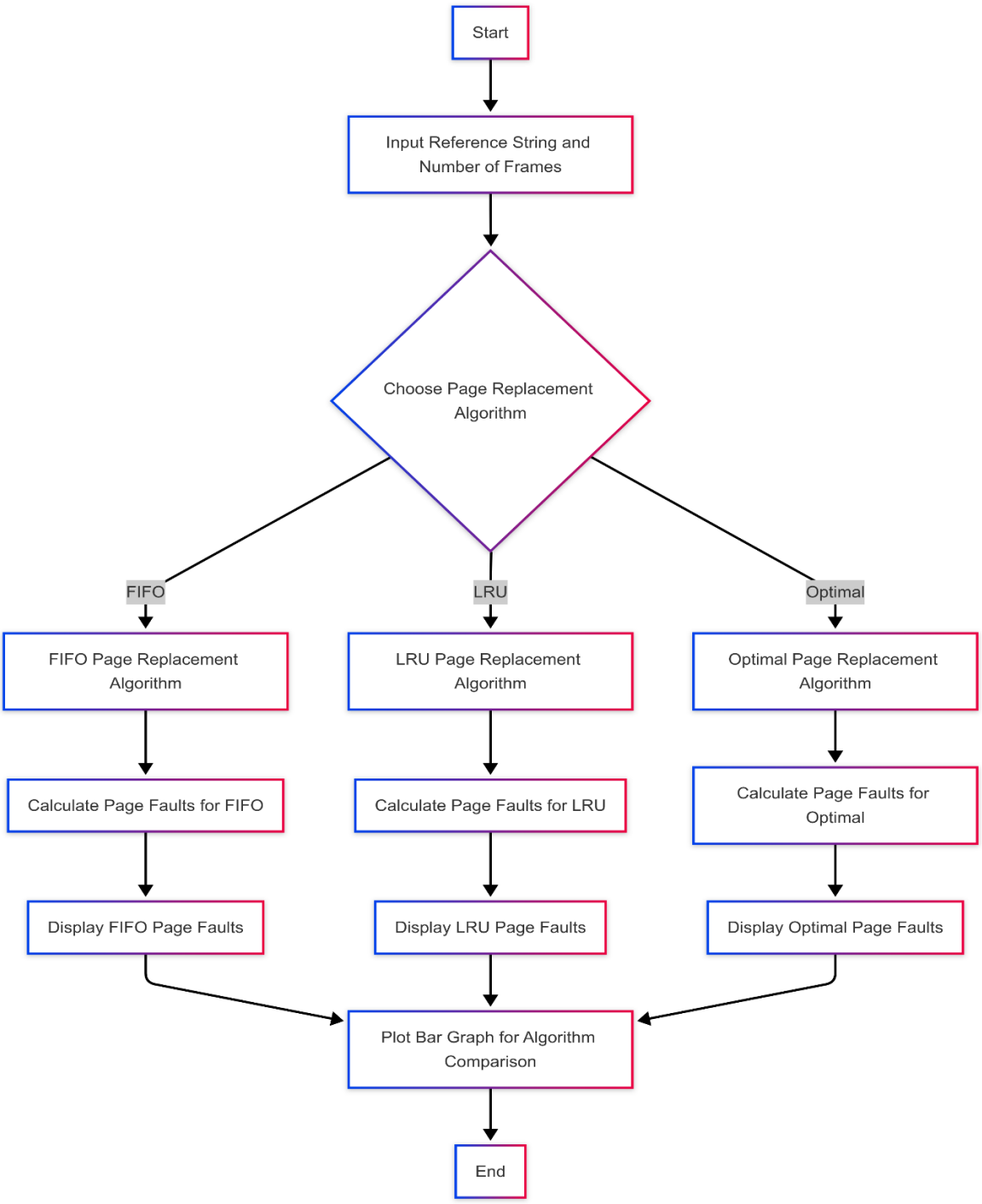
Libraries and Tools:

- **matplotlib**: Used for creating the bar chart to display the comparison of page faults.

Other Tools:

- **GitHub**: Used for version control to track the progress and maintain the project.

5) Flow Diagram:



6) Revision Tracking on GitHub:

Repository Name:

EfficientPageReplacementSimulator

GitHub Link:

https://github.com/CodeBy-Ayush/EfficientPageReplacementSimulator.git

7) Conclusion and Future Scope:

A. Conclusion:

The project successfully implements and compares three major page replacement algorithms. The FIFO algorithm, while simple, tends to produce more page faults. LRU improves performance by keeping track of recent usage but still struggles in some scenarios. The Optimal algorithm provides the best performance but requires knowledge of future page requests, making it impractical in real-world applications.

Future Scope:

- **Enhancing Algorithms**: Future work can include implementing additional algorithms like the Clock algorithm or the Second-Chance algorithm.
- **Optimizing Performance**: Performance improvements for handling larger datasets could be explored.
- **Real-World Application**: The algorithms can be extended to work in real-time operating systems or memory management tools.

8) References

Operating System Concepts by Abraham Silberschatz

Python Documentation (https://docs.python.org/)

Matplotlib Documentation (https://matplotlib.org/stable/contents.html)

Appendix

A. AI-Generated Project Elaboration/Breakdown Report

Project Title: Page Replacement Algorithms Simulator

Overview:

This project simulates and compares three-page replacement algorithms commonly used in operating systems: FIFO (First in First Out), LRU (Least Recently Used), and Optimal. The simulator enables users to input parameters such as page reference sequences and the number of frames (available memory slots). It then calculates the page faults for each algorithm and provides a graphical representation of the results, helping users compare the algorithms' performance.

The primary goal is to help users understand the differences in how these algorithms manage memory and handle page faults, thereby optimizing RAM usage.

Detailed Breakdown of the Project

1. **Input Handling Module**
    - **Purpose:** Collects the user's input data to run the simulations.
    - **Implementation:**
        - Takes two inputs:
            - Number of frames (available slots in memory).
            - Page reference sequence (sequence of page requests).
        - Validates the inputs to ensure correctness and handles any user errors.
2. **Page Replacement Algorithm Module**
    - **Purpose:** Implements the FIFO, LRU, and Optimal page replacement algorithms.
    - **Implementation:**
        - **FIFO (First In, First Out):** Replaces the oldest page in memory first.
        - **LRU (Least Recently Used):** Replaces the page that hasn't been used for the longest time.
        - **Optimal Algorithm:** Predicts future page references and replaces the page that will not be needed for the longest period.
    - Each algorithm computes the number of pages faults and stores the results for further use.
3. **File Handling Module**
    - **Purpose:** Stores the results of the page fault calculations for later visualization.
    - **Implementation:**
        - Writes the page fault data for each algorithm into a file (results.txt).
        - This data is then read by a Python script for generating the graphical output.
4. **Visualization Module (Python)**
    - **Purpose:** Generates visual comparisons of the algorithms' performance using graphs.
    - **Implementation:**
        - Reads data from results.txt (page fault counts for FIFO, LRU, and Optimal).

        - Uses matplotlib to create a bar chart comparing the performance of each algorithm.
        - Integrated into the Tkinter GUI for dynamic user interaction.
5. **GUI Integration (Tkinter)**
    - **Purpose:** Provides an interactive interface for users to input data and view results.
    - **Implementation:**
        - A Tkinter window is created to display the results in the form of a bar chart.
        - Includes a "Refresh" button that updates the results dynamically when new simulations are run.
        - Offers a smooth user experience with easy navigation and visualization of results.
6. **GitHub Integration & Version Control**
    - **Purpose:** Tracks the code changes and provides a collaborative platform for development.
    - **Implementation:**
        - Maintains a GitHub repository for the project (PageReplacementSimulator).
        - Uses Git for version control, with detailed commit messages and version tracking.
        - The README file provides clear instructions on how to run the project and contribute.

Key Features & Enhancements

- **User-Friendly Input Handling:** Simple interface for users to input data for the simulations.
- **Efficient Algorithm Implementation:** FIFO, LRU, and Optimal algorithms are efficiently implemented for performance comparisons.
- **File-Based Result Storage:** Saves results for further analysis and tracking.
- **Interactive Visualization:** Graphs and animations make comparing algorithm performance intuitive.
- **GUI-Based Interface:** Makes the simulator interactive and easy to use.
- **GitHub Integration:** Ensures smooth version control and collaboration on the project.

Summary:

The **Page Replacement Algorithms Simulator** is an educational tool designed to help users understand and compare different page replacement algorithms used in operating systems. By utilizing C++ for the core algorithm implementation and Python for visualization, the project bridges the gap between theoretical concepts and practical, real-time memory management simulation. With features like interactive graphs, a user-friendly GUI, and comprehensive version control via GitHub, this project provides an engaging learning experience for users and developers alike.

A. Problem Statement

Design a simulator that allows users to test and compare different page replacement algorithms (FIFO, LRU, Optimal). The simulator should provide visualizations and

performance metrics to aid in understanding algorithm efficiency.

B. Solution/ Code

```
import matplotlib.pyplot as plt # Importing
matplotlib library

# FIFO (First-In, First-out) page replacements
Algo
def Calculate_fifo_faults(pages, total_frame):
  #Initialize the variable count page fault
    count = 0
    #store pages in memory
    memory_queue = []

    for page in pages:
        if page not in memory_queue:
            if len(memory_queue) < total_frame:
                memory_queue.append(page)
            else:
                memory_queue.pop(0)
                memory_queue.append(page)
            count += 1  # count pages fault

    return count # return total no of page faults

# LRU (Least Recently Used) Algorithm
def Calculate_lru_faults(pages, total_frame):
    fault_count = 0
    recent_page = []

    for page in pages:
        if page not in recent_page:
            if len(recent_page) < total_frame:
                recent_page.append(page)
            else:
                recent_page.pop(0)
                recent_page.append(page)
            fault_count += 1
        else:
            recent_page.remove(page)
            recent_page.append(page)

    return fault_count

# Optimal Page Replacement Algo
def optimal_page_rep(pages, total_frame):
    faults_count = 0
```

```
    memory = []

    for i, page in enumerate(pages):
        if page not in memory:
            if len(memory) < total_frame:
                memory.append(page)
            else:
                future_page = pages[i+1:]
                indices = [future_page.index(p) if p in
future_page else float('inf') for p in memory]
                memory.pop(indices.index(max(indices)
))
                memory.append(page)
            faults_count += 1

    return faults_count

# Main Function to Handle User Input
def main():
    page = list(map(int, input("Enter ref string
(space-separated): ").split()))
    frame = int(input("Enter number of frame: "))

    count = Calculate_fifo_faults(page, frame)
    fault_count = Calculate_lru_faults(page, frame)
    faults_count = optimal_page_rep(page, frame)

    print(f"FIFO Page Faults: {count}")
    print(f"LRU Page Faults: {fault_count}")
    print(f"Optimal Page Faults: {faults_count}")

    # Graphical Representation
    algorithms = ["FIFO", "LRU", "Optimal"]
    fault = [count, fault_count, faults_count]

    plt.bar(algorithms, fault, color=['blue', 'red',
'green'])
    plt.xlabel("Page Replacement Algorithms")
    plt.ylabel("Page Faults")
    plt.title("Page Replacement Performance")
    plt.show()

if __name__ == "__main__":
    main()
```