ACADEMIC TASK - 2

CSE316

(Operating Systems)

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Name: P. Hari Teja

Registration number: 12318279

Roll No.: 28

Section: K23RX

Submitted to:

Gagandeep Kaur Ma’am

LOVELY PROFESSIONAL UNIVERSITY

Efficient Page Replacement Simulator

1) Project Overview:

Efficient memory management is crucial for operating systems, and **page replacement algorithms** help in optimizing **RAM utilization** by replacing pages efficiently when a page fault occurs.

- This project aims to **simulate and compare** three widely used **page replacement algorithms**:
  - **FIFO (First In First Out)**
  - **LRU (Least Recently Used)**
  - **Optimal Page Replacement**
- The simulator allows users to input parameters, execute the algorithms, and visualize their efficiency using **interactive graphs** and **GUI-based animations**. It provides **clear insights into algorithm performance** based on **total page faults**.
- Additionally, the simulator enhances user understanding by providing **step-by-step memory allocation tracking**, allowing users to see how pages are loaded and replaced in real-time, making complex concepts more intuitive and engaging.

2) Module-Wise Breakdown:

Module	Description
C++ Core Implementation	Implements FIFO, LRU, and Optimal page replacement algorithms.
File Handling (results.txt)	Saves page faults count for visualization.
Python Data Visualization	Generates a bar chart to compare algorithm efficiency.
Tkinter GUI Integration	Provides an interactive user interface for visualization.
GitHub Version Control	Tracks code changes and ensures collaborative development.

3) Functionalities:

1. Takes user input for the number of frames and page reference sequence.
2. Computes page faults for FIFO, LRU, and Optimal algorithms.
3. Saves results in results.txt for further analysis.
4. Generates a visualization using Matplotlib to compare results.
5. Provides a GUI-based visualization using Tkinter.
6. Implements GitHub version control for tracking project updates.

4) Technologies Used:

❖ Programming Languages:

- ***C++***
- ***Python***

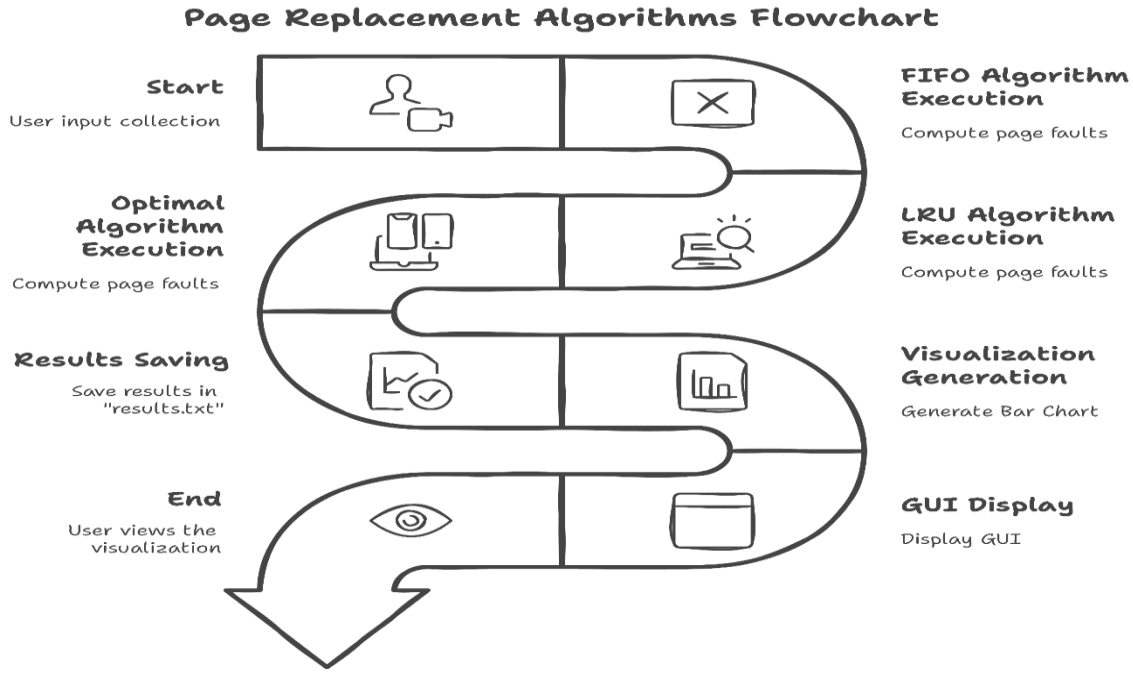
❖ Libraries & Tools:

- ***Matplotlib*** **- Graph visualization for algorithm performance**
- ***Tkinter*** **- GUI-based visualization for enhanced interactivity.**
- ***D3.js & Chart.js*** **- For dynamic graph representation.**

❖ Other Tools:

- ***GitHub*** **- For version control and tracking code revisions.**
- ***VS Code*** **- Code development and debugging.**

5) Flow Diagram:



6) Revision Tracking on GitHub:

- **Repository Name:**
  EfficientPageReplacementSimulator
- **GitHub Link:**
  https://github.com/hariteja-01/EfficientPageReplacementSimulator

7) Conclusion and Future Scope:

A. Conclusion:

This project successfully demonstrates how page replacement algorithms operate by computing total page faults and visualizing their performance. The integration of C++ and Python ensures both efficient execution and interactive visualization, making it easier to understand memory management techniques.

B. Future Scope:

- Adding real-time memory visualization for page replacement.
- Implementing additional algorithms like LFU (Least Frequently Used) and Clock Algorithm.
- Enhancing the GUI with more interactive features like step-by-step execution visualization.
- Expanding the project into a web-based simulation using Flask/Django.

8) References

1. **Operating System Concepts** by Silberschatz, Galvin, Gagne
2. **GeeksforGeeks** - Page Replacement Algorithms
3. **TutorialsPoint** - Memory Management in Operating Systems

Appendix

A. AI-Generated Project Elaboration/Breakdown Report

Below is the purely AI generated report for the project:

- **Project Title: Efficient Page Replacement Algorithm Simulator**
- **Overview:**

This project aims to **simulate and compare** different **page replacement algorithms** used in operating systems. The simulation allows users to input parameters, execute **FIFO, LRU, and Optimal algorithms**, and visualize their performance using **interactive graphs and GUI-based animations**.

The primary objective is to help users **understand the efficiency of different algorithms** in handling memory page replacements, reducing page faults, and optimizing RAM utilization.

🔹 Detailed Breakdown of the Project

1. Input Handling Module

Purpose: Collects user inputs for simulation.

Implementation:

- Takes **two inputs** from the user:
  1. **Number of frames** (available memory slots).
  2. **Page reference sequence** (sequence of memory pages requested by the CPU).
- Validates inputs to prevent errors.

---

2. Page Replacement Algorithm Module

Purpose: Implements **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 was least recently used.
- **Optimal Algorithm:** Predicts future page requests and replaces the page that will not be used for the longest time.

Each algorithm computes the **total number of page faults** and **stores the results** in results.txt.

---

3. File Handling Module

Purpose: Saves computed results for visualization.

Implementation:

- The program writes the **page fault counts** for each algorithm to a file (results.txt).
- This file is used by the **Python script** to generate visualizations.

---

4. Visualization Module (Python)

Purpose: Provides **interactive bar charts and animations** to help users compare algorithms easily.

Implementation:

- Reads results.txt and extracts page fault counts.
- Uses **Matplotlib** to generate a **bar chart** comparing FIFO, LRU, and Optimal algorithms.
- Integrates **Tkinter GUI** to provide an **interactive interface** for users to refresh and view results dynamically.

---

5. GUI Integration (Tkinter)

Purpose: Provides a **user-friendly interface** to visualize results dynamically.

Implementation:

- A Tkinter window is created to display the bar chart inside the GUI.
- Includes a **"Refresh" button** to update results dynamically after running new simulations.

- Enhances user experience with **clean visuals and smooth interactions**.

---

6. GitHub Integration & Version Control
Purpose: Tracks all code changes and provides a repository for collaboration.
Implementation:
- A **GitHub repository** (EfficientPageReplacementSimulator) is maintained.
- Version control is done using git, with **commit messages** for each update.
- The README file contains **detailed instructions** for running the project.

🔹 **Key Features & Enhancements**

✅ **User-Friendly Input Handling** – Ensures seamless interaction.

✅ **Efficient Page Replacement Algorithms** – FIFO, LRU, and Optimal are implemented efficiently.

✅ **File-Based Result Storage** – Saves results for further analysis.

✅ **Interactive Visualization** – Graphs and animations for better understanding.

✅ **GUI-Based Interface** – Enhances the user experience.

✅ **GitHub Integration** – Ensures proper version control and collaboration.

🔹 **Summary:**

The **Efficient Page Replacement Algorithm Simulator** provides an **educational tool** for understanding page replacement strategies. With **C++ for fast computation** and **Python for visualization**, the project successfully bridges the gap between theoretical knowledge and **real-time simulation of memory management in operating systems**.

B. 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.

C. Solution/ Code

C++ Code:

```
#include <bits/stdc++.h>
using namespace std;

// Function to write results to a file
void saveResults(string algo, int pageFaults) {
    ofstream fout("results.txt", ios::app);
    fout << algo << " " << pageFaults << endl;
    fout.close();
```

```
}

// Code for FIFO Page Replacement
void FIFO(vector<int> pages, int frames) {
    queue<int> frameQueue;
    unordered_map<int, bool> pageMap;
    int pageFaults = 0;

    for (int page : pages) {
        if (!pageMap[page]) {
            if (frameQueue.size() >= frames) {
                int oldest = frameQueue.front();
                frameQueue.pop();
                pageMap.erase(oldest);
            }
            frameQueue.push(page);
            pageMap[page] = true;
            pageFaults++;
        }
    }
    cout << "FIFO Page Faults: " << pageFaults << endl;
    saveResults("FIFO", pageFaults);
}

// Code to implement LRU Page Replacement
void LRU(vector<int> pages, int frames) {
    unordered_map<int, int> pageMap;
    int pageFaults = 0;

    for (int i = 0; i < pages.size(); i++) {
        if (pageMap.find(pages[i]) == pageMap.end()) {
            if (pageMap.size() >= frames) {
                int lru = INT_MAX, pageToRemove;
                for (auto p : pageMap) {
                    if (p.second < lru) {
                        lru = p.second;
                        pageToRemove = p.first;
                    }
                }
                pageMap.erase(pageToRemove);
            }
            pageFaults++;
        }
        pageMap[pages[i]] = i;
    }
```

```
    cout << "LRU Page Faults: " << pageFaults << endl;
    saveResults("LRU", pageFaults);
}

// Code for Optimal Page Replacement
void Optimal(vector<int> pages, int frames) {
    vector<int> frame;
    int pageFaults = 0;

    for (int i = 0; i < pages.size(); i++) {
        auto it = find(frame.begin(), frame.end(), pages[i]);
        if (it == frame.end()) { // Page fault
            if (frame.size() >= frames) {
                int farthest = -1, pageToReplace;
                for (int j = 0; j < frame.size(); j++) {
                    int k;
                    for (k = i + 1; k < pages.size(); k++) {
                        if (pages[k] == frame[j]) break;
                    }
                    if (k == pages.size()) {
                        pageToReplace = j;
                        break;
                    }
                    if (k > farthest) {
                        farthest = k;
                        pageToReplace = j;
                    }
                }
                frame[pageToReplace] = pages[i];
            } else {
                frame.push_back(pages[i]);
            }
            pageFaults++;
        }
    }
    cout << "Optimal Page Faults: " << pageFaults << endl;
    saveResults("Optimal", pageFaults);
}

// The main function of cpp to run the above functions.
int main() {
    int frames, n;
    cout << "Enter the number of frames: ";
    cin >> frames;
    cout << "Enter the number of page references: ";
```

```
    cin >> n;

    vector<int> pages(n);
    cout << "Enter the page reference sequence: ";
    for (int i = 0; i < n; i++) {
        cin >> pages[i];
    }
    ofstream fout("results.txt");
    fout.close();

    FIFO(pages, frames);
    LRU(pages, frames);
    Optimal(pages, frames);

    return 0;
}
```

Python Code:

```
import matplotlib.pyplot as plt

with open("results.txt", "r") as file:
    data = file.readlines()

algorithms = []
faults = []
for line in data:
    algo, fault_count = line.split()
    algorithms.append(algo)
    faults.append(int(fault_count))

fig, ax = plt.subplots()
bars = ax.bar(algorithms, faults, color=['orange', 'violet', 'green'])

plt.xlabel("Page Replacement Algorithms")
plt.ylabel("Total Page Faults")
plt.title("Comparison of Page Replacement Algorithms")

annot = ax.annotate("", xy=(0,0), xytext=(10,10), textcoords="offset points",
                    ha="center", va="bottom",
                    bbox=dict(boxstyle="round,pad=0.3", edgecolor="black",
facecolor="white"),
                    fontsize=10, color="black", weight="bold",
                    arrowprops=dict(arrowstyle="wedge,tail_width=0.5", facecolor="black"))
```

```
annot.set_visible(False)

def update_annot(bar):
    x = bar.get_x() + bar.get_width() / 2
    y = bar.get_height()
    annot.xy = (x, y)
    annot.set_text(f"{int(y)}")
    annot.set_visible(True)

def on_hover(event):
    vis = annot.get_visible()
    for bar in bars:
        if bar.contains(event)[0]:
            update_annot(bar)
            fig.canvas.draw_idle()
            return
    if vis:
        annot.set_visible(False)
        fig.canvas.draw_idle()

fig.canvas.mpl_connect("motion_notify_event", on_hover)
plt.show()
```

Full Code in Python:

```
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
from collections import deque

class PageReplacementSimulator:
    def __init__(self, root):
        self.root = root
        self.root.title("Page Replacement Algorithm Simulator")
        self.root.geometry("1000x800")


        self.input_frame = ttk.Frame(root, padding="10")
        self.input_frame.grid(row=0, column=0, sticky="ew", padx=10, pady=10)

        self.result_frame = ttk.Frame(root, padding="10")
        self.result_frame.grid(row=1, column=0, sticky="nsew", padx=10, pady=10)
```

```
        self.root.grid_rowconfigure(1, weight=1)
        self.root.grid_rowconfigure(0, weight=0)
        self.root.grid_columnconfigure(0, weight=1)

        # Input Section
        ttk.Label(self.input_frame, text="Page Reference String (comma-
separated):").grid(row=0, column=0, padx=5, pady=5)
        self.page_entry = ttk.Entry(self.input_frame, width=50)
        self.page_entry.grid(row=0, column=1, padx=5, pady=5)

        ttk.Label(self.input_frame, text="Frame Size:").grid(row=1, column=0, padx=5,
pady=5)
        self.frame_size = ttk.Entry(self.input_frame, width=10)
        self.frame_size.grid(row=1, column=1, padx=5, pady=5, sticky="w")

        ttk.Button(self.input_frame, text="Simulate",
command=self.run_simulation).grid(row=2, column=0, columnspan=2, pady=10)

        # Results Section
        self.notebook = ttk.Notebook(self.result_frame)
        self.notebook.pack(fill="both", expand=True)

        self.tabs = {}
        for algo in ["FIFO", "LRU", "Optimal"]:
            frame = ttk.Frame(self.notebook)
            self.notebook.add(frame, text=algo)
            self.tabs[algo] = frame
# this code implements fifo algo.
    def fifo_algorithm(self, pages, frame_size):
        frames = []
        page_faults = 0
        steps = []

        for page in pages:
            current = frames.copy()
            if page not in frames:
                if len(frames) < frame_size:
                    frames.append(page)
                else:
                    frames.pop(0)
                    frames.append(page)
                page_faults += 1
            steps.append((current, page, page_faults))
        return steps, page_faults
# this code implements lru algo.
```

```
    def lru_algorithm(self, pages, frame_size):
        frames = []
        page_faults = 0
        steps = []
        recent = []

        for page in pages:
            current = frames.copy()
            if page not in frames:
                if len(frames) < frame_size:
                    frames.append(page)
                    recent.append(page)
                else:
                    lru_page = recent.pop(0)
                    frames[frames.index(lru_page)] = page
                    recent.append(page)
                page_faults += 1
            else:
                recent.remove(page)
                recent.append(page)
            steps.append((current, page, page_faults))
        return steps, page_faults
# this implements optimal algo.
    def optimal_algorithm(self, pages, frame_size):
        frames = []
        page_faults = 0
        steps = []

        for i, page in enumerate(pages):
            current = frames.copy()
            if page not in frames:
                if len(frames) < frame_size:
                    frames.append(page)
                else:
                    future = pages[i+1:]
                    replace_idx = self.find_optimal_replace(frames, future)
                    frames[replace_idx] = page
                page_faults += 1
            steps.append((current, page, page_faults))
        return steps, page_faults

    def find_optimal_replace(self, frames, future):
        distances = []
        for frame in frames:
            try:
```

```
                distances.append(future.index(frame))
            except ValueError:
                return frames.index(frame)
        return np.argmax(distances)

    def create_visualization(self, algo, steps, total_faults, pages, frame_size):
        for widget in self.tabs[algo].winfo_children():
            widget.destroy()

        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 7), height_ratios=[1, 1])  # Adjusted figure size

        # Memory state matrix
        states = np.zeros((frame_size, len(pages)))
        for i, (current, page, _) in enumerate(steps):
            for j in range(min(frame_size, len(current))):
                states[j, i] = current[j] if current[j] else 0

        cax = ax1.matshow(states, cmap='viridis')
        fig.colorbar(cax, ax=ax1)
        ax1.set_title(f"{algo} - Memory States")
        ax1.set_xlabel("Reference Number")
        ax1.set_ylabel("Frame Number")

        # Page Faults Plot with Metrics
        faults = [step[2] for step in steps]
        ax2.plot(faults, 'r.-', label='Page Faults')
        #ax2.set_title(f"Page Faults Over Time (Total: {total_faults})")
        ax2.set_xlabel("Reference Number")
        ax2.set_ylabel("Fault Count")
        ax2.legend()

        # Step 4: Performance Metrics
        hit_ratio = 1 - (total_faults / len(pages))
        miss_ratio = 1 - hit_ratio
        metrics_text = (f"Hit Ratio: {hit_ratio:.2%}\n"
                        f"Miss Ratio: {miss_ratio:.2%}\n"
                        f"Total Faults: {total_faults}")
        ax2.text(0.02, 0.98, metrics_text, transform=ax2.transAxes,
                 verticalalignment='top', bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

        plt.tight_layout(pad=3.0)  # Increased padding to prevent overlap
        canvas = FigureCanvasTkAgg(fig, master=self.tabs[algo])
        canvas.draw()
```

```
        canvas.get_tk_widget().pack(fill="both", expand=True)

    def run_simulation(self):
        try:
            pages = [int(x.strip()) for x in self.page_entry.get().split(',')]
            frame_size = int(self.frame_size.get())

            if frame_size <= 0 or not pages:
                raise ValueError("Invalid input")

            fifo_steps, fifo_faults = self.fifo_algorithm(pages, frame_size)
            lru_steps, lru_faults = self.lru_algorithm(pages, frame_size)
            opt_steps, opt_faults = self.optimal_algorithm(pages, frame_size)

            self.create_visualization("FIFO", fifo_steps, fifo_faults, pages, frame_size)
            self.create_visualization("LRU", lru_steps, lru_faults, pages, frame_size)
            self.create_visualization("Optimal", opt_steps, opt_faults, pages, frame_size)

        except ValueError as e:
            messagebox.showerror("Error", f"Invalid input: {str(e)}")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")

if __name__ == "__main__":
    root = tk.Tk()
    app = PageReplacementSimulator(root)
    root.mainloop()
```