ACADEMIC TASK - 2

CSE316

(Operating Systems)

COMPUTER SCIENCE AND ENGINEERING

Section: K23DW

Submitted by:

Name	Reg. No.	Roll No.
Arshi Bansal	12312902	04
Mansi Tyagi	12318015	32
Tanvi Sharma	12313057	15

Submitted to:

Dr. Anudeep Goraya Ma’am

LOVELY PROFESSIONAL UNIVERSITY

<u>Project Report</u>

Efficient Page Replacement Algorithm Simulator

1. Project Overview

Page replacement algorithms play a crucial role in operating systems by efficiently managing limited physical memory. The Efficient Page Replacement Algorithm Simulator is designed to help users test and compare different page replacement strategies such as FIFO (First-In-First-Out), LRU (Least Recently Used), and Optimal Page Replacement. This simulator provides interactive visualizations and performance metrics to enhance users' understanding of algorithm efficiency.

Goals and Expected Outcomes

- Educational Tool: Helps users understand memory management principles.
- Algorithm Comparison: Enables evaluation of different page replacement strategies.
- Performance Metrics: Displays page fault counts, hit ratios, and execution steps.
- User Interaction: Allows customization of memory size and reference string inputs.

Scope

- Simulates FIFO, LRU, and Optimal algorithms.
- Allows users to input custom page reference strings and memory frame sizes.
- Provides step-by-step execution logs and algorithm performance comparison.
- Future expansions include cloud-based deployment, AI recommendations, and additional algorithms.

2. Module-Wise Breakdown

- GUI Module

  The Graphical User Interface (GUI), built using Tkinter, ensures a user-friendly experience. It includes:

  - Input Fields: Users specify the page reference string and number of frames.
  - Algorithm Selection: Dropdown menu for choosing FIFO, LRU, or Optimal.
  - Control Buttons: Execute simulations, compare algorithms, and clear inputs.
  - Output Display: Shows execution logs, hits, and faults.
  - Error Handling: Prevents invalid inputs with appropriate error messages.

- Algorithm Execution Module

This module executes three-page replacement algorithms:

1. FIFO (First-In-First-Out): Replaces the oldest page in memory.
2. LRU (Least Recently Used): Replaces the least recently accessed page.
3. Optimal Page Replacement: Predicts future usage and replaces the page needed farthest in the future.

For each algorithm, this module calculates:

- Page faults and hit ratio.
- Step-by-step memory frame updates.
- Comparison of execution performance.

- Performance Metrics Module

  This module analyzes algorithm efficiency by displaying:

  - Total page faults for each algorithm.
  - Hit ratios (percentage of times a page request is found in memory).
  - Algorithm comparison results.

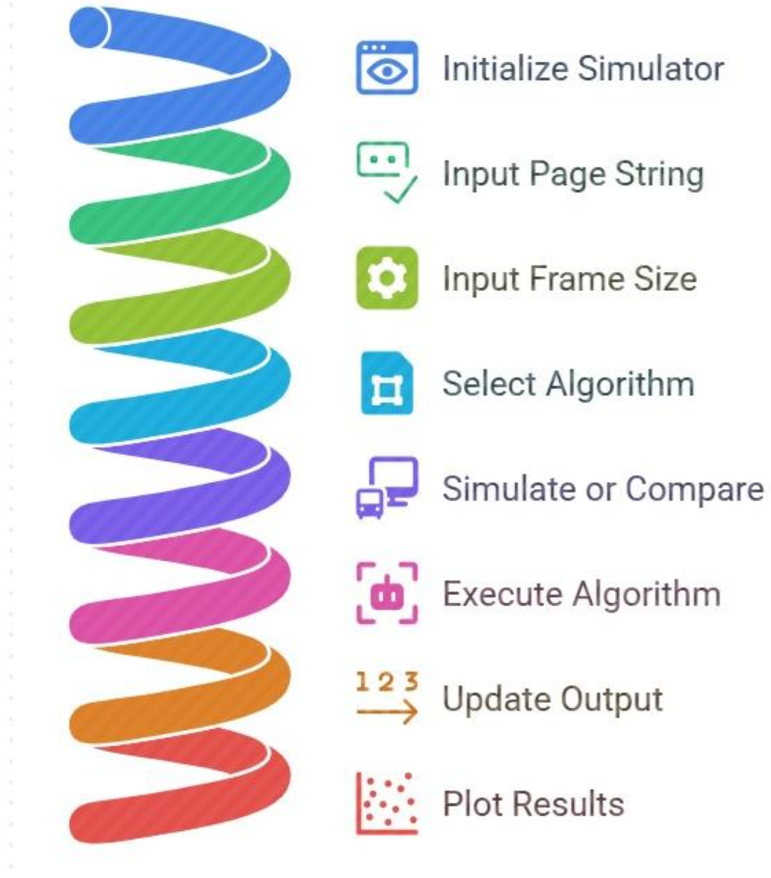
3. Functionalities

- Custom Input Handling: Users enter custom memory reference strings and frame sizes.
- Algorithm Execution & Comparison: Individual execution and comparative analysis.
- Performance Metrics Display: Page faults, hit/miss ratio, and stepwise execution.
- Graphical Representation: (Optional) Matplotlib-based visual performance comparison.
- User-Friendly Design: Intuitive interface with clear output logs.

4. Technology Used

- Programming Language
  - Python: Chosen for its simplicity and library support.
- Libraries & Tools
  - Tkinter: For GUI development.
  - NumPy: Efficient data handling for page replacement simulations.
  - Matplotlib: Optional visualization of execution results.
  - GitHub: Version control system for tracking project progress.

5. Flow Diagram



6. Revision Tracking on GitHub

- Repository Name: PageReplacementSimulator
- GitHub Link: https://github.com/ArshiBansal/PageReplacementSimulator
- Commit History: 9 commits done

7. Conclusion and Future Scope

This project provides a hands-on approach to understanding page replacement algorithms in operating systems. Future improvements may include:

- AI-driven algorithm recommendations.
- Integration of additional algorithms (LFU, Clock, Second-Chance).
- Cloud-based and web-based deployment.

8. References

- Operating Systems Concepts - Silberschatz, Galvin, and Gagne.

- Modern Operating Systems - Andrew S. Tanenbaum.

9. Appendix

A. AI-Generated Project Breakdown Report

- Project Overview:

  The Efficient Page Replacement Algorithm Simulator is designed to help users understand and compare different page replacement strategies such as FIFO, LRU, and Optimal. The simulator provides visual representations and performance metrics, including page faults and hit ratios, making it a useful educational and analytical tool.

- Module-Wise Breakdown:
  - Graphical User Interface (GUI)

    Developed using Tkinter for an intuitive user experience.

    Includes input fields for the page reference string, number of frames, and algorithm selection.

    Buttons for simulation, comparison, and clearing results.

  - Algorithm Processing & Logic

    Implements FIFO, LRU, and Optimal page replacement algorithms.

    Maintains data structures for tracking pages in memory.

    Computes page faults and generates step-wise execution details.

    Data Visualization & Metrics

    Uses Matplotlib for visualizing page frame transitions and algorithm comparisons.

    Displays bar charts comparing page faults across algorithms.

    Plots fault occurrences over time to analyze performance trends.

- Functionalities:
  - GUI Module:
    - User-friendly input fields.
    - Algorithm selection dropdown.
    - Buttons for simulation and comparison.
  - Algorithm Processing:
    - FIFO: Replaces the oldest page in memory.
    - LRU: Replaces the least recently used page.
    - Optimal: Replaces the page that won’t be used for the longest time.
  - Visualization & Metrics:
    - Displays execution steps in text format.
    - Graphs showing faults over time and final memory state.
    - Hit ratio and total faults for performance evaluation.
- Technology Recommendations:

- Programming Language: Python
- GUI Framework: Tkinter
- Visualization: Matplotlib
- Data Handling: NumPy (optional for efficiency)
- Execution Plan:
- Set up the Project Environment
- Install Python and necessary libraries (tkinter, matplotlib, numpy).
- Develop the GUI
- Create input fields, labels, and buttons.
- Integrate event handlers for simulation and comparison.
- Implement Page Replacement Algorithms
- Develop FIFO, LRU, and Optimal logic.
- Track and store page frames and faults.
- Integrate Visualization
- Use Matplotlib to generate graphs for analysis.
- Compare faults across algorithms visually.
- Test and Optimize
- Run multiple test cases.
- Optimize performance and refine the UI.
- Enhancements (if time permits)
- Add additional algorithms (LFU, Clock).
- Export results for further analysis.

B. Problem Statement

Efficient Page Replacement Algorithm Simulator:

Design a simulator that allows users to test and compare different page replacement algorithms (e.g., FIFO, LRU, Optimal). The simulator should provide visualizations and performance metrics to aid in understanding algorithm efficiency.

C. Solution/Code

Python Code

```
import tkinter as tk

from tkinter import ttk, messagebox

import matplotlib

matplotlib.use('TkAgg')  # Force TkAgg backend for reliability

import matplotlib.pyplot as plt

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import numpy as np


class PageReplacementSimulator:

    def __init__(self, root):
```

```
        self.root = root
        self.root.title("Page Replacement Algorithm Simulator")
        self.root.geometry("900x700")
        self.root.configure(bg="#f0f0f0")

        # Title
        title_label = tk.Label(root, text="Page Replacement Simulator", font=("Arial", 18, "bold"), bg="#f0f0f0")
        title_label.pack(pady=10)

        # Input Frame
        input_frame = tk.Frame(root, bg="#f0f0f0")
        input_frame.pack(pady=10)

        # Page String Input
        tk.Label(input_frame, text="Page Reference String (space-separated):", bg="#f0f0f0").grid(row=0, column=0, padx=5, pady=5)
        self.page_entry = tk.Entry(input_frame, width=40)
        self.page_entry.grid(row=0, column=1, padx=5, pady=5)

        # Frame Size Input
        tk.Label(input_frame, text="Number of Frames:", bg="#f0f0f0").grid(row=1, column=0, padx=5, pady=5)
        self.frame_entry = tk.Entry(input_frame, width=10)
        self.frame_entry.grid(row=1, column=1, padx=5, pady=5, sticky="w")

        # Algorithm Selection
        tk.Label(input_frame, text="Select Algorithm:", bg="#f0f0f0").grid(row=2, column=0, padx=5, pady=5)
        self.algo_var = tk.StringVar(value="FIFO")
        algo_menu = ttk.Combobox(input_frame, textvariable=self.algo_var, values=["FIFO", "LRU", "Optimal"], state="readonly")
```

```
        algo_menu.grid(row=2, column=1, padx=5, pady=5, sticky="w")

        # Buttons
        button_frame = tk.Frame(root, bg="#f0f0f0")
        button_frame.pack(pady=10)
        tk.Button(button_frame, text="Simulate", command=self.simulate, bg="#4CAF50",
fg="white", font=("Arial", 10, "bold")).grid(row=0, column=0, padx=10)
        tk.Button(button_frame, text="Compare All", command=self.compare_all,
bg="#2196F3", fg="white", font=("Arial", 10, "bold")).grid(row=0, column=1, padx=10)
        tk.Button(button_frame, text="Clear", command=self.clear, bg="#f44336", fg="white",
font=("Arial", 10, "bold")).grid(row=0, column=2, padx=10)

        # Output Frame
        self.output_frame = tk.Frame(root, bg="#f0f0f0")
        self.output_frame.pack(pady=10, fill="both", expand=True)

        # Text Output (Simulation Steps)
        self.text_output = tk.Text(self.output_frame, height=10, width=80, font=("Courier", 10))
        self.text_output.pack(pady=5)

        # Metrics Label
        self.metrics_label = tk.Label(self.output_frame, text="", font=("Arial", 12),
bg="#f0f0f0")
        self.metrics_label.pack(pady=5)

        # Matplotlib Canvas for Visualization (two subplots)
        self.fig, (self.ax1, self.ax2) = plt.subplots(2, 1, figsize=(6, 6), height_ratios=[1, 1])
        plt.tight_layout(pad=3.0)
        self.canvas = FigureCanvasTkAgg(self.fig, master=self.output_frame)
        self.canvas.get_tk_widget().pack(pady=10)

    def simulate(self):
```

```
        print("Starting simulation...")  # Debug
        self.clear_output()
        try:
            page_string = [int(x) for x in self.page_entry.get().split()]
            frame_size = int(self.frame_entry.get())
            algorithm = self.algo_var.get()
            if frame_size <= 0 or not page_string:
                raise ValueError
        except ValueError:
            messagebox.showerror("Input Error", "Please enter valid page string and frame size.")
            return

        print(f"Page string: {page_string}, Frame size: {frame_size}, Algorithm: {algorithm}")
# Debug
        if algorithm == "FIFO":
            faults, steps, frames_history = self.fifo(page_string, frame_size)
        elif algorithm == "LRU":
            faults, steps, frames_history = self.lru(page_string, frame_size)
        else:  # Optimal
            faults, steps, frames_history = self.optimal(page_string, frame_size)

        # Update text output
        self.text_output.insert(tk.END, "Step | Page | Frames        | Fault/Hit\n")
        self.text_output.insert(tk.END, "-" * 40 + "\n")
        for step in steps:
            self.text_output.insert(tk.END, step + "\n")

        hit_ratio = (len(page_string) - faults) / len(page_string)
        self.metrics_label.config(text=f"Total Page Faults: {faults} | Hit Ratio: {hit_ratio:.2%}")
        print(f"Faults: {faults}, Hit Ratio: {hit_ratio:.2%}")  # Debug
```

```
        self.plot_simulation(page_string, frames_history, steps, algorithm)


    def compare_all(self):
        print("Starting comparison...")  # Debug
        self.clear_output()
        try:
            page_string = [int(x) for x in self.page_entry.get().split()]
            frame_size = int(self.frame_entry.get())
            if frame_size <= 0 or not page_string:
                raise ValueError
        except ValueError:
            messagebox.showerror("Input Error", "Please enter valid page string and frame size.")
            return


        # Run all algorithms
        fifo_faults, fifo_steps, _ = self.fifo(page_string, frame_size)
        lru_faults, lru_steps, _ = self.lru(page_string, frame_size)
        opt_faults, opt_steps, _ = self.optimal(page_string, frame_size)


        # Text output
        self.text_output.insert(tk.END, "Comparison of Algorithms:\n")
        self.text_output.insert(tk.END, "-" * 40 + "\n")
        self.text_output.insert(tk.END, f"FIFO: {fifo_faults} faults, Hit Ratio: {(len(page_string) - fifo_faults) / len(page_string):.2%}\n")
        self.text_output.insert(tk.END, f"LRU: {lru_faults} faults, Hit Ratio: {(len(page_string) - lru_faults) / len(page_string):.2%}\n")
        self.text_output.insert(tk.END, f"Optimal: {opt_faults} faults, Hit Ratio: {(len(page_string) - opt_faults) / len(page_string):.2%}\n")


        # Plot comparison
        self.ax1.cla()
```

```
        self.ax2.cla()


        # Bar chart in ax1
        algorithms = ["FIFO", "LRU", "Optimal"]
        faults = [fifo_faults, lru_faults, opt_faults]
        bars = self.ax1.bar(algorithms, faults, color=["#4CAF50", "#2196F3", "#FF9800"],
edgecolor='black')
        self.ax1.set_title("Total Page Faults Comparison")
        self.ax1.set_ylabel("Number of Faults")
        self.ax1.set_ylim(0, max(faults) + 2)
        for bar in bars:
            height = bar.get_height()
            self.ax1.text(bar.get_x() + bar.get_width()/2., height, f'{int(height)}', ha='center',
va='bottom')


        # Fault/Hit plots in ax2
        steps = list(range(1, len(page_string) + 1))
        fifo_faults_over_time = [1 if "Fault" in step else 0 for step in fifo_steps]
        lru_faults_over_time = [1 if "Fault" in step else 0 for step in lru_steps]
        opt_faults_over_time = [1 if "Fault" in step else 0 for step in opt_steps]
        self.ax2.plot(steps, fifo_faults_over_time, marker="o", label="FIFO",
color="#4CAF50", linestyle='-')
        self.ax2.plot(steps, lru_faults_over_time, marker="o", label="LRU", color="#2196F3",
linestyle='-')
        self.ax2.plot(steps, opt_faults_over_time, marker="o", label="Optimal",
color="#FF9800", linestyle='-')
        self.ax2.set_title("Faults Over Time (All Algorithms)")
        self.ax2.set_xlabel("Step")
        self.ax2.set_ylabel("Fault (1) / Hit (0)")
        self.ax2.set_ylim(-0.2, 1.2)
        self.ax2.legend()
```

```
        self.canvas.draw()
        print("Comparison plotted.")  # Debug

    def plot_simulation(self, page_string, frames_history, steps, algorithm):
        print("Plotting simulation...")  # Debug
        self.ax1.cla()
        self.ax2.cla()

        # Plot final frame state in ax1
        final_frames = frames_history[-1]
        self.ax1.set_title(f"{algorithm} Final Frame State")
        self.ax1.set_ylabel("Frame Content")
        self.ax1.set_ylim(-1, len(final_frames))
        for i, val in enumerate(final_frames):
            self.ax1.text(0, i, str(val) if val != -1 else "-", ha="center", va="center")
        self.ax1.set_xlim(-0.5, 0.5)
        self.ax1.set_xticks([])

        # Plot faults over time in ax2
        faults_over_time = [1 if "Fault" in step else 0 for step in steps]
        self.ax2.plot(range(1, len(page_string) + 1), faults_over_time, marker="o",
label=f"{algorithm} Faults", color="#4CAF50")
        self.ax2.set_title(f"{algorithm} Page Faults Over Time")
        self.ax2.set_xlabel("Step")
        self.ax2.set_ylabel("Fault (1) / Hit (0)")
        self.ax2.legend()

        self.canvas.draw()
        print("Simulation plotted.")  # Debug

    def fifo(self, page_string, frame_size):
```

```
    frames = [-1] * frame_size
    queue = []
    faults = 0
    steps = []
    frames_history = []

    for i, page in enumerate(page_string):
        if page in frames:
            steps.append(f"{i+1:<4} | {page:<4} | {str(frames):<13} | Hit")
        else:
            faults += 1
            if len(queue) < frame_size:
                frames[len(queue)] = page
                queue.append(page)
            else:
                old_page = queue.pop(0)
                frames[frames.index(old_page)] = page
                queue.append(page)
            steps.append(f"{i+1:<4} | {page:<4} | {str(frames):<13} | Fault")
        frames_history.append(frames.copy())
    return faults, steps, frames_history

def lru(self, page_string, frame_size):
    frames = [-1] * frame_size
    recent = []
    faults = 0
    steps = []
    frames_history = []

    for i, page in enumerate(page_string):
```

```
            if page in frames:
                recent.remove(page)
                recent.append(page)
                steps.append(f"{i+1:<4} | {page:<4} | {str(frames):<13} | Hit")
            else:
                faults += 1
                if len(recent) < frame_size:
                    frames[len(recent)] = page
                else:
                    old_page = recent.pop(0)
                    frames[frames.index(old_page)] = page
                recent.append(page)
                steps.append(f"{i+1:<4} | {page:<4} | {str(frames):<13} | Fault")
            frames_history.append(frames.copy())
        return faults, steps, frames_history

    def optimal(self, page_string, frame_size):
        frames = [-1] * frame_size
        faults = 0
        steps = []
        frames_history = []

        for i, page in enumerate(page_string):
            if page in frames:
                steps.append(f"{i+1:<4} | {page:<4} | {str(frames):<13} | Hit")
            else:
                faults += 1
                if len([f for f in frames if f != -1]) < frame_size:
                    frames[frames.index(-1)] = page
                else:
```

```
                future = page_string[i+1:]
                replace_idx = 0
                max_dist = -1
                for j, frame in enumerate(frames):
                    dist = future.index(frame) if frame in future else len(future)
                    if dist > max_dist:
                        max_dist = dist
                        replace_idx = j
                frames[replace_idx] = page
            steps.append(f"{i+1:<4} | {page:<4} | {str(frames):<13} | Fault")
        frames_history.append(frames.copy())
    return faults, steps, frames_history

def clear_output(self):
    self.text_output.delete(1.0, tk.END)
    self.metrics_label.config(text="")
    self.ax1.cla()
    self.ax2.cla()
    self.canvas.draw()
    print("Output cleared.")  # Debug

def clear(self):
    self.page_entry.delete(0, tk.END)
    self.frame_entry.delete(0, tk.END)
    self.clear_output()

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