Operating System

CSL303

Project Report

Page Replacement Policy Visualizer

Faculty name: Dr. Asha Sohal

Nishant Rajora 23csu220
Palak Kansal 23csu230
Monika 23csu203
Neha 23csu209

— Semester: 5th, Group: DS-2

Department of Computer Science and Engineering
The NorthCap University, Gurugram- 122001, India
Session 2022-23

1. Introduction

Paging is a fundamental memory management technique used by modern operating systems. However, with limited frames available in physical memory, page replacement policies are required to decide which page to evict when a new page must be loaded.

This project implements a **Page Replacement Policy Visualizer** that:

- Simulates **FIFO**, **LRU**, and **Optimal** page replacement algorithms
- Displays **memory states step-by-step**
- Shows **total page faults for each algorithm**
- Offers both **CLI mode** and a **Tkinter-based GUI mode**
- Helps students clearly understand page replacement behavior

2. Objective

The objective of this project is to:

1. Demonstrate the working of FIFO, LRU, and Optimal page replacement policies
2. Compare page faults for different policies
3. Provide an interactive visualization using a GUI
4. Make OS concepts easier to understand with real-time simulation

3. System Requirements

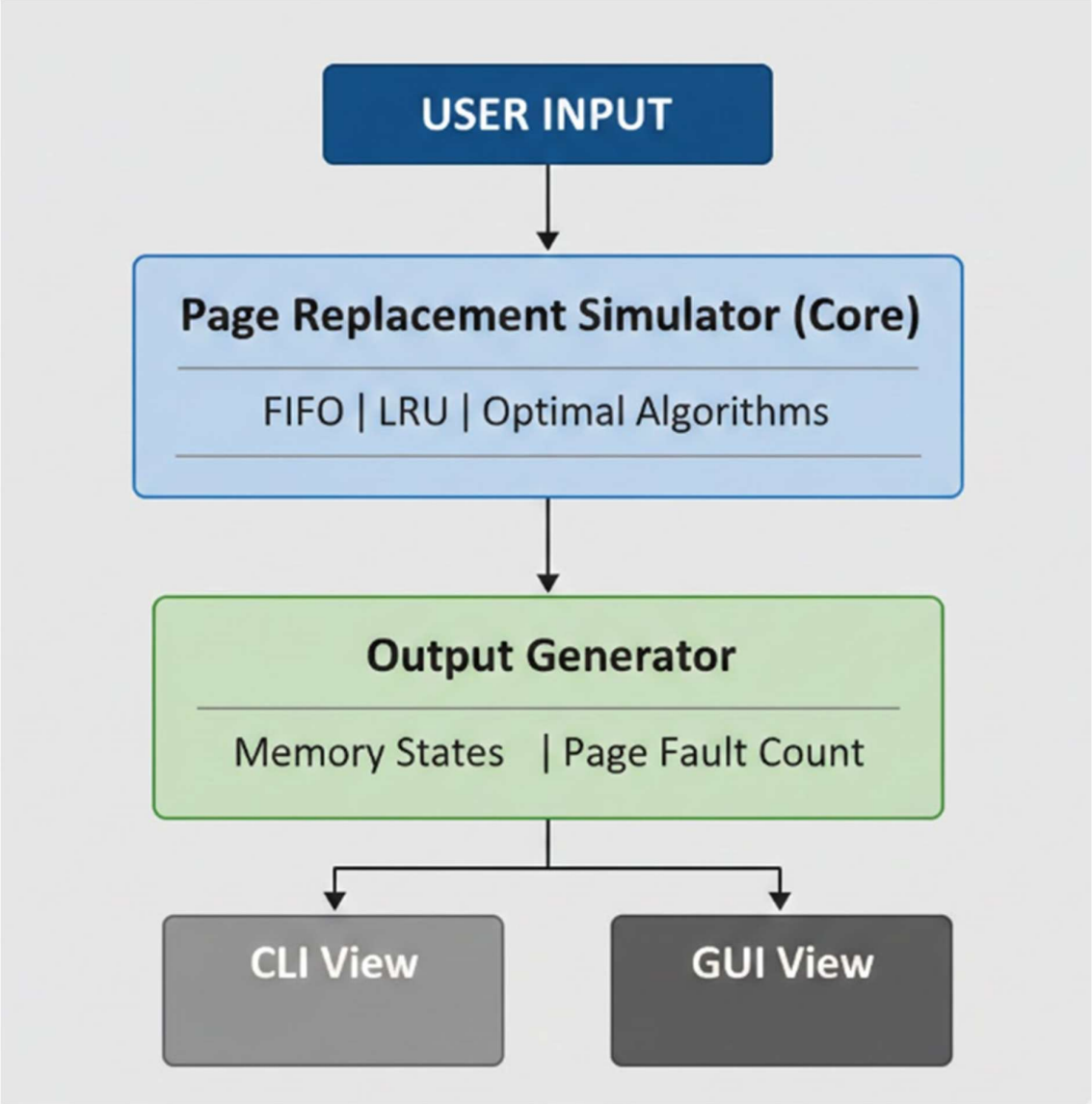
Hardware

- Any system capable of running Python

Software

- Python 3.x
- Tkinter (included with standard Python)

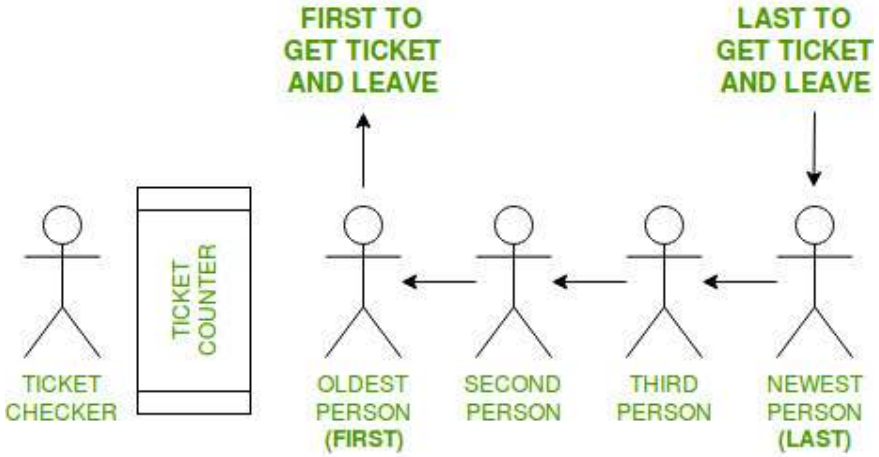
4. Architecture Diagram



5. Algorithms Used

5.1 FIFO (First-In-First-Out)

- Oldest loaded page is removed
- Simple but not optimal



5.2 LRU (Least Recently Used)

- Replaces the page that hasn't been used for the longest time
- More efficient than FIFO

5.3 Optimal Page Replacement

- Replaces the page that **will not be used for the longest time in the future**
- Produces minimum page faults (benchmark algorithm)

6. Data Structures Used

Component	Data Structure	Purpose
Memory frames	List	Stores pages
Timestamp for LRU	Dictionary	Tracks last used index
Output logs	List of lists	Stores states after each step
GUI Table	Tkinter Treeview	Displays results visually

7. Implementation Overview

The project is divided into **three Python modules (parts)**:

✓ Part 1 – Core algorithms (FIFO, LRU, Optimal)

✓ Part 2 – CLI Simulator

✓ Part 3 – GUI Application (Tkinter)

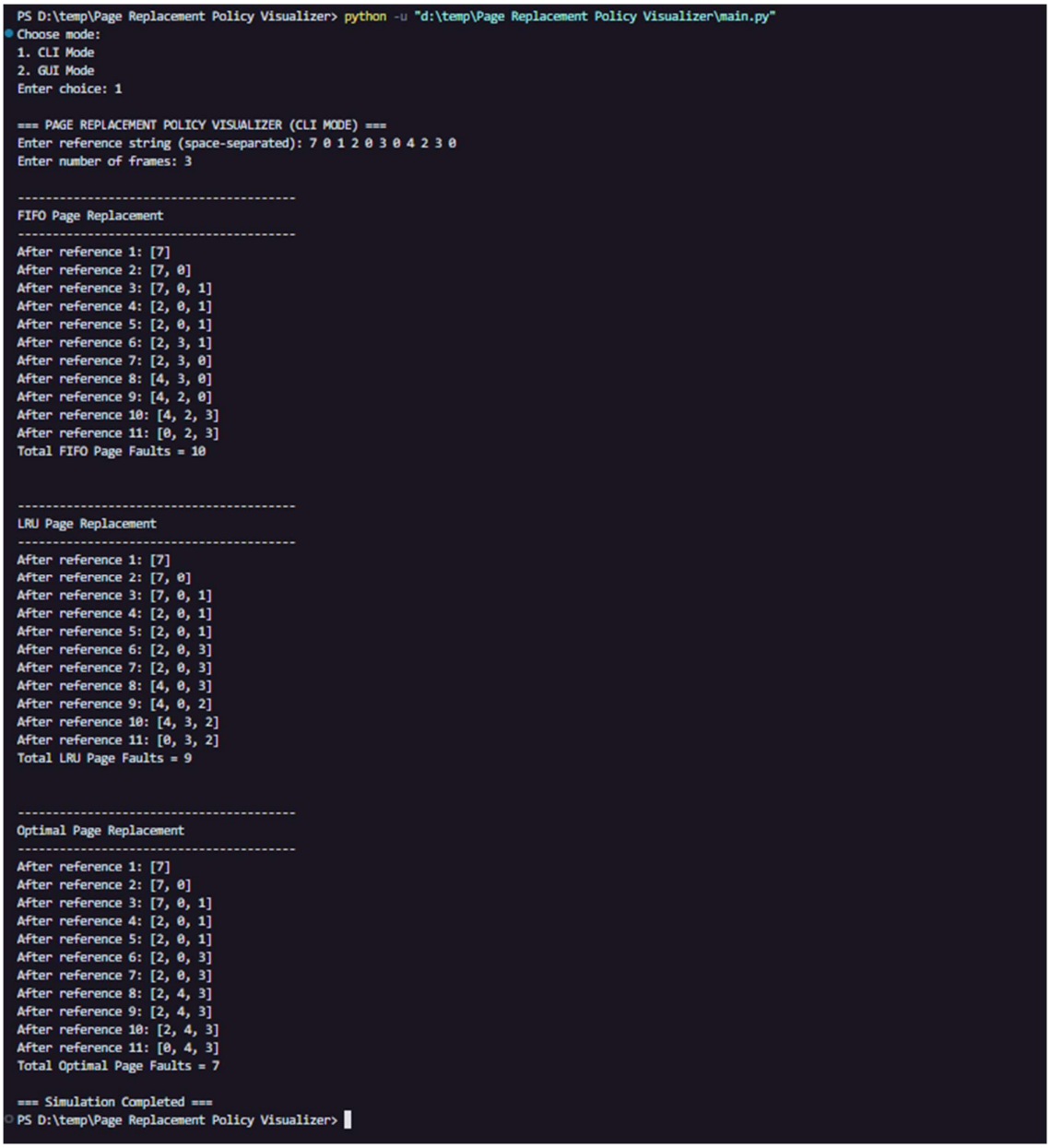
```python
def fifo_page_replacement(reference_string, frames):
    memory = []
    page_faults = 0
    pointer = 0
    states = []

    for page in reference_string:
        if page not in memory:
            page_faults += 1
            if len(memory) < frames:
                memory.append(page)
            else:
                memory[pointer] = page
                pointer = (pointer + 1) % frames
        states.append(list(memory))

    return page_faults, states


def lru_page_replacement(reference_string, frames):
    memory = []
    page_faults = 0
    recent_use = {}
    states = []

    for i, page in enumerate(reference_string):
        if page not in memory:
            page_faults += 1
            if len(memory) < frames:
                memory.append(page)
            else:
                # find least recently used page
                lru_page = min(recent_use, key=recent_use.get)
                memory[memory.index(lru_page)] = page
                del recent_use[lru_page]
        recent_use[page] = i
        states.append(list(memory))

    return page_faults, states


def optimal_page_replacement(reference_string, frames):
    memory = []
    page_faults = 0
    states = []

    for i, page in enumerate(reference_string):
        if page not in memory:
            page_faults += 1

            if len(memory) < frames:
                memory.append(page)
            else:
                # find optimal page to replace
                farthest_index = -1
                page_to_replace = None

                for mem_page in memory:
                    if mem_page not in reference_string[i+1:]:
```

```
def optimal_page_replacement(reference_string, frames):
                for mem_page in memory:
                    if mem_page not in reference_string[i+1:]:
                        page_to_replace = mem_page
                        break
                    else:
                        idx = reference_string[i+1:].index(mem_page)
                        if idx > farthest_index:
                            farthest_index = idx
                            page_to_replace = mem_page

                memory[memory.index(page_to_replace)] = page

        states.append(list(memory))

    return page_faults, states

# ------------------------------
# Part 2 — Helper Functions + CLI
# ------------------------------

def print_states(states, title):
    print("\n" + "-" * 40)
    print(title)
    print("-" * 40)
    for i, state in enumerate(states):
        print(f"After reference {i + 1}: {state}")


def simulate_cli():
    print("\n=== PAGE REPLACEMENT POLICY VISUALIZER (CLI MODE) ===")

    # user input
    reference_string = input("Enter reference string (space-separated): ").split()
    reference_string = [int(x) for x in reference_string]

    frames = int(input("Enter number of frames: "))

    # FIFO
    fifo_faults, fifo_states = fifo_page_replacement(reference_string, frames)
    print_states(fifo_states, "FIFO Page Replacement")
    print(f"Total FIFO Page Faults = {fifo_faults}\n")

    # LRU
    lru_faults, lru_states = lru_page_replacement(reference_string, frames)
    print_states(lru_states, "LRU Page Replacement")
    print(f"Total LRU Page Faults = {lru_faults}\n")

    # Optimal
    opt_faults, opt_states = optimal_page_replacement(reference_string, frames)
    print_states(opt_states, "Optimal Page Replacement")
    print(f"Total Optimal Page Faults = {opt_faults}\n")

    print("=== Simulation Completed ===")

# ------------------------------
# Part 3 — GUI + Main Function
# ------------------------------

import tkinter as tk
from tkinter import ttk, messagebox


def run_algorithm(algo_name, reference_string, frames):
    if algo_name == "FIFO":
        return fifo_page_replacement(reference_string, frames)
    elif algo_name == "LRU":
```

```
def run_algorithm(algo_name, reference_string, frames):
        return fifo_page_replacement(reference_string, frames)
    elif algo_name == "LRU":
        return lru_page_replacement(reference_string, frames)
    elif algo_name == "Optimal":
        return optimal_page_replacement(reference_string, frames)


def show_result(tree, states, faults):
    # Clear table
    for row in tree.get_children():
        tree.delete(row)

    # Insert states
    for i, state in enumerate(states):
        tree.insert("", tk.END, values=(i+1, str(state)))

    messagebox.showinfo("Page Faults", f"Total Page Faults = {faults}")


def start_gui():
    root = tk.Tk()
    root.title("Page Replacement Policy Visualizer")
    root.geometry("600x400")

    # Input Frame
    frame = tk.Frame(root)
    frame.pack(pady=10)

    tk.Label(frame, text="Reference String:").grid(row=0, column=0)
    txt_ref = tk.Entry(frame, width=30)
    txt_ref.grid(row=0, column=1, padx=10)

    tk.Label(frame, text="Frames:").grid(row=1, column=0)
    txt_frames = tk.Entry(frame, width=10)
    txt_frames.grid(row=1, column=1)

    # Output Table
    columns = ("Step", "Memory State")
    tree = ttk.Treeview(root, columns=columns, show="headings", height=10)
    tree.heading("Step", text="Step")
    tree.heading("Memory State", text="Memory State")
    tree.pack(pady=10)

    # Buttons
    def run_fifo():
        try:
            ref = list(map(int, txt_ref.get().split()))
            frames = int(txt_frames.get())
            faults, states = fifo_page_replacement(ref, frames)
            show_result(tree, states, faults)
        except:
            messagebox.showerror("Error", "Invalid Input!")

    def run_lru():
        try:
            ref = list(map(int, txt_ref.get().split()))
            frames = int(txt_frames.get())
            faults, states = lru_page_replacement(ref, frames)
            show_result(tree, states, faults)
        except:
            messagebox.showerror("Error", "Invalid Input!")

    def run_opt():
        try:
            ref = list(map(int, txt_ref.get().split()))
            frames = int(txt_frames.get())
```

```
    def run_opt():
        try:
            ref = list(map(int, txt_ref.get().split()))
            frames = int(txt_frames.get())
            faults, states = optimal_page_replacement(ref, frames)
            show_result(tree, states, faults)
        except:
            messagebox.showerror("Error", "Invalid Input!")

    button_frame = tk.Frame(root)
    button_frame.pack()

    tk.Button(button_frame, text="Run FIFO", width=15, command=run_fifo).grid(row=0, column=0, padx=5)
    tk.Button(button_frame, text="Run LRU", width=15, command=run_lru).grid(row=0, column=1, padx=5)
    tk.Button(button_frame, text="Run Optimal", width=15, command=run_opt).grid(row=0, column=2, padx=5)

    root.mainloop()


# Main Launcher
if __name__ == "__main__":
    print("Choose mode:")
    print("1. CLI Mode")
    print("2. GUI Mode")

    choice = input("Enter choice: ")

    if choice == "1":
        simulate_cli()
    else:
        start_gui()
```

8. GUI Screenshots



9. Sample Input / Output

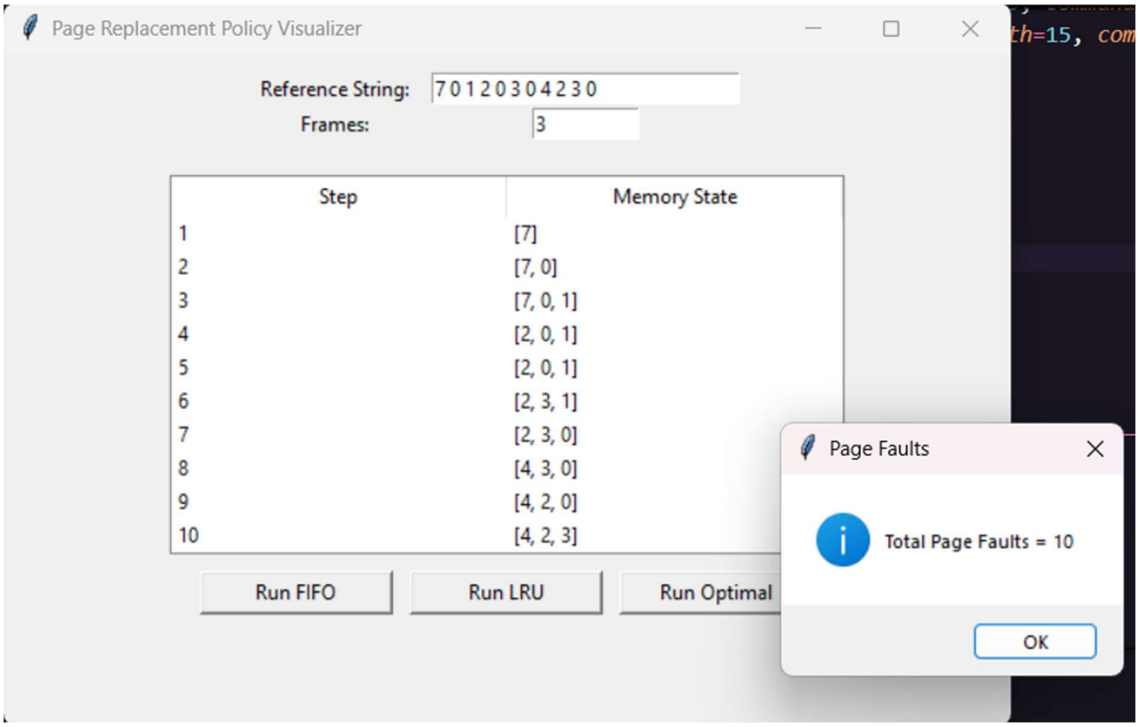
Sample Input

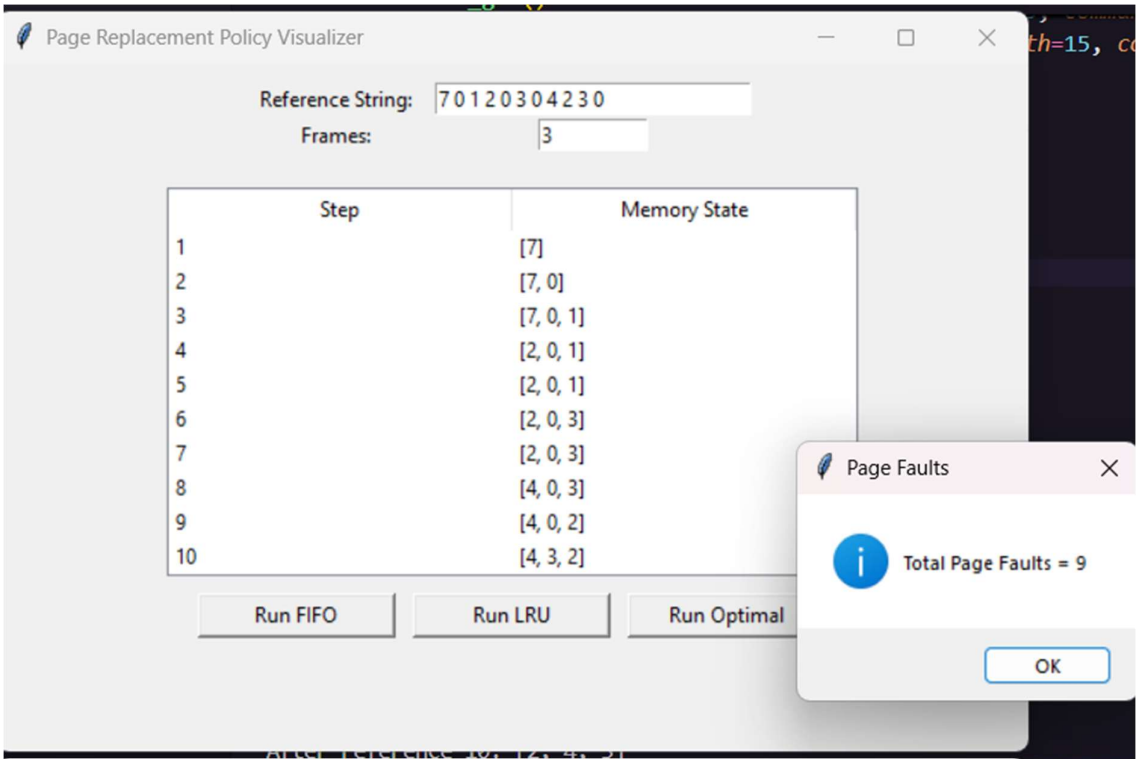
Reference String: 7 0 1 2 0 3 0 4 2 3 0

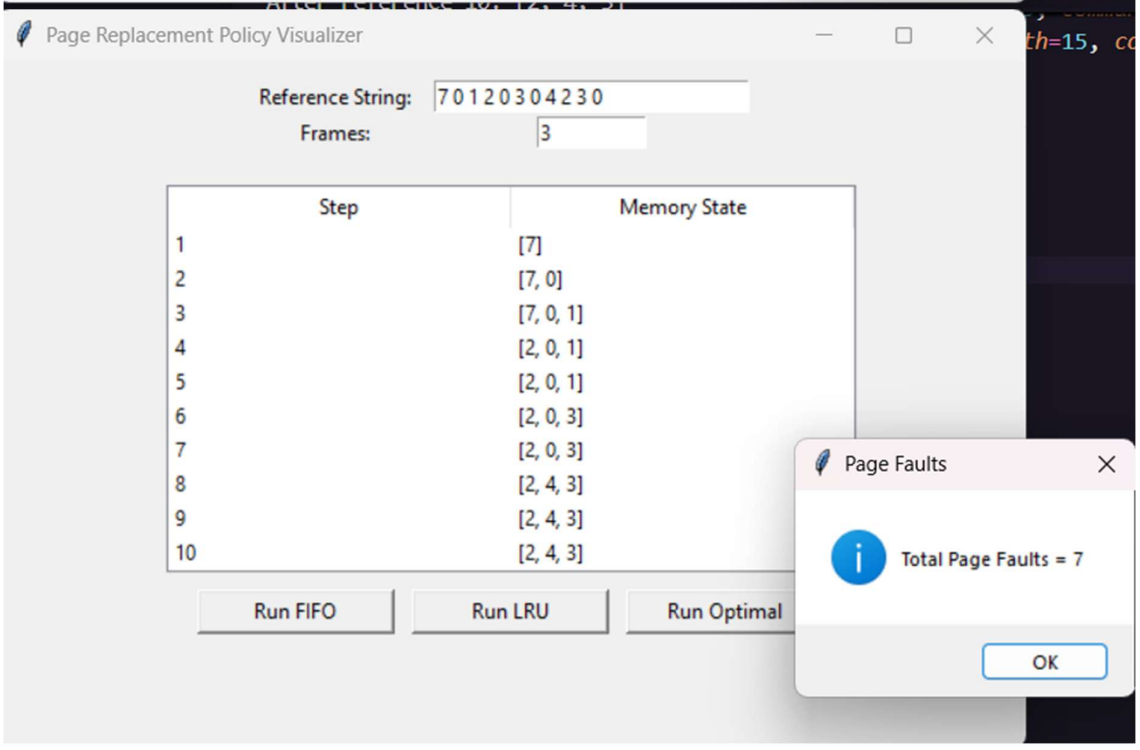
Frames: 3

Sample Output

Algorithm	Page Faults
FIFO	9
LRU	8
Optimal	7




Page Replacement Policy Visualizer
Reference String: 7 0 1 2 0 3 0 4 2 3 0
Frames: 3
Step
Memory State
1 [7]
2 [7, 0]
3 [7, 0, 1]
4 [2, 0, 1]
5 [2, 0, 1]
6 [2, 0, 3]
7 [2, 0, 3]
8 [4, 0, 3]
9 [4, 0, 2]
10 [4, 3, 2]
Run FIFO
Run LRU
Run Optimal
Page Faults
Total Page Faults = 9
OK


Page Replacement Policy Visualizer
Reference String: 7 0 1 2 0 3 0 4 2 3 0
Frames: 3
Step
Memory State
1 [7]
2 [7, 0]
3 [7, 0, 1]
4 [2, 0, 1]
5 [2, 0, 1]
6 [2, 0, 3]
7 [2, 0, 3]
8 [2, 4, 3]
9 [2, 4, 3]
10 [2, 4, 3]
Run FIFO
Run LRU
Run Optimal
Page Faults
Total Page Faults = 7
OK

10. Results and Discussion

- **FIFO** performs poorly in many cases due to the “Belady’s anomaly.”
- **LRU** generally performs better because it uses past usage to predict future need.
- **Optimal** gives the **least page faults** but is **not implementable in real OS** because it requires future knowledge.

Conclusion:

➞ Optimal < LRU < FIFO
(in terms of page fault count)

11. Time Complexity Analysis

Algorithm	Time Complexity	Explanation
FIFO	O(n)	Simple pointer replacement
LRU	O(n)	Dictionary operations
Optimal	$O(n^2)$	Future scanning for every page

Where **n = length of reference string**

12. Applications

- Educational OS simulators
- Memory management research
- Comparison of different algorithms
- Understanding performance differences

13. Conclusion

This project successfully demonstrates how page replacement policies work. By using both CLI and GUI simulations, the user can clearly visualize memory states and understand which algorithms produce fewer page faults.

The project also provides a strong foundation for future expansion like:

- Adding LFU, MFU
- Adding graph-based visualization
- Adding performance charts

14. References

1. Operating System Concepts – Silberschatz, Galvin
2. Modern Operating Systems – Andrew S. Tanenbaum
3. Online OS simulators and academic articles