ACADEMIC TASK - 2

CSE316

(Operating Systems)

COMPUTER SCIENCE AND ENGINEERING

Submitted to:

Hardeep Kaur Mam

Section: K23BM

Submitted by:

Name	**Roll No.**	**Registration No.**
Ayush Kumar	09	12322101
Ansh Babel	61	12322150
Harsh Raj	22	12303684

LOVELY PROFESSIONAL UNIVERSITY

Interactive Scheduling for Page Replacement Algorithm

1. Project Overview:

Managing memory efficiently is a big deal in computing. One tricky part of this is deciding which memory pages to replace when new ones need to be loaded. This project, Interactive Scheduling for Page Replacement Algorithm, is all about simulating different page replacement methods so users can see how they work in real time.

Users can enter a sequence of page requests and choose from common page replacement techniques like **First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal Page Replacement**. The tool then runs the algorithms and shows key stats like the number of page faults, successful hits, and memory states at each step. This hands-on approach helps users get a better feel for how each algorithm behaves and how they compare in different situations.

---

2. Module-Wise Breakdown

The simulator's architecture is a blend of simplicity and purpose, organized into logical modules that handle distinct aspects of its operation. While the codebase is compact, its components work seamlessly to deliver a robust experience. Here's how it breaks down:

Algorithm Implementations

This module houses the core logic of the page replacement strategies, encapsulated within the Page Replacement class as static methods:

- **FIFO Algorithm**: Mimics a queue where the first page in is the first out. It's straightforward but can falter if early pages are frequently reused.
- **LRU Algorithm**: Tracks page usage with an Ordered, replacing the least recently accessed page. It's intuitive for workloads with temporal locality—like a user revisiting recent files.
- **Optimal Algorithm**: Uses future knowledge to replace the page that won't be needed for the longest time. While impractical in real systems (due to its need for foresight), it serves as a benchmark.

Each method takes the reference string and frame count as input, returning page faults, hits, and a list of memory states per step. These implementations are the beating heart of the simulator, translating theoretical rules into tangible outcomes.

Simulation Logic:

Nestled in the main function, this component coordinates the simulation. It processes user inputs, invokes the selected algorithms, and gathers their results. Think of it as the conductor of an orchestra, ensuring each algorithm plays its part before passing the baton to the display modules.

User Interface:

Powered by Streamlit, this module is the user's window into the simulator. It includes:

- A sidebar for entering the reference string, setting frame numbers, and picking algorithms.
- A "Run Simulation" button to kick things off.
- Output sections for results, charts, and detailed steps.

Its design prioritizes ease of use—think of a dashboard where inputs flow naturally into outputs, all without overwhelming the user.

Result Calculation and Visualization:

Once the algorithms finish, this module steps in to crunch numbers and paint pictures. It calculates hit and fault ratios, builds a results table, and generates visuals—bar charts for page faults, pie charts for hit ratios. It also offers an expandable view of each algorithm's steps, letting users peek under the hood. This is where data turns into understanding, much like how a system admin might analyze logs to tweak performance.

By splitting the project into these modules, we keep the code focused and adaptable—ready for future tweaks or expansions.

---

3. Functionalities:-

The simulator isn't just a static tool; it's a dynamic playground packed with features that make exploring page replacement both practical and engaging. Here's what it offers:

- Custom Inputs: Users can define their own reference string (e.g., "7 0 1 2 0 3") and set the number of frames (say, 3 or 5). This mirrors real-world flexibility—think of tweaking memory for a specific app workload.
- Algorithm Choice: Select FIFO, LRU, Optimal, or all three. Want to see how a web browser's caching might differ from a predictive system? Run LRU against Optimal and compare.
- Simulation Trigger: A single "Run Simulation" button starts the process, delivering instant feedback—a nod to the interactive nature of modern tools.

- Performance Metrics: Post-simulation, it spits out:
    - Page Faults: How often memory misses occur.
    - Page Hits: Successful memory accesses.
    - Hit/Fault Ratios: Percentages that show efficiency (e.g., a 60% hit ratio means 6 out of 10 requests were in memory). These stats are gold for understanding trade-offs—like why LRU might edge out FIFO in a file server.
- Visual Insights: Bar charts plot page faults across algorithms, while pie charts break down hit ratios. Picture a manager using these to decide which strategy suits a high-traffic website.
- Step-by-Step Breakdown: An expander reveals each algorithm's moves—page by page, hit or miss. It's like watching a slow-motion replay of memory decisions, perfect for dissecting why Optimal avoids faults better in a predictable sequence.

These features combine to make the simulator a powerful ally for learning and analysis, turning abstract ideas into concrete results.

---

4. Technology Used:-

Building this simulator required a carefully chosen tech stack, balancing functionality with usability. Here's the breakdown:

Programming Languages:

- Python: The backbone of the project, Python shines for its clarity and vast library ecosystem. It's the go-to for rapid prototyping and educational tools like this.

Libraries and Tools

- Streamlit: This gem turns Python scripts into web apps with minimal fuss. It handles the UI, from sliders to charts, letting us focus on logic over layout.
- Pandas: Powers the results table, organizing data into neat DataFrames. It's like Excel for Python—simple yet effective.
- NumPy: Included for potential number-crunching, though lightly used here. It's a safety net for future math-heavy features.
- Matplotlib: Brings visuals to life with bar and pie charts. Its flexibility makes comparing algorithms visually intuitive.

- Collections (OrderedDict): A Python built-in, it’s key to LRU’s logic, tracking page order efficiently.

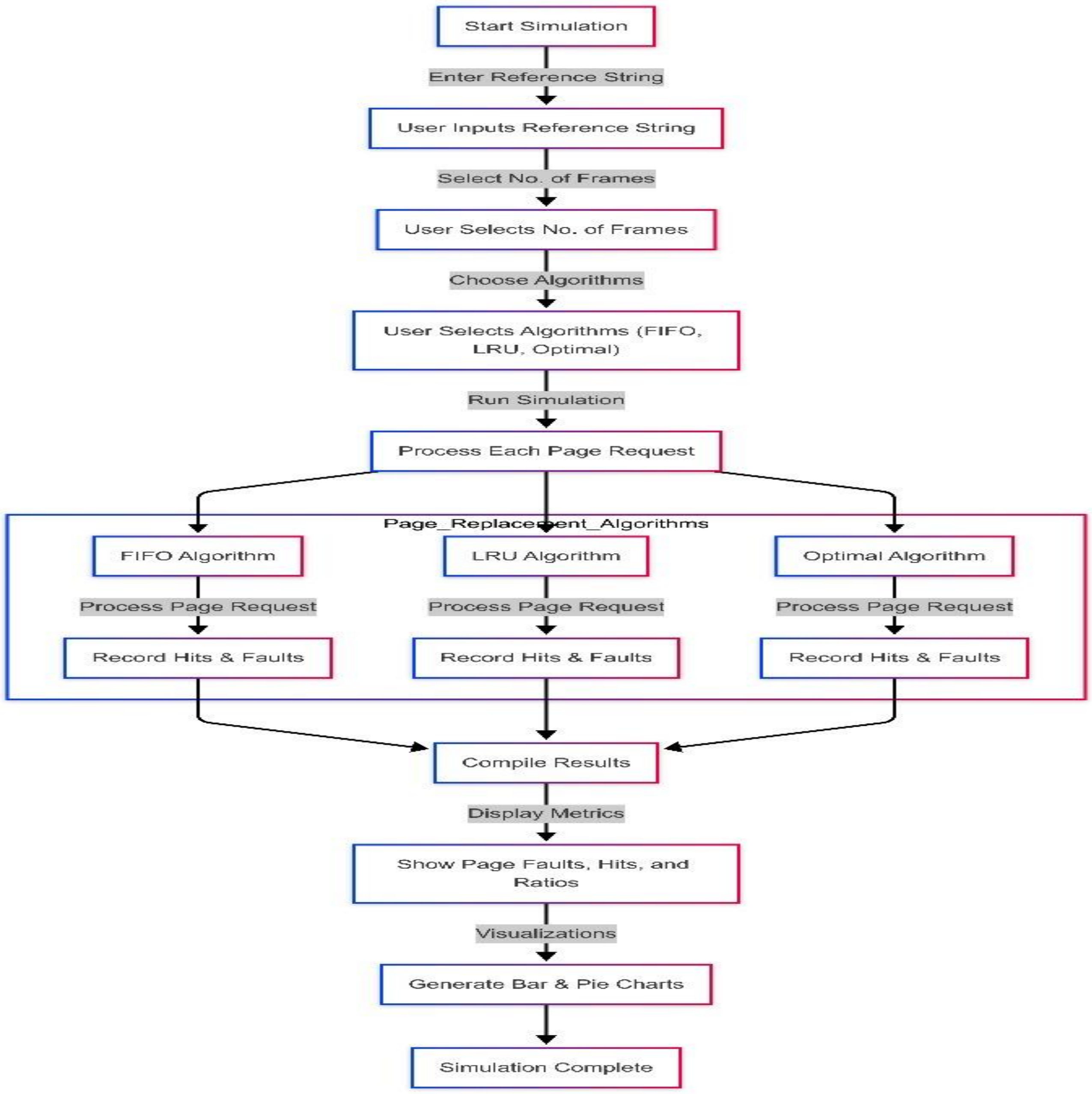
  Other Tools

  •GitHub: Hosts the project, tracking every change. It’s the digital workbench where ideas evolve, from first drafts to polished code.

  This stack ensures the simulator is lightweight yet potent, much like choosing the right tools for a real OS deployment.

---

5. Flow Diagram:-



6. Revision Tracking on GitHub:-

The project lives on GitHub, where its evolution is meticulously tracked. Let's assume it's housed at:

- **Repository Name:** EfficientPageReplacementSimulator
- **GitHub Link:** CodeBy-Ayush/EfficientPageReplacementSimulator: A simulator for efficient page replacement algorithms in OS.Key milestones in its commit history might include:
- Initial Setup: Basic app structure and UI scaffolding.
- Algorithm Additions: FIFO, LRU, and Optimal coded and tested.
- UI Polish: Streamlit tweaks for better usability.
- Visuals: Charts added for faults and ratios.
- Final Touches: Bug fixes, docs, and cleanup.

This versioning mirrors real software development—iterative, transparent, and collaborative.

---

7. Conclusion and Future Scope:

The Interactive Scheduling for Page Replacement Algorithm simulator stands as a testament to how interactive tools can illuminate complex concepts. It delivers a clear, hands-on way to grasp memory management, with results that resonate whether you're studying OS theory or pondering real-world applications like caching in a content delivery network.

Looking forward, the project could grow in exciting ways:

- More Algorithms: Add Clock or Second Chance for broader coverage.
- Dynamic Simulation: Real-time page requests to mimic live systems.
- Enhanced Visuals: Animated memory states for a vivid experience.
- Stats Analysis: Batch testing for deeper performance insights.

These steps could elevate it from a simulator to a full-fledged research tool, pushing the boundaries of interactive learning.

---

8. References:-

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems*. Pearson.
- Streamlit Documentation. Retrieved from **https://docs.streamlit.io/**

Appendix

A. AI-Generated Project Elaboration/Breakdown Report:

This is a well-structured Streamlit application for simulating Page Replacement Algorithms—FIFO, LRU, and Optimal. It provides:

- User Input Handling: Accepts reference strings and number of memory frames.
- Algorithm Implementations: FIFO, LRU, and Optimal are implemented with step-wise memory tracking.
- Results Display: Displays hit/miss counts, performance metrics, and comparisons in tables and charts.
- Interactivity: Allows users to select algorithms and visualize performance.

B. Problem Statement:

Efficient Page Replacement Algorithm Simulator Description:

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:-

Page Replacement Algorithm Simulator

```
import streamlit as st

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from collections import OrderedDict


class PageReplacement:

    @staticmethod

    def fifo_algorithm(pages, total_frames):

        """(FIFO) Page Replacement Algo"""

        queue_fifo = []

        page_faults = 0

        page_hits = 0

        steps = []


        for page in pages:

            if page in queue_fifo:

                page_hits += 1

                steps.append({

                    'page': page,

                    'status': 'Hit',

                    'memory': queue_fifo.copy()

                })

            else:

                page_faults += 1

                if len(queue_fifo) < total_frames:

                    queue_fifo.append(page)

                else:

                    queue_fifo.pop(0)
                    queue_fifo.append(page)

                steps.append({
                    'page': page,
                    'status': 'Miss',
                    'memory': queue_fifo.copy()
                })

        return page_faults, page_hits, steps

    @staticmethod
    def lru_algorithm(pages, total_frames):
        """(LRU) Page Replacement Algo"""
        page_cache = OrderedDict()
        page_faults = 0
        page_hits = 0
        steps = []

        for page in pages:
            if page in page_cache:
                page_hits += 1
                # Move the page to the end to show it's most recently used
                page_cache.move_to_end(page)
                steps.append({
                    'page': page,
                    'status': 'Hit',
                    'memory': list(page_cache.keys())
                })
            else:
                page_faults += 1
                if len(page_cache) >= total_frames:
                    # Remove the least recently used item (first item)
                    page_cache.popitem(last=False)

                page_cache[page] = None
                steps.append({
                    'page': page,
                    'status': 'Miss',
                    'memory': list(page_cache.keys())
                })

        return page_faults, page_hits, steps

    @staticmethod
    def optimal_algorithm(pages, total_frames):
        """Optimal Page Replacement Algo"""
        memory_store = []
        page_faults = 0
        page_hits = 0
        steps = []

        for i, page in enumerate(pages):
            if page in memory_store:
                page_hits += 1
            steps.append({
                'page': page,
                'status': 'Hit',
                'memory': memory_store.copy()
            })
        else:
            page_faults += 1

            if len(memory_store) < total_frames:
                memory_store.append(page)
            else:
                # Find page to replace based on future references
                future_refs = {}
                for mem_page in memory_store:
                    try:
                        future_refs[mem_page] = pages[i+1:].index(mem_page)
                    except ValueError:
                        future_refs[mem_page] = len(pages)

                # Replace the page that will be used furthest in the future
                replace_page = max(future_refs, key=future_refs.get)
                memory_store[memory_store.index(replace_page)] = page

            steps.append({
                'page': page,
                'status': 'Miss',
                'memory': memory_store.copy()
```

```
                })

        return page_faults, page_hits, steps

def main():
    # Page Config
    st.set_page_config(
        page_title="Page Replacement Algorithm Simulator",
        page_icon="💾",
        layout="wide"
    )

    # Title
    st.title("💾 Page Replacement Algorithm Simulator")
    st.markdown("""
    ### Explore Different Page Replacement Strategies
    Compare FIFO, LRU, and Optimal algorithms to understand memory management techniques.
    """)

    # Sidebar Inputs
    st.sidebar.header("🔧 Simulation Parameters")

    # Ref String Input
    default_ref_string = "7 0 1 2 0 3 4 2 3 0 3 2"
    reference_string = st.sidebar.text_input(
        "Enter Reference String",
    value=default_ref_string
)

# Num of Frames
num_frames = st.sidebar.slider(
    "Number of Memory Frames",
    min_value=1,
    max_value=10,
    value=3
)

# Algo Selection
algorithms = st.sidebar.multiselect(
    "Select Algorithms to Simulate",
    ["FIFO", "LRU", "Optimal"],
    default=["FIFO", "LRU", "Optimal"]
)

# Convert ref string to list of int
try:
    pages = list(map(int, reference_string.split()))
except ValueError:
    st.error("Please enter a valid space-separated list of integers.")
    return

# Run Simulation Button
if st.sidebar.button("🚀 Run Simulation"):
        # Results Container
        results_container = st.container()

        with results_container:
            # Performance Metrics
            st.subheader("📊 Performance Metrics")

            # Prepare results DataFrame
            results_data = []

            # Detailed Steps Storage
            detailed_steps = {}

            # Run selected algorithms
            for algo in algorithms:
                if algo == "FIFO":
                    page_faults, page_hits, steps = PageReplacement.fifo_algorithm(pages, 
num_frames)
                elif algo == "LRU":
                    page_faults, page_hits, steps = PageReplacement.lru_algorithm(pages, 
num_frames)
                else:  # Optimal
                    page_faults, page_hits, steps = 
PageReplacement.optimal_algorithm(pages, num_frames)

                # Calculate metrics
                total_pages = len(pages)
                hit_ratio = page_hits / total_pages * 100
        fault_ratio = page_faults / total_pages * 100

        # Store results
        results_data.append({
            'Algorithm': algo,
            'Page Faults': page_faults,
            'Page Hits': page_hits,
            'Hit Ratio (%)': round(hit_ratio, 2),
            'Fault Ratio (%)': round(fault_ratio, 2)
        })

        # Store detailed steps
        detailed_steps[algo] = steps

    # Display Results Table
    results_df = pd.DataFrame(results_data)
    st.table(results_df)

    # Visualization Column
    col1, col2 = st.columns(2)

    # Bar Chart for Page Faults
    with col1:
        st.subheader("Page Faults Comparison")
        plt.figure(figsize=(8, 5))
        plt.bar(results_df['Algorithm'], results_df['Page Faults'])
        plt.title("Page Faults by Algorithm")
```

```
            plt.xlabel("Algorithm")
            plt.ylabel("Number of Page Faults")
            st.pyplot(plt)
            plt.close()

        # Pie Chart for Hit Ratio
        with col2:
            st.subheader("Hit Ratio Comparison")
            plt.figure(figsize=(8, 5))
            plt.pie(
                results_df['Hit Ratio (%)'],
                labels=results_df['Algorithm'],
                autopct='%1.1f%%'
            )
            plt.title("Hit Ratio Distribution")
            st.pyplot(plt)
            plt.close()

        # Detailed Steps Expander
        with st.expander("🔍 Detailed Algorithm Steps"):
            for algo, steps in detailed_steps.items():
                st.subheader(f"{algo} Algorithm Steps")
                steps_df = pd.DataFrame(steps)
                st.dataframe(steps_df)

# Additional Information
st.sidebar.markdown("### 💡 Algorithm Insights")
```

```
    st.sidebar.info("""
    - *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
    """)

def run():
    main()

if _name_ == "_main_":
    run()
```