**ACADEMIC TASK-2**

**Cse316**

*(Operating System)*

**COMPUTER SCIENCE AND ENGINEERING**

**Topic on: Page Replacement Algorithm Simulator**

**Submitted by:**

**Name: Himanshi Chaudhari**

**Registration number: 12318164**

**Roll No.: 58**

**Section: K23GN**

**Submitted to : Dr. Parvinder Singh**

****

**LOVELY PROFESSIONAL UNIVERSITY**

*DECLARATION*

I, Himanshi Chaudhari , a student of Bachelor of Technology under CSE discipline at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is genuine.

**Your name:** *Himanshi Chaudhari*

**Registration Number:** *12318164*

**Phase 3: Report Format**

**1. Project Overview**

In modern operating systems, memory management is crucial for ensuring efficient performance. One of the key challenges is **page replacement**, which determines which page should be swapped out when a new page needs to be loaded into memory.

This project, **"Page Replacement Algorithm Simulator,"** aims to **simulate and compare** different page replacement strategies, such as **FIFO (First-In-First-Out), LRU (Least Recently Used), Optimal, and LFU (Least Frequently Used)**. By visualizing how each algorithm works, the project will help users understand their efficiency in handling memory page requests.

**Goals:**

* Build a **working simulator** that demonstrates different page replacement algorithms in action.
* Allow users to **input a reference string and number of frames** to see how memory is allocated.
* **Track and compare page faults** for each algorithm to evaluate efficiency.
* **Visualize the process** step-by-step, either through text output or graphical representation.

**Expected Outcomes:**

* At the end of this project, we will have:  
   A **fully functional simulator** that runs different page replacement strategies.
* A **step-by-step breakdown** of how memory is managed in each algorithm.
* A **comparison of total page faults** across algorithms.
* A **graphical analysis** to visually compare performance.
* A structured **GitHub repository** for code management and collaboration.

**Scope:**

* Covers **core page replacement policies** (FIFO, LRU, Optimal, LFU).
* Provides **step-by-step execution tracing** of memory allocation.
* Includes **basic visualization** (table-based or graphical).
* Can be **extended** to a **GUI-based or web-based** version in the future.

**2. Module-Wise Breakdown**

| **Module** | **Purpose** |
| --- | --- |
| **1. Input Handling** | Takes user input, validates reference string & frame count. |
| **2. Algorithm Implementation** | Implements FIFO, LRU, Optimal, and LFU page replacement algorithms. |
| **3. Visualization & Performance Analysis** | Displays memory updates, page fault counts, and graphs comparing algorithm efficiency. |

**3. Functionalities**

**Module 1: Input Handling**

* **User Input**: Accepts reference string and number of frames.
* **Validation**: Ensures input is numeric and correctly formatted.

**Example:**

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

Frames: 3

**Validated Output:** [7, 0, 1, 2, 0, 3]

**Module 2: Algorithm Implementation**

* **FIFO**: Replaces the oldest page first.
* **LRU**: Replaces the least recently used page.
* **Optimal**: Replaces the page that won’t be used for the longest time.
* **LFU**: Replaces the least frequently used page.
* **Tracks page faults** and frame updates step by step.

**Example Table (Memory Frame Updates):**

| **Step** | **Page Request** | **FIFO Frames** | **LRU Frames** | **Optimal Frames** | **LFU Frames** |
| --- | --- | --- | --- | --- | --- |
| 1 | 7 | 7 | 7 | 7 | 7 |
| 2 | 0 | 7, 0 | 7, 0 | 7, 0 | 7, 0 |
| 3 | 1 | 7, 0, 1 | 7, 0, 1 | 7, 0, 1 | 7, 0, 1 |

**(Page Fault occurs when a page is not in memory)**

**Module 3: Visualization & Performance Analysis**

* **Memory Frame Display** (Step-by-step execution).
* **Page Fault Count Display** (Total faults per algorithm).
* **Graphical Comparison** (Bar charts for performance comparison).

**Example Output:**

diff

Copy code

Algorithm | Page Faults

------------------------

FIFO | 9

LRU | 7

Optimal | 5

LFU | 8

📊 **Graph Representation:**

markdown

████ FIFO

███ LRU

██ Optimal

███ LFU

* *With this breakdown, the simulator effectively models different page replacement strategies and provides a clear analysis of their efficiency.*

**4. Technology Used**

**Programming Language:** Python  
**Libraries:** Matplotlib, Pandas, Tkinter  
**Other Tools:** GitHub for version control

**5. Flow Diagram**

A simple **flowchart** representing the **page replacement workflow**:

pysql

CopyEdit

+----------------------+

| Start Simulation |

+----------------------+

|

+----------------------+

| Get User Input |

| (Reference String, |

| Frames, Algorithm)|

+----------------------+

|

+----------------------+

| Simulate Algorithm |

+----------------------+

|

+----------------------+

| Display Results |

| (Page Faults, Hit |

| Ratio, Graphs) |

+----------------------+

|

+----------------------+

| End |

+----------------------+

**6. Revision Tracking on GitHub**

**Repository Name:** Page-Replacement-Simulator  
**GitHub Link:** [Insert GitHub Repository URL]

**7. Conclusion and Future Scope**

**Conclusion**

* Implemented and analysed **FIFO, LRU, Optimal, and LFU**.
* Compared algorithms based on **page fault frequency**.
* Provided **graphical visualizations** for better understanding.

**Future Scope**

* **Add more algorithms** (e.g., Clock, Second-Chance).
* **Enhance GUI** (Tkinter/PyQt-based interactive visualization).
* **Develop a web-based tool** for real-time simulation.

**8. References**

1. "Operating System Concepts" – Silberschatz, Galvin, and Gagne.
2. "Modern Operating Systems" – Andrew S. Tanenbaum.
3. GitHub Docs for Version Control.

**Appendix**

1. **AI-Generated Project Breakdown**

***Phase 1: Problem Assignment***

**Step 1: AI-Guided Development**

**Problem Statement:**

Design and implement a **Page Replacement Algorithm Simulator** that compares different page replacement strategies (FIFO, LRU, Optimal, LFU). The simulator should allow users to enter a reference string and number of frames, visualize how pages are replaced, and display page fault statistics.

**1. Project Overview**

**Goals:**

* To create a **simulator** that demonstrates different **page replacement algorithms**.
* To analysis **page faults and hit ratios** across algorithms.
* To provide **visualization** of memory allocation for better understanding.

**Expected Outcomes:**

* A working **simulation tool** for page replacement algorithms.
* A **graphical analysis** comparing algorithm efficiencies.
* A **GitHub repository** with multiple revisions and documentation.

**Scope:**

* Simulating **FIFO, LRU, Optimal, and LFU**.
* **Step-by-step execution visualization**.
* Allowing user input for reference strings and frame numbers.
* Providing **comparative performance statistics**

**2. Module-Wise Breakdown**

**Module 1: Input & Processing**

* Takes user input (reference string, frames, algorithm choice).
* Implements **page replacement logic** for FIFO, LRU, Optimal, LFU.

**Module 2: Simulation & Visualization**

* **Console-based simulation** (for basic understanding).
* **Graphical representation** of page replacement (Matplotlib/tkinter ).
* **Table format** displaying step-by-step page replacements.

**Module 3: Analysis & Comparison**

* **Page fault and hit ratio calculation**.
* **Graph comparison** of algorithm efficiency.
* **Export results** (optional CSV file for further analysis).

**3. Functionalities**

**For Each Algorithm**

✅ Accept user inputs (reference string, number of frames).  
✅ Implement **FIFO, LRU, Optimal, and LFU** logic.  
✅ Display the memory state after each request.  
✅ Calculate **page faults and hit ratio**.  
✅ Provide a **graphical representation** of page replacement.

**4. Technology Recommendations**

**Programming Languages**

* **Python** (Recommended for fast prototyping & visualization).
* **C++** (For optimized memory handling).
* **Java** (For GUI-based implementations).

**Libraries & Tools**

* **Matplotlib** (Graph visualization)
* **Tkinter/PyQt** (GUI)
* **Pandas** (For result storage)
* **GitHub** (Version control & collaboration)

**5. Execution Plan**

**Step 1: Project Setup**

✅ Create a **GitHub repository**  
✅ Install necessary **libraries** (matplotlib, pandas)  
✅ Define **problem scope** and list algorithms

**Step 2: Implement Page Replacement Algorithms**

✅ Implement **FIFO** using a queue.  
✅ Implement **LRU** using **HashMap + Doubly Linked List**.  
✅ Implement **Optimal** by predicting future requests.  
✅ Implement **LFU** using **Min-Heap + Frequency Counter**.

**Step 3: User Input Handling**

✅ Ask users for **reference string** and **frame count**.  
✅ Implement a **menu-driven system** for algorithm selection.

**Step 4: Visualization**

✅ Print **step-by-step** page replacement in a table.  
✅ Use **Matplotlib/Tkinter** for visualization.

**Step 5: Performance Analysis**

✅ Compute **page faults & hit ratio**.  
✅ Compare algorithms using **graphs**.

**Phase 2: Revision Tracking on GitHub**

**GitHub Workflow**

1. **Create a GitHub Repository** (Page-Replacement-Simulator)
2. **Push initial files** (README, problem statement).
3. **Maintain at least 7 revisions**:
   * Rev 1: Initial algorithm implementations (FIFO, LRU)
   * Rev 2: Implement Optimal and LFU
   * Rev 3: Add user input handling
   * Rev 4: Add **console-based** simulation
   * Rev 5: Implement **Matplotlib/Tkinter** visualization
   * Rev 6: Optimize algorithms (reduce time complexity)
   * Rev 7: Final testing & README update
4. **Use branches** for feature updates and merge after testing.
5. **README** should include:
   * Project overview.
   * Installation & usage instructions.
   * Explanation of algorithms.

**B. Problem Statement**

Here’s the **Python implementation** for your **Page Replacement Algorithm Simulator** with FIFO, LRU, Optimal, and LFU. It includes **user input handling, visualization, and performance comparison**.

**🔹 Step 1: Install Required Libraries**

Before running the code, install Matplotlib:

bash

pip install matplotlib

**🔹 Step 2: Complete Code Implementation**

This Python program:

* Takes **user input** (reference string, frame count).
* Implements **FIFO, LRU, Optimal, and LFU**.
* **Visualizes** page replacements step-by-step.
* **Compares page faults** of different algorithms.

***Page Replacement Simulator***

import matplotlib.pyplot as plt

from collections import deque, defaultdict

# Function to simulate FIFO Page Replacement

def fifo\_page\_replacement(pages, frames):

memory = deque()

page\_faults = 0

history = []

for page in pages:

if page not in memory:

if len(memory) == frames:

memory.popleft()

memory.append(page)

page\_faults += 1

history.append(list(memory))

return page\_faults, history

# Function to simulate LRU Page Replacement

def lru\_page\_replacement(pages, frames):

memory = []

page\_faults = 0

history = []

for page in pages:

if page not in memory:

if len(memory) == frames:

memory.pop(0)

page\_faults += 1

else:

memory.remove(page) # Remove page and re-add it (mark as recently used)

memory.append(page)

history.append(list(memory))

return page\_faults, history

**Step 3: How to Use This Project**

1.Run the program.  
2️.Enter a **reference string** (e.g., 7 0 1 2 0 3 4 2 3 0 3 2).  
3️.Enter the **number of frames** (e.g., 3).  
4️.The program will:

* Simulate **FIFO, LRU, Optimal, and LFU**.
* Print **step-by-step memory states**.
* Display a **bar chart** comparing page faults.

**Step 4: Project Breakdown**

| **Module** | **Purpose** |
| --- | --- |
| **Input Handling** | Takes user input (reference string, frame count) |
| **Algorithm Implementation** | Implements FIFO, LRU, Optimal, LFU |
| **Visualization** | Displays memory state changes & bar chart |

**Step 5: Technologies Used**

**Python** for implementation  
**Matplotlib** for visualization  
**GitHub** for version tracking

**Step 6: Version Control with GitHub**

1️.**Create a GitHub repo** (e.g., page-replacement-simulator).  
2️.**Push changes regularly** with clear commit messages.  
3️.**Use branches** for adding features & testing.  
4️.**Maintain a README** with project details.

***pip install matplotlib***

**Step 1: Implementing Page Replacement Algorithms**

Below is the **Python implementation** for **FIFO, LRU, Optimal, and LFU** with **step-by-step visualization**.

**🔹 Complete Code**

python

CopyEdit

import matplotlib.pyplot as plt

from collections import deque, OrderedDict

# Function to simulate FIFO Page Replacement

def fifo\_page\_replacement(pages, frames):

memory = []

page\_faults = 0

history = [] # To track step-by-step memory state

for page in pages:

if page not in memory:

page\_faults += 1

if len(memory) < frames:

memory.append(page)

else:

memory.pop(0) # Remove oldest page

memory.append(page)

history.append(memory[:]) # Store memory state after each step

return page\_faults, history

# Function to simulate LRU Page Replacement

def lru\_page\_replacement(pages, frames):

memory = OrderedDict()

page\_faults = 0

history = []

for page in pages:

if page not in memory:

page\_faults += 1

if len(memory) >= frames:

memory.popitem(last=False) # Remove least recently used

else:

memory.move\_to\_end(page) # Update LRU position

memory[page] = True # Insert page

history.append(list(memory.keys())) # Store state

return page\_faults, history

# Function to simulate Optimal Page Replacement

def optimal\_page\_replacement(pages, frames):

memory = []

page\_faults = 0

history = []

for i, page in enumerate(pages):

if page not in memory:

page\_faults += 1

if len(memory) < frames:

memory.append(page)

else:

# Find page to be replaced (farthest used in future)

future\_use = {p: pages[i+1:].index(p) if p in pages[i+1:] else float('inf') for p in memory}

to\_remove = max(future\_use, key=future\_use.get)

memory[memory.index(to\_remove)] = page

history.append(memory[:]) # Store state

return page\_faults, history

# Function to simulate LFU Page Replacement

def lfu\_page\_replacement(pages, frames):

memory = {}

page\_faults = 0

history = []

for page in pages:

if page in memory:

memory[page] += 1

else:

page\_faults += 1

if len(memory) >= frames:

# Remove least frequently used page

to\_remove = min(memory, key=memory.get)

del memory[to\_remove]

memory[page] = 1 # Insert new page

history.append(list(memory.keys())) # Store state

return page\_faults, history

# Function to visualize memory allocation step by step

def visualize\_memory(history, algorithm\_name):

print(f"\n{algorithm\_name} Simulation:")

for step, state in enumerate(history):

print(f"Step {step+1}: {state}")

# Function to compare all algorithms

def compare\_algorithms(pages, frames):

algorithms = {

"FIFO": fifo\_page\_replacement,

"LRU": lru\_page\_replacement,

"Optimal": optimal\_page\_replacement,

"LFU": lfu\_page\_replacement

}

results = {}

for name, func in algorithms.items():

faults, history = func(pages, frames)

results[name] = faults

visualize\_memory(history, name)

# Plot bar chart for comparison

plt.figure(figsize=(8,5))

plt.bar(results.keys(), results.values(), color=['blue', 'green', 'red', 'purple'])

plt.xlabel("Page Replacement Algorithms")

plt.ylabel("Page Faults")

plt.title("Comparison of Page Replacement Algorithms")

plt.show()

# Main Function

if \_\_name\_\_ == "\_\_main\_\_":

pages = list(map(int, input("Enter reference string (space-separated): ").split()))

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

compare\_algorithms(pages, frames)

**Step 3:Explanation of Code**

**Key Functionalities**

1. **Implements**:
   * **FIFO** (First-In-First-Out)
   * **LRU** (Least Recently Used)
   * **Optimal** (Minimizing future faults)
   * **LFU** (Least Frequently Used)
2. **Tracks memory state** at each step.
3. **Prints step-by-step execution** for all algorithms.
4. **Visualizes page faults** with a bar chart.

**Step 4: Sample Run**

**Input:**

Enter reference string (space-separated): 7 0 1 2 0 3 4 2 3 0 3 2

Enter number of frames: 3

**Output (Console)**

FIFO Simulation:

Step 1: [7]

Step 2: [7, 0]

Step 3: [7, 0, 1]

Step 4: [0, 1, 2]

Step 5: [0, 1, 2]

...

Total Page Faults: 9

LRU Simulation:

Step 1: [7]

Step 2: [7, 0]

Step 3: [7, 0, 1]

Step 4: [0, 1, 2]

...

Total Page Faults: 8

Optimal Simulation:

Step 1: [7]

Step 2: [7, 0]

Step 3: [7, 0, 1]

Step 4: [0, 1, 2]

...

Total Page Faults: 6

LFU Simulation:

Step 1: [7]

Step 2: [7, 0]

Step 3: [7, 0, 1]

Step 4: [0, 1, 2]

...

Total Page Faults: 7

**Step 5: GitHub Workflow**

1️.**Create a new repository** (page-replacement-simulator).  
2️.**Push initial code**:

bash

git init

git add .

git commit -m "Initial commit - Implemented page replacement algorithms"

git branch -M main

git remote add origin <your-github-repo-url>

git push -u origin main

3️.**Track changes**:

bash

git add .

git commit -m "Added visualization & performance comparison"

git push

**Step 6: Future Scope**

Convert this into a **GUI-based tool** using **Tkinter/PyQt**.

Implement **more algorithms** (Clock, Second-Chance).  
 Create a **web-based simulator** using Flask/Django.

**Final Thoughts**

✅ **Efficient & modular** implementation.  
✅ **Clear step-by-step execution & visualization**.  
✅ **Easy to extend with additional features**.

**🔹 Conclusion**

The **Page Replacement Algorithm Simulator** successfully demonstrates and compares different page replacement strategies used in operating systems. By implementing **FIFO, LRU, Optimal, and LFU**, the simulator helps users understand how page faults occur and how different algorithms manage memory efficiently.

The **visualization module** enhances learning by displaying **step-by-step memory updates** and **graphical comparisons**, making it easier to analyze algorithm performance.

**✅ Key Takeaways:**

✔ **Practical Understanding**: Users can see how different page replacement algorithms work.  
✔ **Performance Comparison**: The simulator provides insights into which algorithm performs best in different scenarios.  
✔ **Educational Tool**: Useful for students and researchers studying OS memory management.

**Future Scope:**

* Add more algorithms like **MFU (Most Frequently Used)** or **Random Page Replacement**.
* Implement **real-world case studies** to test performance under different workloads.
* Enhance the GUI for **better user interaction**.
* Extend to **multi-threaded memory management simulations**.