SIMULATION OF MEMORY MANAGEMENT FUNCTIONS

A Course End Project Report in Operating Systems Laboratory

(Course Code - A8512)

Submitted in the Partial Fulfilment of the

Requirements

for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

Submitted

By

B. Keerthana	**23881A67D4**
G. Greeshma	**23881A67E4**
G. Yogitha	**23881A67E5**
Jinna Neha	**23881A67F3**

Under the Esteemed Guidance of

Dr. K. Nikhila
Assistant Professor

VARDHAMAN COLLEGE OF ENGINEERING, HYDERABAD

(AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

November, 2024

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Course End Project titled **"Simulation of Memory Management Functions"** is carried out by **"B. Keerthana" , "G. Greeshma" , "G. Yogitha", and "Jinna Neha" with** Roll Numbers **"23881A67D4", "23881A67E4" , "23881A67E5" , and "23881A67F3"** towards **A8512 – Operating Systems Laboratory** course in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering (Data Science)** during the Academic year 2024-25.

Signature of the Course Faculty

Dr. Nikhila
Assistant Professor, CSD

Signature of the HoD

Dr. G. Sreenivasulu

HOD, CSE

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We wish to express our deep sense of gratitude to **Dr. Nikhila**, **Assistant Professor**, Department of Computer Science and Engineering, Vardhaman College of Engineering, for her able guidance and useful suggestions, which helped us in completing the design part of potential project in time.

We particularly thankful to **Dr. G. Sreenivasulu**, Associate Professor & Head, Department of Computer Science and Engineering for his guidance, intense support and encouragement, which helped us to mould our project into a successful one.

We show gratitude to our honorable Principal **Dr. J.V.R. Ravindra**, for having provided all the facilities and support.

We avail this opportunity to express our deep sense of gratitude and heartfelt thanks to **Dr.Teegala Vijender Reddy**, Chairman and **Sri Teegala Upender Reddy**, Secretary of VCE, for providing a congenial atmosphere to complete this project successfully.

We also thank all the staff members of Computer Science and Engineering for their valuable support and generous advice. Finally, thanks to all our friends and family members for their continuous support and enthusiastic help.

B. Keerthana– 23881A67D4
G. Greeshma– 23881A67E4
G. Yogitha– 23881A67E5
Jinna Neha– 23881A67F3

INDEX

1. Introduction

In modern computing systems, efficient memory management is essential to ensure optimal system performance. Virtual memory is a critical concept that enables systems to use physical memory more effectively by abstracting it into a larger, logical memory space. This project focuses on simulating a virtual memory system with paging, a widely used memory management technique, and analyzing the performance of page replacement algorithms.

Paging divides the memory into fixed-sized blocks, called pages (in virtual memory) and frames (in physical memory). This method simplifies memory allocation and eliminates external fragmentation. However, when the system runs out of available frames, a page replacement algorithm determines which page to evict to accommodate new ones. The choice of the algorithm significantly impacts the system's performance, especially under high memory load.

This project implements two widely used page replacement algorithms:

1. Least Recently Used (LRU): Prioritizes keeping recently accessed pages in memory by replacing the least recently used ones.
2. First-In-First-Out (FIFO): Replaces the page that was loaded into memory the earliest, regardless of recent usage.

To simulate these mechanisms, a Graphical User Interface (GUI) has been developed, allowing users to input parameters like reference strings and the number of frames. The GUI also provides visual feedback on page faults, memory states, and algorithm performance, offering insights into the efficiency of each method.

Through this project, we aim to demonstrate:

- The fundamentals of virtual memory and paging.
- The practical implementation of demand paging and page replacement strategies.
- The analysis of algorithm efficiency under different memory access patterns.

This simulation provides a hands-on understanding of how memory management policies influence overall system performance, making it a valuable educational tool for computer science enthusiasts.

2. Objective of the Project

The primary objective of this project is to simulate a virtual memory system with **paging** and analyze the performance of various page replacement algorithms. By implementing these concepts in a graphical user interface (GUI), the project aims to provide an interactive and educational platform for understanding the critical aspects of memory management in modern computer systems. Specifically, the objectives are:

1. **To Demonstrate Virtual Memory Concepts:**
   Simulate how virtual memory operates using paging, including the management of page tables and frames.

2. **To Implement Page Replacement Algorithms**:
   Develop and compare the performance of two widely used page replacement strategies:

   - Least Recently Used (LRU)
   - First-In-First-Out (FIFO)

3. **To Analyze System Performance**:
   Measure and evaluate the impact of page replacement algorithms on system performance, particularly in terms of:

   - Number of page faults.
   - Memory utilization.
   - Efficiency under different memory access patterns.

4. **To Provide an Interactive Learning Tool**:
   Design a user-friendly GUI that:

   - Allows users to input custom reference strings and frame counts.
   - Visualizes the memory states and page fault occurrences in real time.

5. **To Encourage Practical Understanding**:
   Bridge the gap between theoretical knowledge and practical application of memory management concepts through hands-on simulation and analysis.

3. Problem statement

Efficient memory management is a cornerstone of modern computing systems, as it directly impacts system performance and user experience. One critical challenge in memory management is the replacement of memory pages when physical memory is fully occupied. The problem becomes complex in multitasking environments, where multiple processes compete for limited memory resources, leading to page faults and performance degradation.

This project aims to address the following key issues:

1. **Understanding Virtual Memory Operations**:
   How can virtual memory be effectively simulated to represent the interaction between virtual pages and physical frames?

2. **Managing Page Faults**:
   What strategies can be implemented to minimize page faults when pages need to be loaded into physical memory?

3. **Evaluating Page Replacement Algorithms**:
   How do different page replacement algorithms, such as **Least Recently Used (LRU)** and **First-In-First-Out (FIFO)**, perform under various memory access patterns? Which algorithm is more efficient, and under what circumstances?

4. **Interactive Visualization of Memory Management**:
   How can a graphical interface be designed to provide real-time insights into the working of paging and page replacement, making the concepts accessible and intuitive for learners?

5. **Impact Analysis on System Performance**:
   What are the measurable effects of the choice of page replacement algorithm on critical performance metrics, such as the number of page faults and memory usage efficiency?

By addressing these questions, the project aims to develop a simulation tool that demonstrates virtual memory operations, implements page replacement strategies, and provides a comparative analysis of their impact on system performance.

4. Software and hardware requirements

Software Requirements

1. Operating System:
    - Windows 10 or Windows 11.
2. Programming Language:
    - Python (Version 3.8 or higher).
3. Libraries/Frameworks:
    - Tkinter: Pre-installed with Python, used for the GUI.
    - Matplotlib: For performance visualization.
    - Pip: For managing and installing Python libraries (ensure it is installed).
4. Development Environment:
    - Recommended: VS Code (with Python extension) or PyCharm.

Hardware Requirements

1. Processor:
    - Minimum: Dual-core processor (e.g., Intel Core i3 or equivalent).
2. RAM:
    - Minimum: 4 GB (to run simulations without significant lag).
3. Storage:
    - At least 500 MB of free space for Python and related libraries.
4. Display:
    - Minimum: 1366 x 768 resolution for proper visualization of the GUI.

5. Project Description

This project focuses on simulating a virtual memory management system that uses paging as its memory allocation technique. Paging allows dividing the memory into fixed-size blocks (pages and frames) to manage memory efficiently and reduce fragmentation. When a requested page is not in memory, a page replacement algorithm is used to decide which page to evict, ensuring optimal system performance.

Key Features of the Project

1. **Virtual Memory Simulation**:
    - Simulates the interaction between virtual memory (pages) and physical memory (frames).
    - Implements demand paging, where pages are loaded into memory only when required.
2. **Page Replacement Algorithms**:
    - **Least Recently Used (LRU)**: Replaces the page that has not been accessed for the longest time.
    - **First-In-First-Out (FIFO)**: Replaces the oldest page in memory.
3. **Performance Analysis**:
    - Tracks key metrics like the number of page faults and the hit/miss ratio.
    - Compares the efficiency of the LRU and FIFO algorithms under various memory access patterns.
4. **Graphical User Interface (GUI)**:
    - Allows users to interact with the simulation by:
        - Specifying the number of frames and reference strings.
        - Visualizing page table updates and memory states in real-time.
    - Displays performance metrics and analysis using graphical charts.

Workflow of the Project

1. The user inputs:
    - The **reference string** (sequence of page requests).
    - The **number of frames** available in physical memory.
    - The **page replacement algorithm** (LRU or FIFO).
2. The system:
    - Simulates the paging process.
    - Handles page faults and applies the selected page replacement algorithm.
    - Updates the page table and memory states dynamically.
3. Outputs:
    - Visual representation of memory states and page faults.
    - Performance metrics to compare algorithm efficiency.

6. Steps/Algorithm/Procedure

Steps for implementation:

- Understand the Concepts
- Set Up the Development Environment
- Design the System Architecture
- Implement the Core Logic
- Create the GUI
- Test the Simulation
- Add Visualization
- Document the Results
- Finalize and Optimize

Algorithm:

General Simulation Algorithm

1. Input:
    - Reference string (sequence of page requests).
    - Number of available frames.
    - Page replacement algorithm (FIFO or LRU).
2. Initialize:
    - An empty list or structure to represent frames in physical memory.
    - A counter for page faults, initially set to 0.
3. Process Reference String:
    - For each page in the reference string:
        - If the page is already in memory:
            - Mark it as a hit (no page fault).
        - Else:

- Increment the page fault counter.
- If the memory is full:
    - Use the selected page replacement algorithm to determine the page to evict.
- Load the new page into memory.

4. Output:
    - The total number of page faults.
    - Final state of the memory frames.
    - (Optional) Visualization of page states and faults.

FIFO Algorithm

1. Initialize:
    - A queue to track the order of pages loaded into memory.
2. For each page in the reference string:
    - If the page is already in the queue:
        - Do nothing (hit).
    - Else:
        - Increment the page fault counter.
        - If the queue is full:
            - Remove the page at the front of the queue (FIFO principle).
        - Add the new page to the back of the queue.
3. Return:
    - The total number of page faults.

LRU Algorithm

1. Initialize:
    - A list or dictionary to track the last usage time of each page.
2. For each page in the reference string:
    - If the page is already in memory:
        - Update its last usage time to the current step (hit).
    - Else:
        - Increment the page fault counter.
        - If memory is full:
            - Identify the page with the oldest last usage time.
            - Evict that page from memory.
        - Load the new page and set its usage time to the current step.
3. Return:
    - The total number of page faults.

Procedure:

1. Setup and Initialization

1. Input Parameters:
    - Reference String: A sequence of page requests (e.g., [7, 0, 1, 2, 0, 3, 0, 4, 2]).
    - Number of Frames: The total number of available frames in physical memory (e.g., 3 frames).
    - Page Replacement Algorithm: Choose between FIFO or LRU.
2. Initialize Data Structures:

- Memory (Frames): A list or queue to represent the physical memory (e.g., memory = []).
- Page Table: A mapping of pages to frames (not strictly necessary in FIFO but can be tracked for debugging).
- Page Fault Counter: A variable to count the number of page faults (e.g., page_faults = 0).
- Usage Tracking (For LRU): A dictionary or list to track the usage time of pages in LRU (e.g., usage_time = {}).

2. Simulate the Paging Process

Step 1: Process the Reference String

- For each page in the reference string:
  - Check if the Page is Already in Memory:
    - If yes, it's a hit. Move on to the next page.
    - If no, it's a page fault. Increment the page_faults counter.

Step 2: Handle Page Faults

- When a page fault occurs:
  - FIFO Algorithm:
    - If there is space in memory (i.e., the number of pages is less than the available frames), add the new page.
    - If memory is full, remove the oldest page from memory (the page at the front of the queue) and add the new page at the end of the queue.
  - LRU Algorithm:
    - If there is space in memory, add the new page.
    - If memory is full, identify the least recently used page (the one with the oldest last usage time) and replace it with the new page.

        - Update the usage time for the newly added page.

Step 3: Track and Update Memory State

- For FIFO: Use a queue to maintain the order of page arrivals.
- For LRU: Use a dictionary to track the last usage time of each page (using the current iteration as the timestamp).

Step 4: Continue to the Next Page in the Reference String

3. Output Results

1. Page Fault Count: Display the total number of page faults after processing the reference string.
2. Final Memory State: Display the contents of the memory after processing all page requests.
3. Performance Metrics: Optional - Show graphs or visualizations comparing the number of page faults for FIFO vs. LRU under different scenarios.

4. Implementation Flowchart

1. Start: Initialize variables and get inputs (reference string, number of frames, and algorithm type).
2. For each page in the reference string:
    - Is the page in memory?
        - Yes → Skip to next page.
        - No → Handle page fault.
3. Handle page fault:
    - FIFO: Evict the oldest page if memory is full, add the new page.
    - LRU: Evict the least recently used page, add the new page.
4. Update memory state and page table.
5. Repeat for all pages in the reference string.
6. End: Output results (number of page faults, final memory state, performance analysis).

---

5. Example Walkthrough

Given the reference string: [7, 0, 1, 2, 0, 3, 0, 4, 2] and 3 frames.

FIFO Simulation:

- Page Faults: Initialize memory as empty.
    1. Page 7: Page fault. Add 7 → [7].
    2. Page 0: Page fault. Add 0 → [7, 0].
    3. Page 1: Page fault. Add 1 → [7, 0, 1].
    4. Page 2: Page fault. Replace 7 → [0, 1, 2].
    5. Page 0: No page fault (already in memory).
    6. Page 3: Page fault. Replace 1 → [0, 2, 3].
    7. Page 0: No page fault (already in memory).
    8. Page 4: Page fault. Replace 2 → [0, 3, 4].
    9. Page 2: Page fault. Replace 3 → [0, 4, 2].
- Total Page Faults: 9

LRU Simulation:

- Page Faults: Initialize memory as empty.
    1. Page 7: Page fault. Add 7 → [7].
    2. Page 0: Page fault. Add 0 → [7, 0].
    3. Page 1: Page fault. Add 1 → [7, 0, 1].
    4. Page 2: Page fault. Replace 7 → [0, 1, 2].
    5. Page 0: No page fault (already in memory).
    6. Page 3: Page fault. Replace 1 → [0, 2, 3].

7. Page 0: No page fault (already in memory).
8. Page 4: Page fault. Replace 2 → [0, 3, 4].
9. Page 2: Page fault. Replace 3 → [0, 4, 2].

- Total Page Faults: 9

7. Code

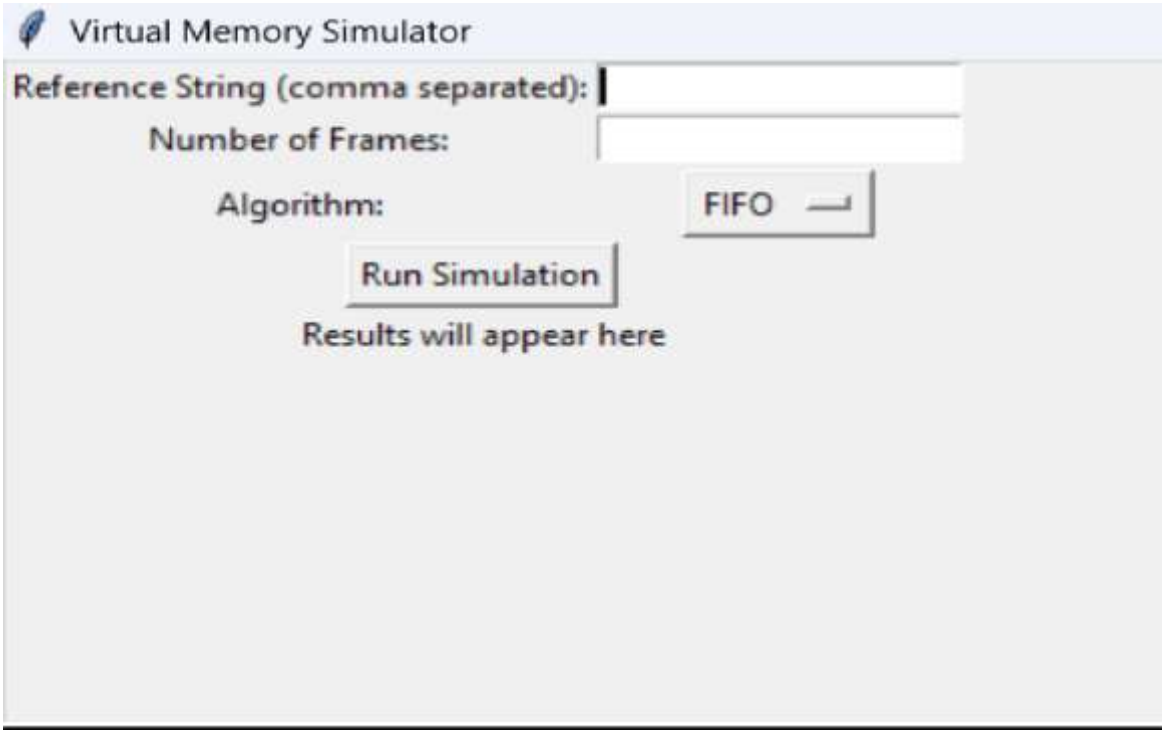
```
import tkinter as tk
import matplotlib.pyplot as plt
from collections import deque


# Function to simulate FIFO page replacement
def fifo_page_replacement(reference_string, num_frames):
    memory = []
    page_faults = 0
    for page in reference_string:
        if page not in memory:
            page_faults += 1
            if len(memory) == num_frames:
                memory.pop(0)  # Remove the oldest page
            memory.append(page)
    return page_faults, memory


# Function to simulate LRU page replacement
def lru_page_replacement(reference_string, num_frames):
    memory = []
    usage_time = {}  # Dictionary to track last usage time of pages
    page_faults = 0
```

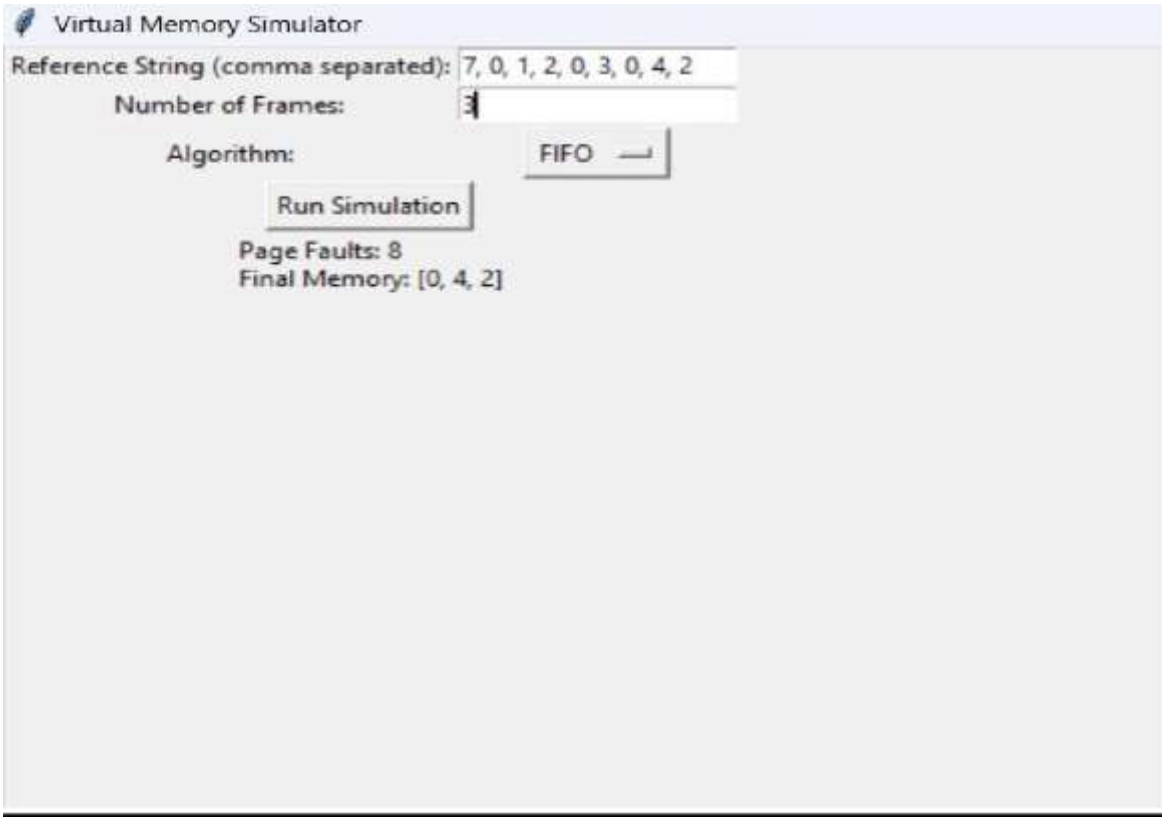
```
    for i, page in enumerate(reference_string):
        if page not in memory:
            page_faults += 1
            if len(memory) == num_frames:
                # Find the least recently used page
                lru_page = min(usage_time, key=usage_time.get)
                memory.remove(lru_page)
                del usage_time[lru_page]
            memory.append(page)
        usage_time[page] = i  # Update the last usage time for LRU
    return page_faults, memory


# GUI to input parameters and show results
def run_simulation():
    # Get inputs from the user
    reference_string = list(map(int, reference_entry.get().split(',')))
    num_frames = int(frame_entry.get())
    algorithm = algo_var.get()
    if algorithm == "FIFO":
        page_faults, final_memory = fifo_page_replacement(reference_string, num_frames)
    elif algorithm == "LRU":
        page_faults, final_memory = lru_page_replacement(reference_string, num_frames)
```

```
    # Display results in the GUI
    result_label.config(text=f"Page Faults: {page_faults}\nFinal Memory: {final_memory}")

    # Display a bar chart for page faults
    plt.bar(['FIFO' if algorithm == 'FIFO' else 'LRU'], [page_faults])
    plt.ylabel('Page Faults')
    plt.title(f'{algorithm} Algorithm Performance')
    plt.show()

# Set up the GUI
root = tk.Tk()
root.title("Virtual Memory Simulator")

# Create input fields
tk.Label(root, text="Reference String (comma separated):").grid(row=0, column=0)
reference_entry = tk.Entry(root)
reference_entry.grid(row=0, column=1)

tk.Label(root, text="Number of Frames:").grid(row=1, column=0)
frame_entry = tk.Entry(root)
frame_entry.grid(row=1, column=1)

tk.Label(root, text="Algorithm:").grid(row=2, column=0)
```

```
algo_var = tk.StringVar(value="FIFO")

tk.OptionMenu(root, algo_var, "FIFO", "LRU").grid(row=2, column=1)



# Run button to trigger the simulation

tk.Button(root, text="Run Simulation", command=run_simulation).grid(row=3, column=0,
columnspan=2)



# Label to display results

result_label = tk.Label(root, text="Results will appear here", justify=tk.LEFT)

result_label.grid(row=4, column=0, columnspan=2)



# Run the GUI

root.mainloop()
```

8. Result(s)


Virtual Memory Simulator
Reference String (comma separated):
Number of Frames:
Algorithm:
FIFO
Run Simulation
Results will appear here


Virtual Memory Simulator
Reference String (comma separated): 7, 0, 1, 2, 0, 3, 0, 4, 2
Number of Frames: 3
Algorithm:
FIFO
Run Simulation
Page Faults: 8
Final Memory: [0, 4, 2]

9. Conclusion and Future work

Conclusion

The Virtual Memory System Simulation with Paging project demonstrates the practical application of paging and page replacement algorithms (FIFO and LRU) in operating systems. By simulating how memory management works in real-time with a reference string and a given number of memory frames, this project provides valuable insights into the efficiency and behavior of memory management techniques.

Key takeaways from the project:

- **Paging** allows large programs to run on systems with limited physical memory by swapping pages in and out of physical memory.
- **Page Replacement Algorithms** help manage memory efficiently when the system faces more page requests than can fit in physical memory.
    - **FIFO (First-In-First-Out)** is simple and easy to implement but may not always result in optimal memory utilization.
    - **LRU (Least Recently Used)** is more sophisticated and aims to minimize page faults by keeping the most recently used pages in memory. It tends to perform better in many practical scenarios.
- The use of a **Graphical User Interface (GUI)** made the simulation user-friendly and interactive, allowing for easy testing and visualization of results.
- The **performance metrics** such as the number of page faults provide valuable data on how different algorithms behave with varying reference strings and memory frame sizes.

In conclusion, this project serves as an educational tool for understanding the importance of memory management and page replacement strategies in real-world operating systems. It highlights the trade-offs involved in selecting a page replacement algorithm based on system constraints and workload characteristics.

Future work

Future enhancements can include:

- **More Page Replacement Algorithms**: Implementing other algorithms like **Optimal**, **Clock**, or **Second Chance**.
- **Visualization of Memory State**: Adding a dynamic graphical display to show how memory changes with each page request.
- **Extended Performance Analysis**: Including metrics such as page hit rate, memory utilization, and time complexity for different algorithms.

10. References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (9th ed.). Wiley.

    - This textbook is one of the most widely used in computer science for understanding operating system concepts. It provides a thorough explanation of memory management, including paging and page replacement algorithms like FIFO and LRU.

2. Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Pearson.

    - A renowned book for understanding modern operating systems. It covers in-depth details of virtual memory, paging, page replacement algorithms, and performance evaluation, which are key concepts for your project.

3. Stallings, W. (2015). *Operating Systems: Internals and Design Principles* (8th ed.). Pearson.

    - This book provides an in-depth explanation of the internals of operating systems, including detailed coverage of memory management and paging algorithms, making it essential for understanding how virtual memory systems work.

4. Vogt, T. (2018). *Python GUI Programming with Tkinter: A Comprehensive Guide to Building Graphical User Interfaces with Python*. Packt Publishing.

    - Since your project includes a GUI, this book will help you learn how to build user interfaces with Tkinter in Python, guiding you in integrating the page replacement algorithms with a graphical display.

5. Matplotlib Documentation. (2021). *Matplotlib 3.4.3 documentation*. Retrieved from https://matplotlib.org/stable/contents.html

    - Matplotlib will be used for data visualization in your project (bar charts for page faults). The official documentation is a great resource for learning how to use Matplotlib for plotting and understanding its capabilities.