Documentation

Project Title: Page Replacement Algorithm Simulator

1. Abstract

This project simulates the memory management unit (MMU) of an Operating System, specifically focusing on **demand paging**. The software implements three distinct page replacement algorithms (FIFO, LRU, and Optimal) to manage page faults when the allocated physical memory frames are full. The simulator accepts a user-defined reference string and frame count, visually displaying the state of memory frames at every step and calculating the total Page Faults and Hits for performance comparison.

2. Introduction

In modern operating systems, physical memory (RAM) is limited. When a program tries to access a page that is not currently in physical memory, a **Page Fault** occurs. The OS must then swap a page from the disk into a physical frame. If all frames are occupied, a **Page Replacement Algorithm** determines which existing page must be evicted to make room for the new one.

Objectives:

- To simulate the mechanics of page swapping.
- To compare the efficiency (fault rate) of different algorithms.
- To visualize the frame content changes dynamically.

3. System Requirements

- **Programming Language:** C++ (Standard Template Library)
- **Compiler:** GCC (g++) or any standard C++ compiler
- **Operating System:** Linux using Ubuntu
- **Libraries Used:** <vector>, <iostream>, <algorithm>

4. Algorithm Implementation Details

4.1 First-In-First-Out (FIFO)

- **Theory:** The simplest algorithm. It associates each page with the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.
- **Implementation Logic:**
    - We utilize a circular buffer approach rather than a physical queue to maintain efficiency.
    - A pointer index tracks the oldest page. When a page fault occurs and memory is full, the page at frames[index] is replaced.
    - The pointer is updated using modular arithmetic: index = (index + 1) % framesCount.

4.2 Least Recently Used (LRU)

- **Theory:** LRU replaces the page that has not been used for the longest period of time. It relies on the principle of **temporal locality**; pages used recently will likely be used again soon.
- **Implementation Logic:**
    - When a page fault occurs, the simulator scans the *history* of the reference string (pages[0] to pages[current-1]).
    - It identifies which of the currently loaded frames was seen furthest back in the past.
    - That specific frame index is targeted for replacement.

4.3 Optimal Page Replacement

- **Theory:** This algorithm replaces the page that will not be used for the longest period of time in the future. It guarantees the lowest possible page-fault rate (proven by Belady).
- **Implementation Logic:**
    - Unlike LRU which looks backward, the Optimal algorithm looks *forward* into the reference string (pages[current+1] to pages[end]).
    - If a page in the frame is never used again (nextUse == -1), it is immediately chosen for replacement.
    - Otherwise, the page whose next occurrence is furthest in the future is replaced.

5. Code Structure Analysis

The application is modularized into specific functions for readability and maintainability.

Component	Description
main()	The driver function. Handles user input (Frames, Reference String) and controls the menu loop.
printFrames()	A utility function that renders the current state of physical memory (e.g., [7] [0] [1]) after every step.
calculateFIFO()	Implements the FIFO logic. Uses a circular index to manage replacement order.
calculateLRU()	Implements LRU logic. Uses nested loops to scan backward and find the least recently accessed page.
calculateOptimal()	Implements Optimal logic. Uses nested loops to scan forward to find the page used furthest in the future.

6. Use Manual & Execution Guide

6.1 Compilation

To compile the project in a Linux/Terminal environment:

Command= (“g++ page_replacement.cpp -o simulator”)

6.2 Running the Application

Execute the binary:

Command= (“./simulator”)

Sample Usage Scenario

Step 1: Input Configuration

Enter number of Frames: 3

Enter number of Pages: 10

Enter the Page Reference String: 7 0 1 2 0 3 0 4 2 3

Step 2: Algorithm Selection The menu will appear. Select 1 for FIFO.

Step 3: Output Interpretation The system will print the status of every page request:

- **HIT:** The page was already in memory. No change in frames.
- **MISS:** The page was not found. The frames are updated.

Example Output Snippet (FIFO):

Page 7: MISS -> [7] [ ] [ ]

Page 0: MISS -> [7] [0] [ ]

Page 1: MISS -> [7] [0] [1]

Page 2: MISS -> [2] [0] [1]

...

Total Page Faults = Miss Count

7. Conclusion and Future Scope

7.1 Future Scope

1. GUI Integration: Developing a graphical interface to visualize the frame filling process dynamically.
2. Additional Algorithms: Implementing Second-Chance (Clock) or LFU (Least Frequently Used) algorithms.

7.2 Conclusion

This project successfully demonstrates the trade-offs between complexity and efficiency in memory management.

- **FIFO Algorithms** : It is easy to implement but suffers from Belady’s Anomaly.

- **LRU Algorithms : It** is efficient and approximates the Optimal algorithm well but requires significant hardware/software overhead to track history.
- **Optimal Algorithms :It** provides the theoretical best performance but is unimplementable in real-time systems as it requires knowledge of future events; it serves as a benchmark for measuring other algorithms.