CSE 406 – Lab Report 9: FIFO Page Replacement Algorithm

Submitted By:

Sharif Md. Yousuf
ID: 22101128
Section: C-2
4th Year, 1st Semester
Spring 2025

Submitted To:

Atia Rahman Orthi
Lecturer
Department of Computer Science & Engineering
University of Asia Pacific

Date of Submission:
03 October, 2025 (Friday)

1 Problem Statement

In this lab, I was tasked with implementing a page replacement simulator using the FIFO (First In First Out) algorithm. The challenge was to create a program that, given a fixed number of frames and a sequence of page references, would simulate the process of loading pages into memory and replacing them when necessary. My program needed to track page hits and page faults, display the frame status at each step, and calculate performance metrics including hit ratio and fault ratio.

Input

For my implementation, the program accepts:

- Frame size (number of available frames in memory)
- Number of page references
- Sequence of page references

Here's an example of what I worked with:

```
Frame size: 3
Number of pages: 15
Page sequence: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2
```

Output

My program produces:

- A detailed table showing page reference, frame status, and hit/miss for each step
- Total page requests
- Total page hits and page faults
- Hit ratio and fault ratio as percentages

2 Objective

Through this lab, I aimed to achieve several learning goals:

- Gain a deep understanding of the FIFO page replacement algorithm
- Successfully implement FIFO to manage page frames using a queue data structure
- Learn how to track and visualize the state of memory frames at each page reference
- Calculate and analyze performance metrics including hit ratio and fault ratio
- Understand the concept of page faults and their impact on system performance
- Analyze the strengths and weaknesses of the FIFO approach to page replacement

3 Source Code Screenshot

FIFO_Page_Replacement_Algorithm.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

int main() {
    int frame_size, num_pages;
    cin >> frame_size >> num_pages;

    vector<int> pages(num_pages);
    for (int i = 0; i < num_pages; i++) {
        cin >> pages[i];
    }

    vector<int> frames(frame_size, -1);
    queue<int> fifoQueue;
    int pageFaults = 0;
    int pageHits = 0;

    cout << "\nPage Reference | Frames Status | Hit/Miss\n";
    cout << "-----------------------------------------\n";
    for (int i = 0; i < num_pages; i++) {
        int currentPage = pages[i];
        bool found = false;

        for (int j = 0; j < frame_size; j++) {
            if (frames[j] == currentPage) {
                found = true;
                pageHits++;
                break;
            }
        }

        cout << "      " << currentPage << "        | ";

        if (!found) {
            pageFaults++;

            if (fifoQueue.size() < frame_size) {
                for (int j = 0; j < frame_size; j++) {
                    if (frames[j] == -1) {
                        frames[j] = currentPage;
                        fifoQueue.push(currentPage);
                        break;
                    }
                }
```

SharifdotG

Figure 1: FIFO Page Replacement Algorithm Source Code (Part 1)

FIFO_Page_Replacement_Algorithm.cpp

```cpp
            } else {
                int pageToReplace = fifoQueue.front();
                fifoQueue.pop();

                for (int j = 0; j < frame_size; j++) {
                    if (frames[j] == pageToReplace) {
                        frames[j] = currentPage;
                        fifoQueue.push(currentPage);
                        break;
                    }
                }
            }

            for (int j = 0; j < frame_size; j++) {
                if (frames[j] == -1) {
                    cout << "[ ] ";
                } else {
                    cout << "[" << frames[j] << "] ";
                }
            }
            cout << "| MISS";
        } else {
            for (int j = 0; j < frame_size; j++) {
                if (frames[j] == -1) {
                    cout << "[ ] ";
                } else {
                    cout << "[" << frames[j] << "] ";
                }
            }
            cout << "| HIT";
        }
        cout << endl;
    }

    cout << "\n====== RESULTS ======\n";
    cout << "Total Page Requests: " << num_pages << endl;
    cout << "Total Page Hits: " << pageHits << endl;
    cout << "Total Page Faults: " << pageFaults << endl;
    cout << "Hit Ratio: " << (double)pageHits / num_pages * 100 << "%" << endl;
    cout << "Fault Ratio: " << (double)pageFaults / num_pages * 100 << "%" << endl;

    return 0;
}
```

SharifdotG

Figure 2: FIFO Page Replacement Algorithm Source Code (Part 2)

4 Output Screenshot

Figure 3: FIFO Page Replacement Program Output

5 Discussion

Working with the FIFO page replacement algorithm has been an enlightening experience that introduced me to the fundamental concepts of memory management in operating systems. I discovered that FIFO represents one of the simplest approaches to page replacement, following the intuitive principle that the oldest page in memory should be replaced first. Here's what I learned about its key characteristics:

- **Simplicity:** I found that FIFO's greatest strength is its straightforward implementation. Using a queue data structure, the algorithm naturally maintains the order in which pages entered memory, making replacement decisions trivial.
- **Queue-Based Management:** What I appreciated was how FIFO leverages the FIFO queue property perfectly - the page at the front of the queue is always the oldest and thus the first candidate for replacement.
- **Fair Replacement Policy:** I noticed that FIFO treats all pages equally, giving each page an equal opportunity to remain in memory based solely on arrival time, without considering usage patterns.
- **Performance Limitations:** However, I observed a significant drawback - FIFO doesn't consider page usage frequency or recency. A frequently used page that arrived early might be replaced, causing unnecessary page faults.
- **Belady's Anomaly:** Through my research, I learned that FIFO suffers from Belady's Anomaly, where increasing the number of frames can sometimes lead to more page faults rather than fewer.

Performance Analysis:

- **Page Fault Rate:** I observed that FIFO's performance heavily depends on the page reference pattern. Sequential or repeated patterns can lead to higher fault rates if frequently used pages are replaced.
- **Predictability:** The algorithm provides predictable behavior, making it easy to analyze and debug, which is valuable for understanding memory management concepts.
- **Real-world Limitations:** I learned that while FIFO is rarely used alone in modern systems, it serves as a foundation for understanding more sophisticated algorithms like LRU and optimal page replacement.

Through this implementation, I gained insights into the trade-offs between simplicity and performance in algorithm design, understanding that FIFO's ease of implementation comes at the cost of optimal performance.

6 Conclusion

Completing this FIFO page replacement implementation has been a valuable educational experience that provided me with foundational knowledge of memory management in operating systems. Through this project, I've gained a clear understanding of how page replacement algorithms work and their critical role in virtual memory systems.

While I observed that FIFO may not provide optimal performance due to its ignorance of page usage patterns, I've learned that it serves as an important baseline for understanding more complex algorithms. The simplicity of FIFO makes it an excellent starting point for studying page replacement strategies, and its predictable behavior helps in grasping the fundamental concepts of page faults and memory management.

This lab has been invaluable in showing me how the design of memory management algorithms involves balancing simplicity, performance, and practical implementation considerations. Understanding FIFO's strengths and limitations will be essential as I explore more advanced page replacement algorithms like LRU, LFU, and optimal replacement in future studies. The hands-on experience with tracking hits, faults, and calculating performance metrics has prepared me well for analyzing and comparing different memory management strategies.