Faculty of Computing

Semester 2 2024/2025

SECR2043 Operating Systems

Section 3

Virtual Memory Management – Replacement Algorithm: FIFO, LRU

Lecturer: Dr. Danlami Gabi

Group Salad

Student Name	Matric No.
GOE JIE YING	A23CS0224
LAM YOKE YU	A23CS0233
TAN YI YA	A23CS0187
TEH RU QIAN	A23CS0191

Table of Content

1.0 Abstract

Efficient virtual memory management is crucial for maintaining optimal system performance in modern operating systems. A key challenge arises when physical memory is full and the system must decide which pages to replace, which will be a decision made by page replacement algorithms. This project investigates and compares two widely used page replacement strategies: First-In, First-Out (FIFO) and Least Recently Used (LRU).

To illustrate this concept, a real-world analogy is presented: a chef working in a busy restaurant must manage a limited prep table space while needing various seasonings from a larger collection on a shelf. Just as the chef must decide which seasoning bottles to replace to keep cooking efficiently, an operating system must select pages to replace in memory to minimize page faults and maintain performance.

In this project, a simulation program was developed in C++ to model both FIFO and LRU algorithms using a fixed string representing the sequence of seasoning bottles used by the chef. The simulation captures the state of seasoning bottles on the table throughout the stated sequence, and calculates the number of times the chef had to take or replace a seasoning bottle from the shelf. The results demonstrate that by using LRU algorithm to replace the seasoning bottles, the number of time accessing the shelf is less than using FIFO algorithm, and was proved to be a more effective way for the chef to swap out his seasonings. Thus, the need of the chef is satisfied.

This project reinforces the importance of understanding and choosing appropriate page replacement strategies and provides a practical, relatable demonstration to support learning about virtual memory management.

2.0 Introduction

Virtual memory, also known as logical memory, is a foundational concept in Operating Systems. It promotes efficient multiprogramming by allowing the operation of processes in a space larger than the physical memory. This is achieved through demand paging, a method of loading data pages from secondary storage into memory only when they are needed. The ability of executing partially-loaded programs frees the constraints of physical memory and prevents errors and unusual routines caused by the large data structures. At the same time, more programs are able to run at the same time because each of them now takes less memory to execute, and less I/O are needed to load or swap processes.

Virtual Address Space is the logical view of process storage in memory. When a process generates a virtual address, the operating system will then translate this to a physical address in memory. This translation relies on a page table that keeps track of the existence of pages in physical memory. Each page entry uses a valid-invalid bit to indicate whether the referenced page is currently loaded. A page fault occurs when a process tries to access a page marked invalid because it is not in physical memory. To handle this, the system retrieves the missing page from the backing store of virtual memory and loads it into physical memory. If the physical memory is full, the operating system will find free pages in memory to page out by page replacement algorithm. Common algorithms include FIFO (First-In, First-Out), LRU (Least Recently Used) and OPT (Optimal Page Replacement). These strategies aim to minimize page faults and optimize overall system performance by carefully selecting pages for replacement based on usage patterns and predicted future needs (Silberschatz et al., 2013, 399-404).

This project focuses on the concept of FIFO and LRU Replacement Algorithms, which are two distinct and contrary approaches for page replacement.

2.1 FIFO Page Replacement

First in, first out (FIFO) algorithm is the simplest algorithm. Pages in memory are held in a FIFO queue. When a new page is brought into memory, the oldest page is chosen to be replaced, and a new page is inserted at the tail of the queue.

The performance of the FIFO is not always good. For example, while the page replaced could be a page that was initialized long ago and no longer in use, it could also be something that was initialized early and in use constantly. Although the program will still work correctly, the replaced active page will cause a page fault immediately to retrieve the page back. Thus, the page fault rate increases and the process execution will be slowed. Belady’s anomaly is also found in FIFO, where page fault rate may increase with the increasing number of allocated frames (Silberschatz et al., 2013, 413,414).

2.2 LRU Page Replacement

Least recently used (LRU) is a better performance page replacement algorithm. LRU means to replace the page that is not in use for the longest period of time. In real cases, when we do not have the future knowledge of the reference string, LRU is practical and has less page fault than FIFO.

There are two feasible implementations for LRU, counters and stack. With the counter approach, the time value of the last reference to every page will be kept record and updated once a page is referenced, while the page with the smallest time value will be replaced. With the stack approach means when a page is referenced, it will be moved to the top of the stack, and the least recently used page, which is at the most bottom of the stack, will be removed (Silberschatz et al., 2013, 416, 417).

3.0 Problem Statement

A chef working full-time in a restaurant keeps a variety of seasonings in bottles on a shelf. However, when the chef is cooking, he cannot always keep all his seasonings on his prep table, since it is usually filled with other ingredients and tools. Sometimes he can place several seasoning bottles on the table; other times, there is only enough space for one, depending on how much room is available. If he needs a seasoning that isn't already on the prep table, he has to replace one of the bottles to make space for the one he needs from the shelf. This makes the chef wonder: what is the most effective way to replace the bottles so he can save effort and time while cooking?

4.0 Related Works

Numerous studies and textbooks have investigated page replacement algorithms, focusing on their impact on virtual memory performance in operating systems. Early foundational works, such as Belady (1966), demonstrated that FIFO can lead to counterintuitive results like Belady's anomaly, where increasing the number of frames may result in more page faults. This anomaly has been widely referenced to highlight the limitations of FIFO in real-world scenarios.

Several academic projects and educational tools have been developed to visualize how FIFO and LRU operate. These visualizations help students and practitioners observe the differences in page replacement behavior under different memory access sequences, reinforcing the importance of selecting the right algorithm for a given workload (Smith et al., 2019).

More recent works have explored hybrid or improved variants of these algorithms, as well as hardware-assisted techniques that can approximate LRU with less overhead. These developments emphasize that while FIFO and LRU remain fundamental to understanding virtual memory management, there is ongoing research into optimizing their real-world performance.

This project builds upon these related works by coding a simulation program that illustrates how FIFO and LRU manage page replacement in a relatable real-world context. By doing so, it aims to strengthen learners' understanding of their practical trade-offs and performance implications.

5.0 Methodology

In this project, our group focused on simulating virtual memory management through two common page replacement algorithms: FIFO (First-In, First-Out) and LRU (Least Recently Used). The objective was to visualize how pages are replaced when memory is full, using a trace table to indicate memory content and page faults at each step.

5.1 Project Management Phases

The development process was divided into five main phases:

1. Conceptualization: Define project goals and objectives to simulate and compare FIFO and LRU page replacement algorithms using C++.
2. Design: Build the memory structure, trace table design, and program logic needed to simulate virtual memory behavior.
3. Implementation: Code two separate C++ programs for FIFO and LRU using vectors and control structures.
4. Testing: Validate both programs with the same reference string to ensure correct memory updates and page fault tracking.
5. Analysis: Evaluate and compare the total number of page faults produced by each algorithm to understand their efficiency.

5.2 Hardware and Software Requirements

- Hardware: Standard personal computer or laptop for running simulations
- Software: C++ compiler (g++), Visual Studio Code for development

5.3 Task Distribution

- Program Development (FIFO and LRU): Lam Yoke Yu
- Testing and Verification: All Members
- Literature Review and Methodology Writing: Tan Yi Ya
- Design & Implementation Documentation: Goe Jie Ying
- Results, Analysis, and Conclusion: Teh Ru Qian
- Final Report Compilation and Editing: All Members

6.0 Design & Implementation

This section outlines the system design and technical development of the C++ programs created to simulate FIFO and LRU page replacement algorithms. The process includes system flow, algorithm logic, and code structure.

6.1 System Design

The system consists of three main components:

1. Input: A predefined page reference string (70120304230303212017O1) and a fixed frame size (3 frames).
2. Processing

   FIFO: Replaces the oldest page in memory using a circular index.

   LRU: Replaces the least recently used page by tracking usage frequency.
3. Output

   A step-by-step trace table printed to the console showing:

   - The current page request
   - Memory frame contents
   - Whether a page fault occurred
   - The final total number of page faults

6.2 System Flowchart

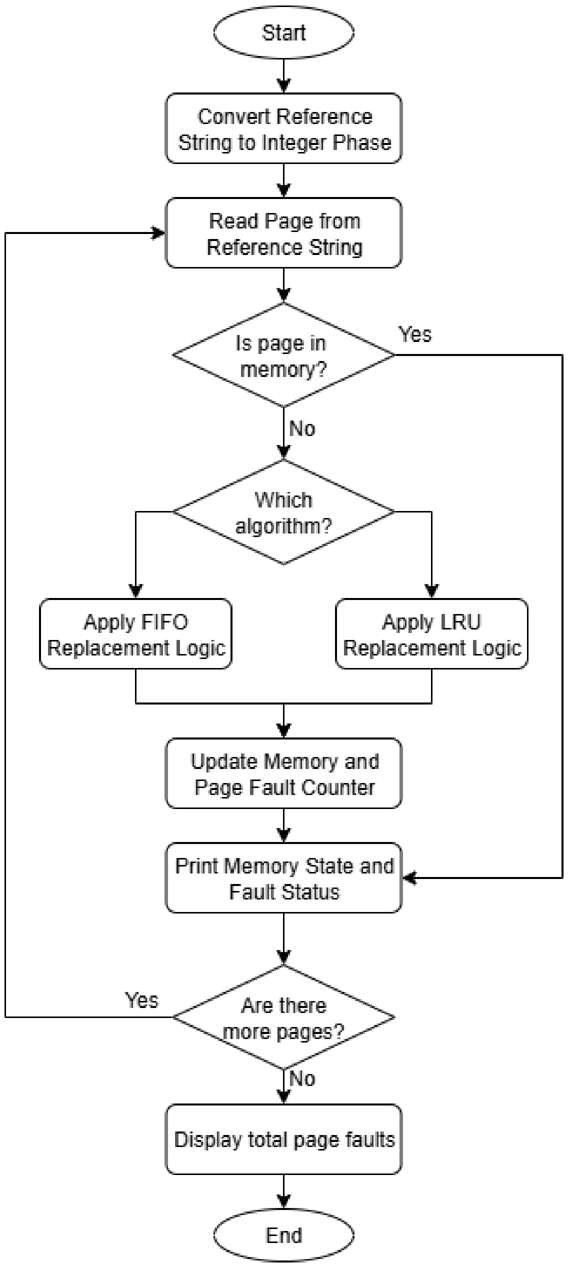

Start
Convert Reference String to Integer Phase
Read Page from Reference String
Is page in memory?
Yes
No
Which algorithm?
Apply FIFO Replacement Logic
Apply LRU Replacement Logic
Update Memory and Page Fault Counter
Print Memory State and Fault Status
Are there more pages?
Yes
No
Display total page faults
End

Figure 6.2 Flowchart of FIFO and LRU Page Replacement Algorithm

Figure 6.2 illustrates the simulation flow for FIFO and LRU page replacement algorithms. The program begins by converting the reference string into a list of pages. Each page is processed sequentially to determine if it is already in memory. If not, the selected algorithm is applied to replace a page. After each operation, the memory state and page fault status are updated and displayed. The loop continues until all page requests have been processed.

6.3 Code Implementation

The following outlines the key implementation steps for both the FIFO and LRU page replacement algorithms. The programs were written in C++ and executed in a console-based environment.

6.3.1 FIFO Implementation

1. Initialize Simulation Environment

- Include necessary libraries: <iostream>, <vector>, and <string>.
- Define and initialize variables: referenceString, frames, pages, and memory.
- Code:

```
string referenceString = "70120304230303212017O1";
int frames = 3;
vector<int> memory(frames, -1);
```

2. Convert Reference String to Pages

- Convert the reference string from characters to integers and store it in a vector.
- Code:

```
for(int i = 0; i < n; i++){
    pages[i] = referenceString[i] - '0';
}
```

3. Check Page Presence in Memory

- Loop through each frame to check if the current page already exists.
- Code:

```
for(int i = 0; i < frames; i++){
    if(memory[i] == page){
        available = 1;
        break;
    }
}
```

4. Apply FIFO Replacement Logic

- If a page fault occurs, replace the page at the current index using circular replacement logic.
- Code:

```
if(!available){
```

```
            pageFaults++;
            memory[currentIndex] = page;
            currentIndex = (currentIndex + 1) % frames;
        }
```

5. Print Memory State and Page Faults

- Display the current memory contents and whether a page fault occurred.

6.3.2 LRU Implementation

1. Initialize Simulation Environment

- Include libraries and define variables: referenceString, frames, memory, and frequency.
- Code:

```
        vector<int> memory(frames, -1);
        vector<int> frequency(frames, 100);
```

2. Convert Reference String to Pages

- Convert characters in the reference string into integer values.
- Code:

```
        for(int i = 0; i < n; i++){
            pages[i] = referenceString[i] - '0';
        }
```

3. Update Frequency on Page Hit

- If the page is found in memory, reset its frequency and increment others.
- Code:

```
        if(memory[i] == page){
            frequency[i] = 0;
            for(int j = 0; j < frames; j++){
                frequency[j]++;
            }
        }
```

4. Apply LRU Replacement Logic

- On a page fault, find the page with the highest frequency value and replace it.
- Code:

```
        int maxIndex = 0;
        for(int i = 1; i < frames; i++){
            if(frequency[i] > frequency[maxIndex]){
                maxIndex = i;
            }
        }
        memory[maxIndex] = page;
        frequency[maxIndex] = 0;
```

5. Print Memory State and Page Faults

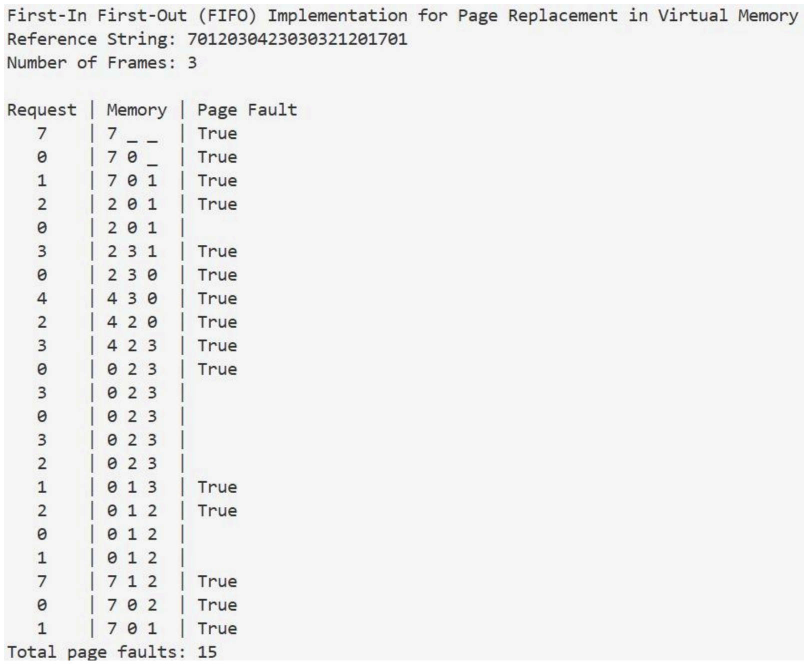
- Display the current memory contents and whether a page fault occurred.

7.0 Results & Discussion

This section compares and analyses the performance of two page replacement algorithms: First-In, First-Out (FIFO) and Least Recently Used (LRU). A reference string of 70120304230303212017O1 and a fixed frame size of three pages were used to carry out the simulation.

7.1 Results

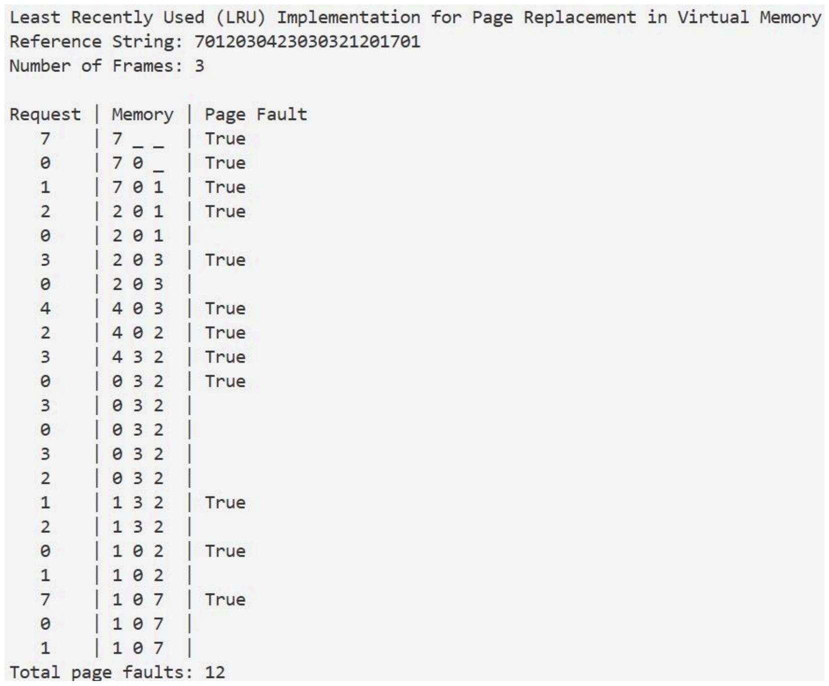
The figures below show the detailed step-by-step memory states for each algorithm in a way that is easy to understand. Each figure shows the page request, the content of the current frame and whether or not a page fault happened at that step. Figure 7.1.1 shows that the FIFO algorithm had 15 page faults, while Figure 7.1.2 shows that LRU had 12 page faults for the same input.



```
First-In First-Out (FIFO) Implementation for Page Replacement in Virtual Memory
Reference String: 70120304230303212017O1
Number of Frames: 3

Request | Memory | Page Fault
   7    | 7 _ _  | True
   0    | 7 0 _  | True
   1    | 7 0 1  | True
   2    | 2 0 1  | True
   0    | 2 0 1  |
   3    | 2 3 1  | True
   0    | 2 3 0  | True
   4    | 4 3 0  | True
   2    | 4 2 0  | True
   3    | 4 2 3  | True
   0    | 0 2 3  | True
   3    | 0 2 3  |
   0    | 0 2 3  |
   3    | 0 2 3  |
   2    | 0 2 3  |
   1    | 0 1 3  | True
   2    | 0 1 2  | True
   0    | 0 1 2  |
   1    | 0 1 2  |
   7    | 7 1 2  | True
   0    | 7 0 2  | True
   1    | 7 0 1  | True
Total page faults: 15
```

Figure 7.1.1 FIFO Implementation for Page Replacement



```
Least Recently Used (LRU) Implementation for Page Replacement in Virtual Memory
Reference String: 70120304230303212017O1
Number of Frames: 3

Request | Memory | Page Fault
   7    | 7 _ _  | True
   0    | 7 0 _  | True
   1    | 7 0 1  | True
   2    | 2 0 1  | True
   0    | 2 0 1  |
   3    | 2 0 3  | True
   0    | 2 0 3  |
   4    | 4 0 3  | True
   2    | 4 0 2  | True
   3    | 4 3 2  | True
   0    | 0 3 2  | True
   3    | 0 3 2  |
   0    | 0 3 2  |
   3    | 0 3 2  |
   2    | 0 3 2  |
   1    | 1 3 2  | True
   2    | 1 3 2  |
   0    | 1 0 2  | True
   1    | 1 0 2  |
   7    | 1 0 7  | True
   0    | 1 0 7  |
   1    | 1 0 7  |
Total page faults: 12
```

Figure 7.1.2 LRU Implementation for Page Replacement

Table 7.1 provides a summary of the results:

Algorithm	Total Page Faults
FIFO	15
LRU	12

Table 7.1 Summary of Results

7.2 Discussion

The result shows that the two algorithms work very differently with the same input. The LRU algorithm caused 12 page faults, while the FIFO algorithm caused 15 page faults. This means that LRU was better at keeping memory from being disrupted.

This result is alignment with theoretical expectations. The basic concept behind the FIFO algorithm is to replace the page that has been in memory the longest, despite it having been used recently. Because of this, it might remove pages that are still being used frequently, which would increase the fault rate. This was shown in the results, where redundant failures resulted from the premature replacement of frequently reused pages, such as 0.

By contrast, LRU always removes the least recently accessed page when making replacement decisions based on usage history. This method reduces unnecessary errors and better preserves frequently accessed pages. Thus , LRU was better at adapting to the temporal locality of the reference string.

To better understand this concept, consider the analogy of a chef in a busy restaurant kitchen. The chef has limited space on the prep table and needs to choose which seasoning bottles to keep close at hand. The rest of the seasonings are stored on a shelf. If the chef uses FIFO, he will always remove the seasoning that has been on the table the longest, even if it is used frequently. This could slow down his cooking when he needs to fetch it again. If he uses LRU, he will remove the seasoning that hasn't been used in a while, keeping the most commonly used items nearby. This results in smoother and faster cooking, just like how LRU leads to better memory performance by keeping active pages in memory.

8.0 Future Work & Conclusion

8.1 Future Work

The following ways are suggested for further development:

- Extension to variable frame sizes, to assess the scalability and adaptability of each algorithm.
- Development of a graphical user interface (GUI) to allow dynamic visualization of memory frame operations and faults.
- Measure additional metrics like hit rate, execution time and memory usage to analyze algorithm efficiency from multiple perspectives.

8.2 Conclusion

In this project, we compared and simulated the First-In First-Out (FIFO) and Least Recently Used (LRU) page replacement algorithms used in virtual memory management. We built programs in C++ that used three memory frames to handle a fixed page reference string. According to the results, LRU performed better than FIFO, generating only 12 page faults rather than 15 page faults. This indicates that LRU is more efficient since it helps it keep relevant pages in memory by taking into consideration how recently a page was used.

Through simulation and the use of a real-world analogy (a chef managing limited seasoning space), we demonstrated the importance of smart memory management. LRU's ability to retain recently used pages made it more suitable for reducing faults and improving system performance. This project helped us understand the practical impact of memory algorithms in operating systems and how they can be modeled and evaluated effectively through coding.

9.0 Acknowledgement

We would like to thank our lecturer, Dr Muhammad Zafran bin Muhammad Zaly Shah and Dr Danlami Gabi, for their guidance, support and helpful feedback throughout this project. Their advice helped us better understand how virtual memory and page replacement algorithms work. We also thank all our group members for working hard and supporting each other during the research, coding, testing and presentation. Everyone gave their best effort and worked well as a team. Lastly, we are grateful for the useful online resources and tutorials that helped us learn more about FIFO and LRU algorithms.

10.0 References

[1] Silbershatz, Galvin, and Gagne, "Operating Systems Concepts, 9th Edition, 2013, John Wiley & Sons.

[2] GeeksforGeeks, "Page Replacement Algorithms in Operating Systems," *GeeksforGeeks*, Jul. 16, 2015. https://www.geeksforgeeks.org/operating-systems/page-replacement-algorithms-in-operating-systems/

[3] William Stallings, Operating Systems: Internals and Design Principles, 6th Edition, 2008, Prentice-Hall.

[4] McHoes, A.M. and Flynn, I.M., Understanding Operating System, 6th Edition, Course Technology, Cengage Learning, 2011.

[5] H.M. Deitel, Operating Systems, 3rd Edition, Pearson Prentice Hall.

[6] Smith, J., Lee, K., & Chen, Y. (2019). Visual Simulation of Page Replacement Algorithms for Educational Purposes. *Journal of Computer Science Education*.

[7] Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*.

Video Links

Youtube Links: https://youtu.be/kypghXbJoCQ

Appendix

Code for FIFO Implementation

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;

int main(){
    // Initialisation
    string referenceString = "7012030423030321201701";
    int n = referenceString.length();
    vector<int> pages(n);
    int frames = 3;
    vector<int> memory(frames, -1);
    int pageFaults = 0;

    for(int i = 0; i < n; i++){
        pages[i] = referenceString[i] - '0';
    }

    // Print Information
    cout
        << "First-In First-Out (FIFO) Implementation for Page
Replacement in Virtual Memory\n"
    << "Reference String: " << referenceString << endl
    << "Number of Frames: " << frames << endl << endl;

    // FIFO Implementation
    int currentIndex = 0;
    bool available = 0;
    // Print trace table header
    cout << "Request | Memory | Page Fault\n";
    for(int page : pages){
        // Check if the page is in memory
        available = 0;
        for(int i = 0; i < frames; i++){
            if(memory[i] == page){
                available = 1;
                break;
            }
        }
        // If not available add to the memory
        if(!available){
            pageFaults++;
            memory[currentIndex] = page;
            currentIndex = (currentIndex + 1) % frames;
        }

        // Print trace table
        cout << "   " << page << "    | ";
        for (int frame : memory){
            if(frame != -1){
                cout << frame << " ";
            }
            else{
                cout << "_ ";
            }
        }
        cout << " | ";
        if(!available){
            cout << "True\n";
        }
        else{
            cout << "\n";
        }
    }
    cout << "Total page faults: " << pageFaults << "\n";
    return 0;
}
```

Code for LRU Implementation

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;

int main(){
    // Initialisation
    string referenceString = "70120304230303212017 01";
    int n = referenceString.length();
    vector<int> pages(n);
    int frames = 3;
    // Variable to track memory
    vector<int> memory(frames, -1);
    // Variable to track frequency
    vector<int> frequency(frames, 100);
    int pageFaults = 0;

    for(int i = 0; i < n; i++){
        pages[i] = referenceString[i] - '0';
    }

    // Print Information
    cout
        << "Least Recently Used (LRU) Implementation for Page
Replacement in Virtual Memory\n"
    << "Reference String: " << referenceString << endl
    << "Number of Frames: " << frames << endl << endl;

    // LRU Implementation
    bool available = 0;
    // Print trace table header
    cout << "Request | Memory | Page Fault\n";
    for(int page : pages){
        // Check if the page is in memory
        available = 0;
        for(int i = 0; i < frames; i++){
            if(memory[i] == page){
                available = 1;
                frequency[i] = 0;
                for(int j = 0; j < frames; j++){
                    frequency[j]++;
                }
                break;
            }
        }

        // If not available add to the memory
        if(!available){
            pageFaults++;
            int maxIndex = 0;
            for(int i = 1; i < frames; i++){
                if(frequency[i] > frequency[maxIndex]){
                    maxIndex = i;
                }
            }
            memory[maxIndex] = page;
            frequency[maxIndex] = 0;
            for(int j = 0; j < frames; j++){
                    frequency[j]++;
                }
        }
        // Print trace table
        cout << "   " << page << "    | ";
        for (int frame : memory){
            if(frame != -1){
                cout << frame << " ";
            }
            else{
                cout << "_ ";
            }
        }
        cout << " | ";
        if(!available){
            cout << "True\n";
        }
        else{
            cout << "\n";
        }
    }
    cout << "Total page faults: " << pageFaults << "\n";
    return 0;
}
```