A

MINI PROJECT REPORT ON

"Page Replacement Algorithms in OS"

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE SECONF YEAR OF

Bachelor of Technology in

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

Mr. Shouryajit Adip Patil	03
Mr. Shubham Vijaykumar Patil	15
Mr. Ajitkumar Pundlik Rathod	19
Mr. Sanket Ramchandra Nirmalkar	20

UNDER THE GUIDANCE OF

Prof. Savitha S.

TATYASAHEB KORE INSTITUTE OF ENGINEERING AND TECHNOLOGY,

WARANANAGAR An Autonomous Institute
Academic Year 2022-23

SWVSM's
TATYASAHEB KORE INSTITUTE OF ENGINEERING AND TECHNOLOGY, WARANANAGAR
An Autonomous Institute

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Mini Project report entitled,

"PAGE REPLACEMENT ALGORITHMS IN OS"

Submitted by

Name	Roll No.
Mr. SHOURYAJIT A. PATIL	03
Mr. SHUBHAM V. PATIL	15
Mr. AJITKUMAR P. RATHOD	19
Mr. SANKET R. NIRMALKAR	20

of S. Y. B. Tech in the partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering from Shivaji University, Kolhapur. It is also certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. This Mini project work is a record of students' own work, carried out by them under our supervision and guidance during academic year 2021-22 .

Prof Savita S.	**Prof B.A.Chougule**
(Guide)	**(Mini Project Coordinator)**
Prof.R.B.Patil	**Dr.S.V.Anekar**
(HOD CSE Dept.)	**(Principal)**

(External Examiner)

Date-
Place-Warananagar

ACKNOWLEDGEMENT

We are greatly indebted to our guide **Prof. Savitha S.** for his unstinted support and valuable suggestions. We are grateful to him not only for the guidance, but also for his unending patience and keeping our spirits high throughout. We express our sincere thanks to our beloved Head of the Department,

Prof. R. B. Patil and Principal, **Dr. S. V. Anekar** for being source of inspiration and providing us the opportunity to work on this project.

We extend heartfelt thanks to all the teaching and non-teaching staff of the department of Computer Science and Engineering for their assistance and cooperation.

Finally, we would like to thank our parents and friends for their moral support and encouragement throughout our academics.

Student Name	**Roll No.**	**Sign**
MR. SHOURYAJIT ADIP PATIL	03	
MR. SHUBHAM VIJAYKUMAR PATIL	15	
MR. AJITKUMAR PUNDLIK RATHOD	19	
MR. SANKET RAMCHANDRA NIRMALKAR	20	

Date:

Place: Warananagar

Table of Contents

Chapter no.	Name of Chapter		Page No
1	**Introduction**		1
	1.1	Introduction	1
	1.2	Problem Statement	2
2	**Proposed Design**		3
	2.1	Flowchart	3-5
	2.2	Algorithm	6
3	**Solution Techniques**		7
	3.1	Data Structures / Explanation of Algorithm Used	7 - 11
4	**System Requirements**		12
	4.1	Hardware Requirement	12
	4.2	Software Requirements	12
5	**Implementation Details**		13
	5.1	System defined functions	13
	5.2	User defined functions	13
	5.3	User Manual (Screen Shots)	18-19
6		**Conclusion**	20
	References		21

Chapter 1
Introduction

1.1 Introduction

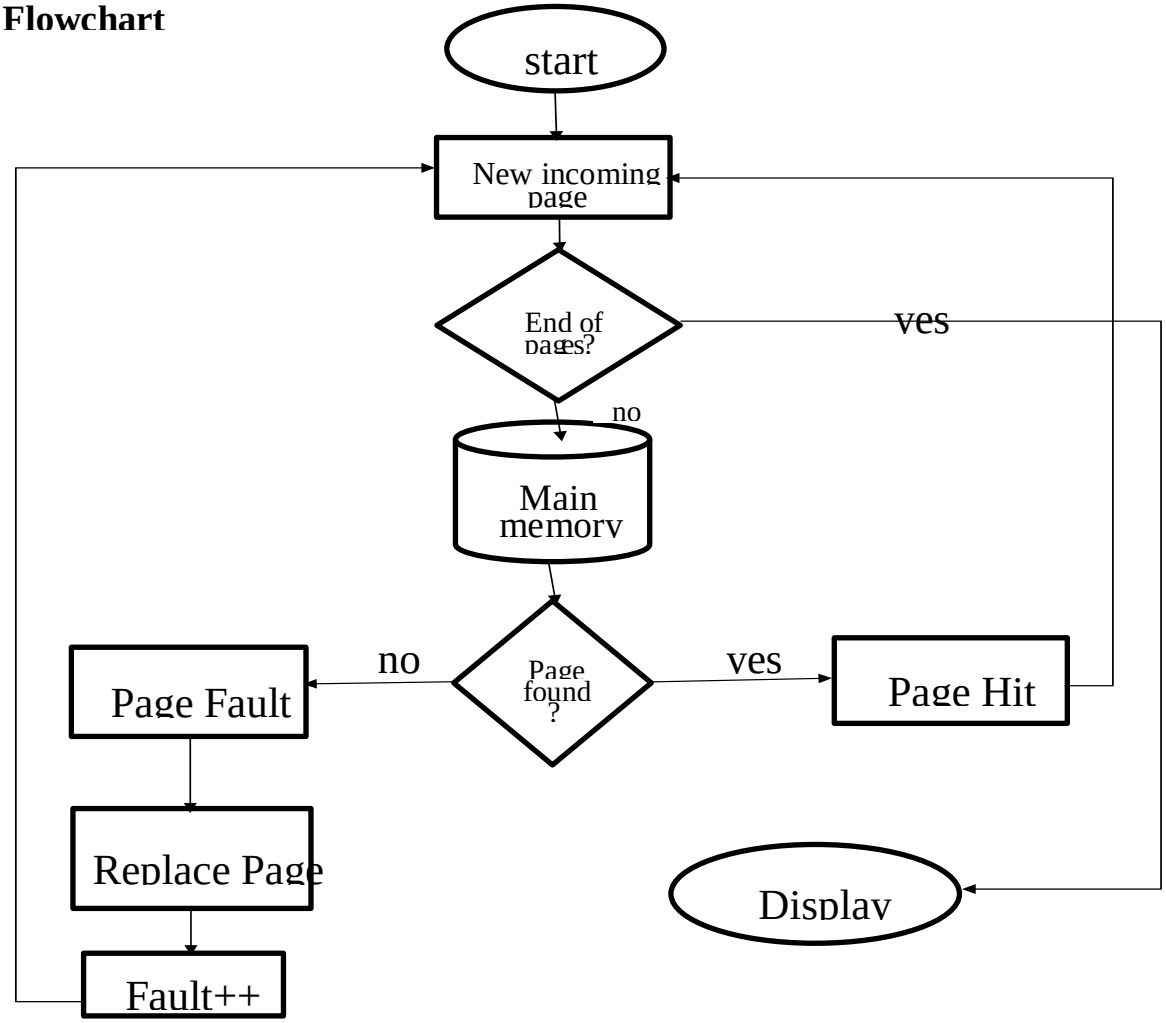
In the memory management replacement of the pages plays important role. When the process is in execution (Tanenbaum 2008, Soares et al. 2010), pages are required to be in the main memory If page is not found in the main memory, then it is called as page fault. When a page fault occurs, the process needs to find the page that caused the page fault in disk. If there is a free space in main memory it will put in this area. If there isn't free space it is necessary to make a page replacement with the algorithm that is implemented. After that the page that is removed will be written in disk. Then the process can continue the job, because the required page is in the main memory and the process can read it. There are many page replacement algorithms (Tanenbaum 2008) such as: The Optimal, The Not Recently Used (NRU), The Not Frequently Used (NFU), The First-inFirst-out (FIFO), The Least Recently Used (LRU), Aging, The Working Set, The page replacement concept can be used in many areas of computer design.

1.2**Problem Statement:**

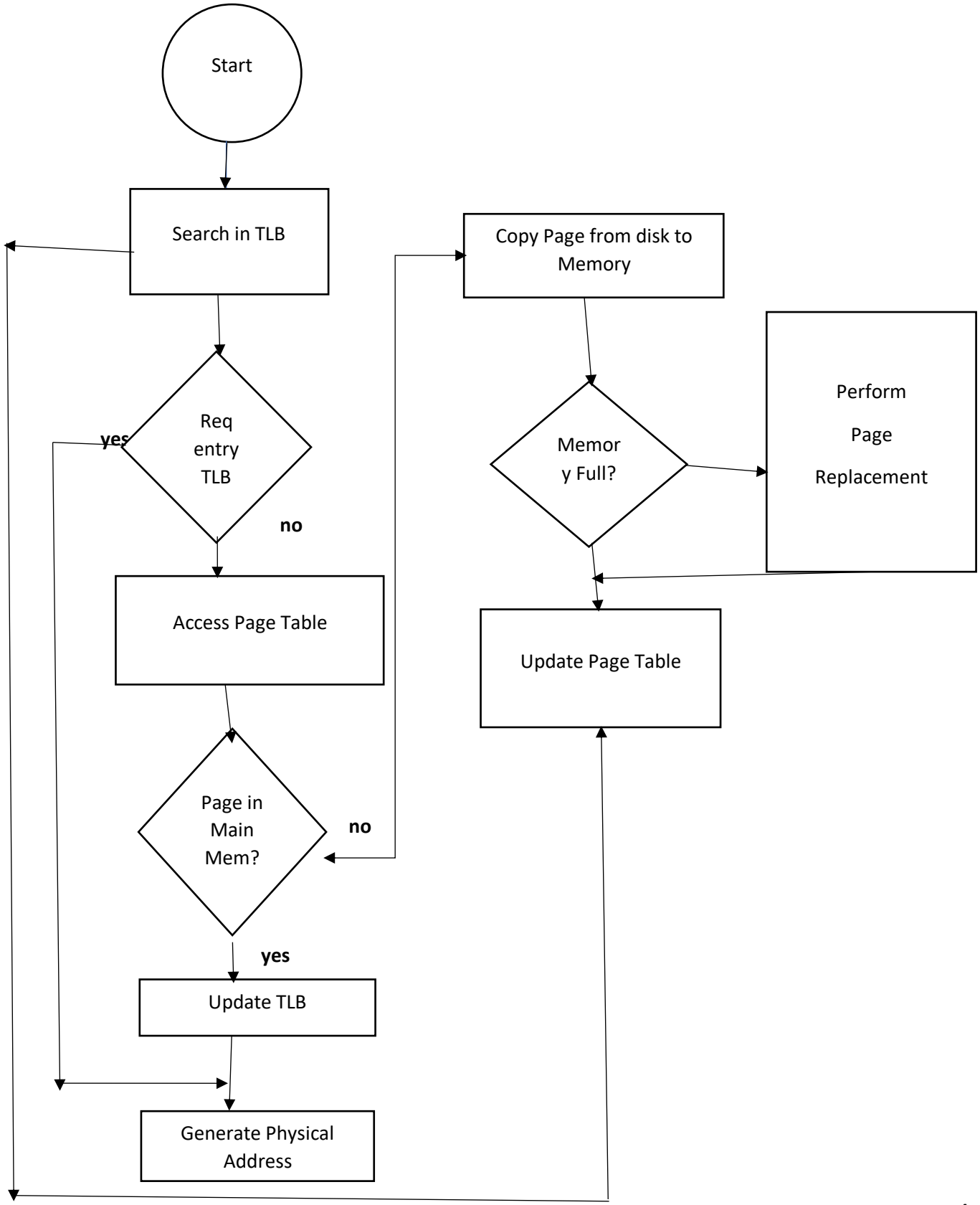
To design and develop a project to compare first in first out (FIFO) and optimal page replacement algorithms, for given set of page numbers, and page frames.

Chapter 2
Proposed Design

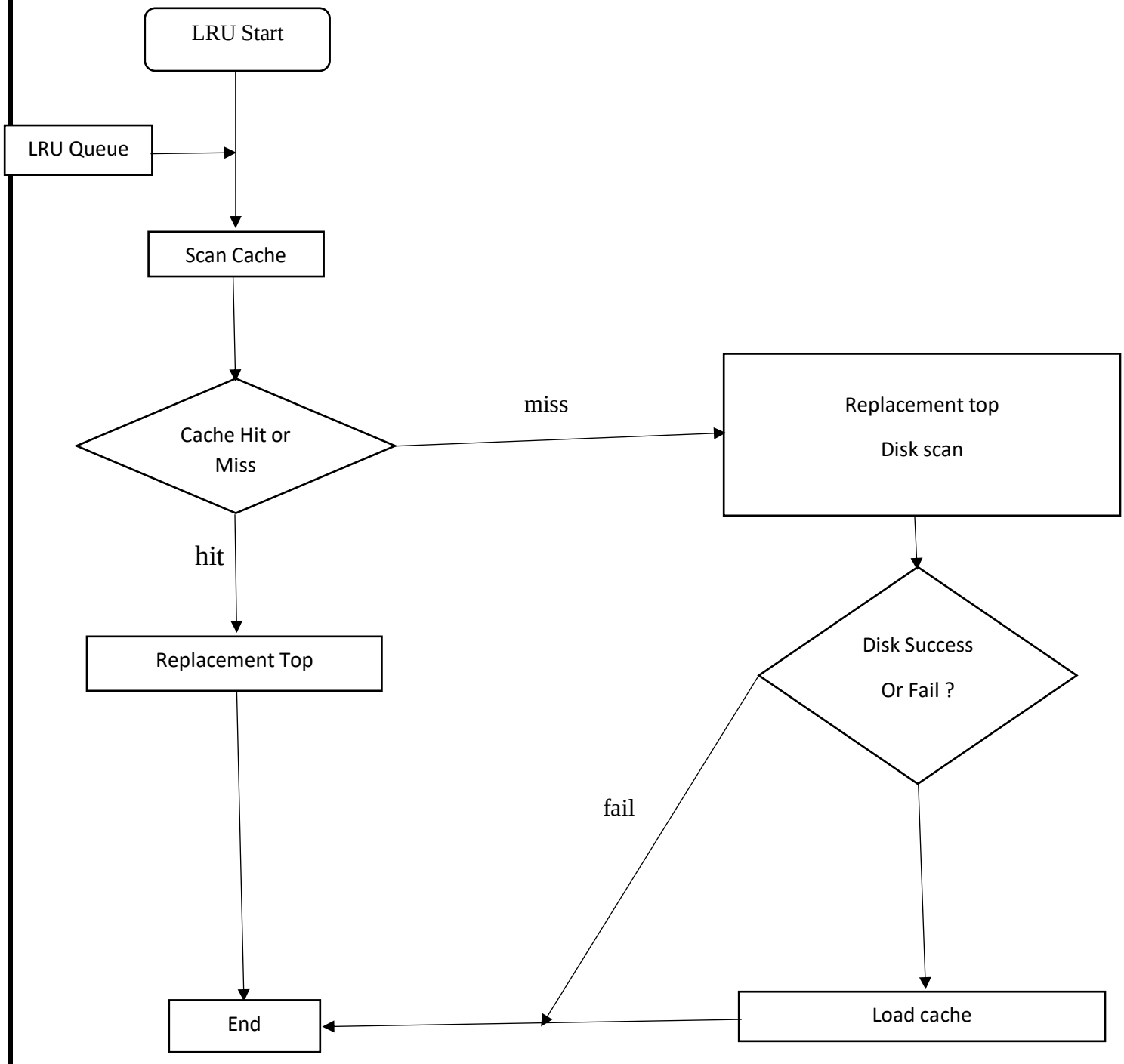
2 Proposed Design
2.1 Flowchart



2.Optimal page replacement :



3.LRU page Replacement :



2.2Algorithm:

1. If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

b) Simultaneously maintain the pages in the queue to perform FIFO.

c) Increment page fault ii) Else

If current page is present in set, do nothing.

Else

a) Remove the first page from the queue as it was the first to be entered inthememory

b) Replace the first page in the queue with the current page in the string.

c) Store current page in the queue.

d) Increment page faults.

2. Return page faults.

Chapter 3

Solution Techniques:

In operating systems that use paging for memory management, page replacement algorithms are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

FIFO :

The FIFO (First-In, First-Out) page replacement algorithm is a simple and intuitive method used by operating systems to manage memory when a page fault occurs. In this algorithm, the page that has been in memory the longest is selected for replacement.

Here is the full information about the FIFO page replacement algorithm:

Data Structure: The FIFO algorithm uses a queue data structure to keep track of the order in which pages are loaded into memory. The queue follows a First-In, First-Out order, meaning that the page that entered the queue first will be the first one to be replaced.

Page Table: The operating system maintains a page table that keeps track of the pages currently in memory and their corresponding information, such as the page number, frame number, and any additional metadata.

Page Fault Handling: When a page fault occurs, indicating that the requested page is not present in memory, the operating system initiates the page replacement process.

Selecting the Victim Page: The page at the front of the queue is selected as the victim page for replacement since it has been in memory the longest. The front page is the one that was loaded first and has spent the most time in memory.

Updating the Page Table: The page table is updated to reflect the replacement. The victim page is removed from memory, and the new page is loaded into the corresponding frame. The page table is then updated with the new page's information.

Updating the Queue: After the replacement, the queue is updated to reflect the new order of pages in memory. The new page is added to the back of the queue, and the victim page is removed from the front.

Repeat: The process repeats whenever a page fault occurs, and the queue keeps growing until all available memory frames are occupied.

It's important to note that the FIFO algorithm suffers from the "Belady's anomaly," which means that increasing the number of page frames can sometimes lead to an increase in the number of page faults. This anomaly occurs when a page that was previously evicted from memory by FIFO would have stayed in memory if there were fewer page frames available.

Despite its simplicity, the FIFO algorithm is not always the most efficient in terms of overall page fault rate. Other page replacement algorithms, such as LRU (Least Recently Used) or Optimal, often provide better performance. However, FIFO remains useful due to its simplicity and low computational overhead.

LRU :

The Least Recently Used (LRU) page replacement algorithm is a popular method used by operating systems to manage memory. It works based on the principle that pages that have not been used recently are less likely to be used in the near future. The LRU algorithm replaces the least recently used page in memory with a new page when a page fault occurs. Here's a full explanation of how the LRU algorithm works:

Maintain a data structure to track the usage of pages, such as a linked list or a queue. Each page has a timestamp or a counter associated with it to indicate when it was last accessed.

When a page is referenced (read or written to), the operating system checks if it is present in the memory. If it is present, the page's timestamp is updated to the current time.

If the referenced page is not present in memory (a page fault occurs), the operating system needs to select a page to evict. It examines the timestamps or counters of all pages to identify the least recently used page.

The page identified as the least recently used is then evicted from memory to make space for the new page. This eviction may involve writing the page's contents back to disk if it has been modified.

The new page is then brought into memory and its timestamp or counter is set to the current time.

By using the LRU algorithm, the operating system ensures that the pages most likely to be used in the near future are kept in memory, while less frequently used pages are evicted. This helps to minimize page faults and optimize memory usage.

Note: Implementing the LRU algorithm efficiently can be challenging, especially when the number of pages is large. Various data structures and algorithms, such as doubly linked lists or hash tables, can be used to improve the performance of the LRU algorithm.

Optimal :

The optimal page replacement algorithm, also known as the "OPT" algorithm or the "Belady's algorithm," is an idealized algorithm for page replacement in an operating system. It assumes that it has complete information about future memory references.

The OPT algorithm works by replacing the page that will not be used for the longest period of time in the future. It requires knowledge of the entire future sequence of memory references to make the optimal decision.

However, in reality, it is impossible to have complete information about future memory references. Therefore, the OPT algorithm is mainly used as a benchmark to measure the performance of other page replacement algorithms.

To illustrate how the OPT algorithm works, consider the following example:

Suppose we have a memory of size N and a sequence of memory references: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Let's assume the memory size is 3. Initially, the memory is empty. When the first memory reference 1 arrives, we place it in an empty frame. Then, we encounter memory reference 2, and we place it in another empty frame. The next memory reference is 3, and we place it in the remaining empty frame.

Now, when we encounter memory reference 4, we need to replace a page because all frames are occupied. Since we have complete knowledge of future memory references, we can look ahead and see that the page 1 will not be used until after the references 2, 3, 4. Therefore, we replace page 1 with page 4.

The process continues, and we repeat the same steps until the end of the sequence. With complete knowledge, we make optimal replacements at each step.

Again, it's important to note that the OPT algorithm is not practical for real-world implementations because it requires future information, which is not available. Instead, various approximation algorithms like LRU (Least Recently Used), FIFO (First In, First Out), and various other algorithms are used to achieve a reasonable approximation of optimal page replacement. These algorithms make decisions based on past behaviour and access patterns rather than future information.

Page replacement algorithms are used in operating systems to manage the limited physical memory efficiently. When a process requests a page that is not present in the main memory, a page fault occurs, and the operating system needs to decide which page to replace in order to bring the requested page into memory.

Here are some popular page replacement algorithms along with a brief explanation of each:

Optimal Page Replacement (OPT):

OPT algorithm replaces the page that will not be used for the longest period of time in the future.

This algorithm requires knowledge of the future page references, which is generally not available in real systems.

It is used as a theoretical benchmark to measure the performance of other algorithms.

First-In-First-Out (FIFO):

FIFO algorithm replaces the page that has been in memory the longest.

It is a simple and easy-to-implement algorithm.

However, it suffers from the "Belady's Anomaly," where increasing the number of frames can result in more page faults.

Least Recently Used (LRU):

LRU algorithm replaces the page that has not been used for the longest period of time.

It is based on the principle of locality, assuming that pages that have been used recently are more likely to be used in the near future.

LRU requires maintaining a list or a queue of pages, which can be costly in terms of memory and computational overhead.

Chapter 4
System Requirements

4.System Requirements

4.1 Hardware Requirement:

1. Personal Computer with minimum configuration:
   - Hard disk =20GB
   - RAM =1GB

4.2 Software Requirement:

1. Windows 11 OS.
2. Language- C++.
3. IDE: Code Blocks
4. Compiler: GCC 6.3

Chapter 5
Implementation Details

5.Implementation Details:

5.1 System defined functions:

1. system("color B4") : used for applying background colour.

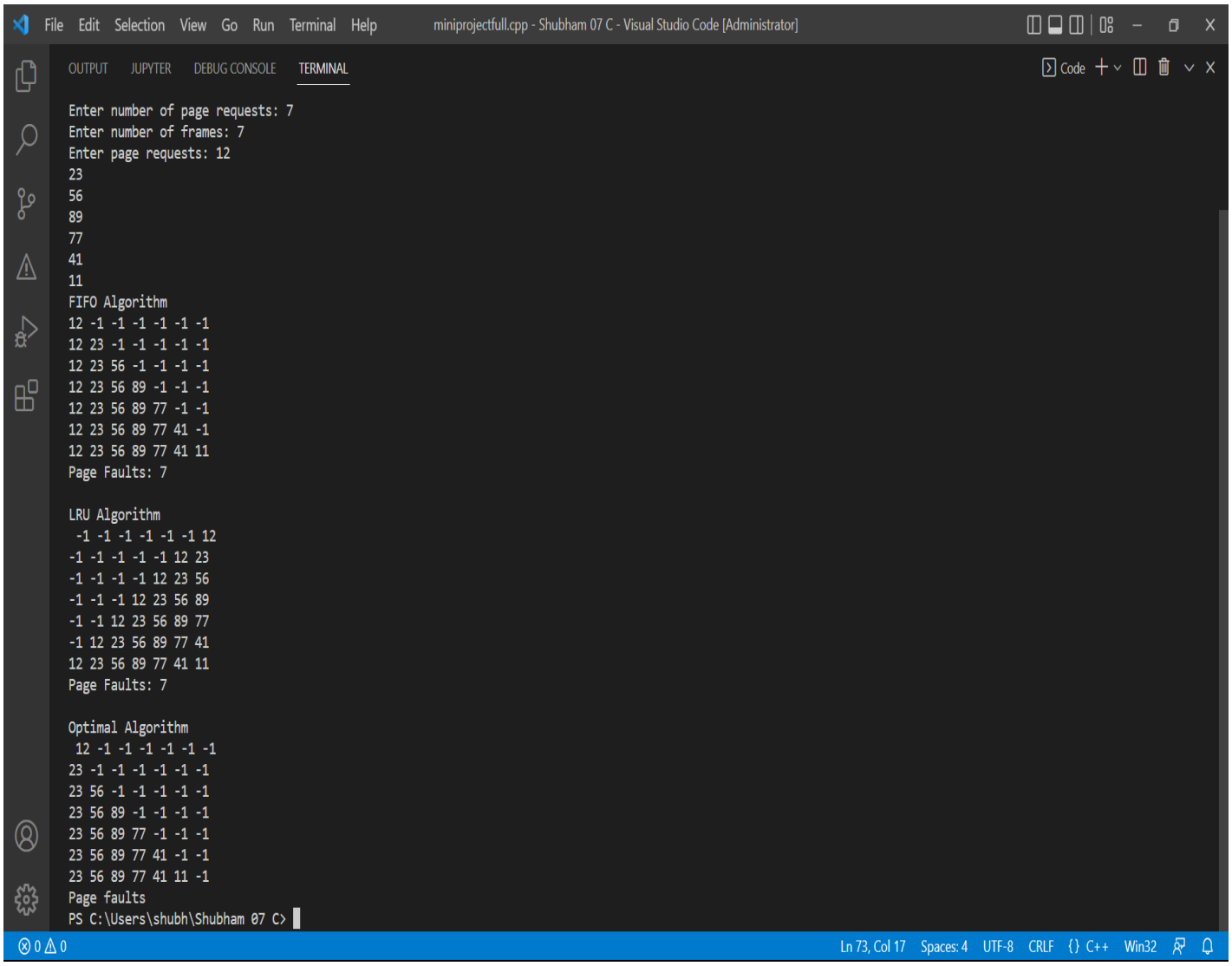
2. system("cls"): It clears the screen when one iteration is complete.

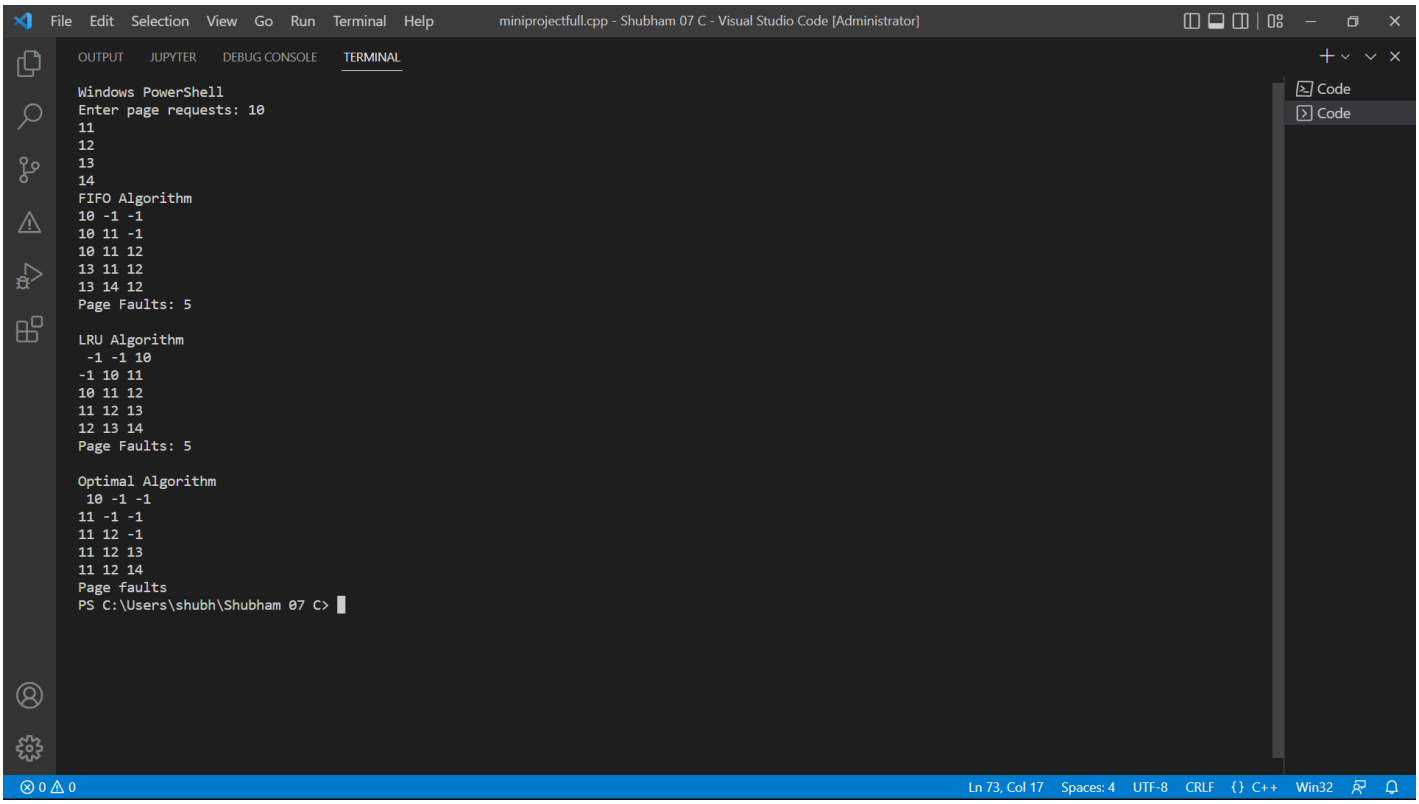
5.2 User defined functions:

1.with return type with argument:

a) int sm(int n, int a[ ]) : It returns integer and take one integer value and int array as arguments.

b) int opt(int n, int a[ ]) : It returns integer and take one integer value and int array as arguments.


File Edit Selection View Go Run Terminal Help
miniprojectfull.cpp - Shubham 07 C - Visual Studio Code [Administrator]
OUTPUT JUPYTER DEBUG CONSOLE TERMINAL
Code
Enter number of page requests: 7
Enter number of frames: 7
Enter page requests: 12
23
56
89
77
41
11
FIFO Algorithm
12 -1 -1 -1 -1 -1 -1
12 23 -1 -1 -1 -1 -1
12 23 56 -1 -1 -1 -1
12 23 56 89 -1 -1 -1
12 23 56 89 77 -1 -1
12 23 56 89 77 41 -1
12 23 56 89 77 41 11
Page Faults: 7
LRU Algorithm
-1 -1 -1 -1 -1 -1 12
-1 -1 -1 -1 -1 12 23
-1 -1 -1 -1 12 23 56
-1 -1 -1 12 23 56 89
-1 -1 12 23 56 89 77
-1 12 23 56 89 77 41
12 23 56 89 77 41 11
Page Faults: 7
Optimal Algorithm
12 -1 -1 -1 -1 -1 -1
23 -1 -1 -1 -1 -1 -1
23 56 -1 -1 -1 -1 -1
23 56 89 -1 -1 -1 -1
23 56 89 77 -1 -1 -1
23 56 89 77 41 -1 -1
23 56 89 77 41 11 -1
Page faults
PS C:\Users\shubh\Shubham 07 C>
Ln 73, Col 17 Spaces: 4 UTF-8 CRLF {} C++ Win32



Here, we used three page replacement Algorithms in OS which are Fifo(First In First Out), LRU(least Recently Used) and Optimal . All the cases execute simultaneously. Here the empty spaces in queue are initialized by -1 that's why we see -1 in program. The number of page faults is calculated accordingly and given as an output. one row contains elements up to the use required number of frames.

Working the Algorithms :

FIFO :

FIFO (First-In, First-Out) is a page replacement algorithm used in operating systems to manage memory in a virtual memory system. It works based on the principle that the oldest page in the memory should be the one to be replaced when a new page needs to be brought in.

Here's how the FIFO page replacement algorithm works:

The operating system maintains a queue, often implemented as a linked list, to keep track of the pages in memory. This queue represents the order in which pages were brought into memory. When a new page needs to be brought into memory and there are no free frames available, the operating system selects the page at the front of the queue (the oldest page) for replacement. The selected page is then removed from the memory and replaced with the new page. The new page is inserted at the end of the queue, indicating that it is the most recently brought-in page. Whenever a page is referenced, its position in the queue does not change. Only the newly brought-in pages are added to the end of the queue. This process continues as long as there are page faults (occurrence of a memory reference to a page not currently in memory).

One advantage of the FIFO algorithm is its simplicity and low overhead, as it only requires maintaining a simple queue. However, it suffers from the "Belady's Anomaly" problem, where increasing the number of page frames can actually result in more page faults. This anomaly occurs because the algorithm does not consider the frequency of page usage or the importance of pages in the queue. Overall, FIFO is a straightforward page replacement algorithm that ensures a fair distribution of memory resources but may not always result in the optimal performance in terms of minimizing page faults.

LRU :

The Least Recently Used (LRU) page replacement algorithm is a commonly used algorithm in operating systems to manage memory and handle page faults. It is based on the principle that the least recently used page is most likely to be the least useful in the future, so it should be replaced when a new page needs to be brought into memory.

Here's how the LRU algorithm works:

Each page in memory is assigned a timestamp or counter to keep track of when it was last accessed. This can be done using a counter that gets incremented on each memory access or using a timestamp that records the actual time of the access. When a page fault occurs (i.e., a referenced page is not present in memory), the operating system checks if there is any free page frame available in memory. If there is, it simply allocates a new page frame and loads the required page into it. If there are no free page frames available, the operating system needs to select a page from memory to replace. In the LRU algorithm, the page with the smallest timestamp or the oldest access time is selected for replacement. Once a page is selected for replacement, the operating system writes the modified page back to disk (if it has been modified) and loads the required page into the selected page frame. The timestamp or counter of the newly loaded page is updated to the current time or counter value to mark it as the most recently used page. By using this approach, the LRU algorithm tries to minimize the number of page faults by replacing the page that has not been accessed for the longest time. This is based on the assumption that if a page has not been accessed recently, it is less likely to be accessed in the near future.

It's worth noting that implementing an efficient LRU algorithm can be challenging, especially in systems with a large number of pages. Various data structures, such as linked lists, arrays, or caches, can be used to keep track of page accesses and timestamps efficiently.

Optimal :

The optimal page replacement algorithm, also known as the OPT algorithm or the MIN algorithm, is an idealized page replacement algorithm that selects the page for eviction that will not be referenced for the longest period of time in the future. While the optimal algorithm is not feasible to implement in a real operating system due to its reliance on future knowledge, it serves as a theoretical benchmark for evaluating the performance of other page replacement algorithms.

The optimal page replacement algorithm works by predicting the future page references and selecting the page that will not be referenced for the longest time. This prediction requires complete knowledge of the future memory references, which is impossible in most scenarios. However, it can be used as a baseline to compare the performance of other algorithms.

To illustrate how the optimal page replacement algorithm works, consider the following example: Assume a memory of fixed size, such as four pages.

The reference string represents the sequence of memory accesses: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Each number represents a page reference.

We initially have an empty memory.

Here's how the OPT algorithm would make decisions:

When the first page reference 1 occurs, we have an empty memory and place it in a page frame.

Memory: [1]

When the second page reference 2 occurs, we place it in another page frame.

Memory: [1, 2]

When the third page reference 3 occurs, we place it in a new page frame.

Memory: [1, 2, 3]

When the fourth page reference 4 occurs, we place it in the last available page frame.

Memory: [1, 2, 3, 4]

When the fifth page reference 1 occurs, we do not need to evict any page since it is already in memory.

Memory: [1, 2, 3, 4]

When the sixth page reference 2 occurs, we do not need to evict any page since it is already in memory.

Memory: [1, 2, 3, 4]

When the seventh page reference 5 occurs, we need to replace a page. At this point, any page could be a candidate since they are all referenced in the future. To determine the optimal page, we look at the future references:

The reference string after this point is 1, 2, 3, 4, 5.

The optimal page to replace is the one that will be referenced the furthest in the future.

In this case, page 3 will not be referenced again, so it is evicted.

Memory: [1, 2, 5, 4]

The process continues in a similar fashion until all page references have been processed. As mentioned earlier, the optimal page replacement algorithm is not practical to implement in real operating systems because it requires future knowledge of memory references. However, it is commonly used as a benchmark to compare the performance of other algorithms.

Chapter 6
Conclusion

6. Conclusion:

We have studied different page replacement algorithms. After analysis we obtain that the optimal page replacement algorithm gives less page fault rate among the other algorithms by creating running program which contains array data structure, user defined functions, different conditional statements and file handling for helping purpose. But it can't be used in the real world, because we don't know the pages that are going to be required in the near future. It is used for comparing the other algorithms efficiency.

References

[1] www.studytonight.com

[2] www.javatpoint.com

[3] www.geeksforgeeks.com

[4] Text book : Operating system concepts 8th edition by Peter Bear Galvin