Page Replacement Algorithms

CSCE – 5640.003
FALL 2024

Professor Amar M. Maharjan
Department of Computer Science
University of North Texas

Shreya Sri Bearelly (11734229)
Divya Sree Dandu (11594287)
Ganesh G. (11700551)
Vikas Varala (11704793)

Overview:

Available memory is critical in today's computing system where optimization of systems' memory usually translates to optimized system's response time. The page replacement algorithms are fundamental parts of the virtual memory, which keeps the most suitable memory pages while making effective use of physical memory that is scarce. This project focuses on implementing and analysing five widely recognized page replacement algorithms: First In, First Out, Optimal, Least Recently Used, Least Frequently Used and Most Frequently Used. It is to uncover these mechanisms, model how they work with the help of artificial reference strings, and analyze their efficiency by number of page faults.

Algorithms to be implemented:
FIFO (First-In-First-Out): Simplest algorithm. Replaces the oldest page.
Optimal: Of all the methods, theoretically the best approach, but it needs the knowledge of the future.
LRU (Least Recently Used): Removes the page that has been inactive for the longest time, or the first in the list. Second Chance: A modification of FIFO where, in addition to using a reference bit, the memory requests another go.
LFU (Least Frequently Used): Removes the page used most infrequently.
MFU (Most Frequently Used): A page that has the highest frequency in the cluster is replaced by the page which has the second largest occurrences.

Problem Statement:

Present day computing systems are in many occasions faced with processing requirements that cannot fit into physical memory. This makes use of virtual memory in which only important pages can be in the memory at a given time. When the virtual address corresponding to the required page is not in memory (a page fault occurs), it is necessary to bring it into memory, and it is typical to replace some other page. The major problem which arises is that of deciding which page should be replace in order to minimize the page faults that the system experiences in the subsequent time. Thus, wrong choices of page replacement policies result in serious performance degradation and knowing the algorithms and their proper selection depending on the working conditions and computer characteristics, is very important.

Problems Encountered in page replacement algorithm:

1. **High Page Fault Rate:** The use of such algorithms as FIFO can mean experiencing high levels of page faults even where the pages repeatedly accessed are rapidly called (as in the case of the "Belady's Anomaly").
2. **Complexity of Implementation:** Such algorithms as Optimal and LRU need more complex methods (for instance tracking future references or simply page access history).
3. **Overhead in Tracking Data:** To understand the frequency, the algorithms like LFU and MFU require counters which naturally add to memory and computational complexity. Dependency on
4. **Reference Patterns:** The accuracy deteriorates due as a result of its sensitivity to the reference string, whether it is cyclic or random.

5. **Frame Allocation:** Others argue that determining the number of frames for processes affects the resultant performance. If too little frames are provided then more page fault occur**.**

Importance:

The page replacement algorithms play an important role in managing the operating system memory where availability of physical memory is a constraint. Because they decide which pages must be replaced when a page fault is encountered, they directly affect the system's performance by reducing response time and efficiently using resources. A good algorithm minimizes the page fault rates in order to favorably enhance the normal flow of programs and positively impact the system. This is critical, particularly with multiple processes in a system competing over memory space and with applications demanding fast computations making page replacement an essential concept in quality system flow.

Efficient page replacement directly impacts the following areas:
System Performance: The goal of decreasing the page faults somehow reduces the cost or overhead of data retrieval from secondary storage which is much slower than physical memory access cost.
Scalability: Applications created today, including massive data stores, computational models, and services residing on cloud platforms, need flexible memory management with high memory consumption capabilities.
Multitasking Efficiency: In of the operating systems where there are several processes with each having its requirements in the use of memory, the best that is applied is page replacement policy.
Energy Consumption: Minimizing page faults may decrease the number of disk I/O operations therefore reducing the energy consumption in power controlled systems like portable and embedded systems.

Detailed Breakdown of Each Algorithm:

1.FIFO(First-In-First-Out):

Description: FIFO is the simplest page replacement algorithm. Pages are replaced in the order they were brought into memory, regardless of their usage. The oldest page is replaced when a new page is needed.
Pros: Simple to implement and understand. Requires minimal bookkeeping.
Cons: Suffers from Belady's Anomaly, where increasing the number of frames may increase the page fault rate. Does not consider page usage patterns, leading to suboptimal performance.

2.Optimal:

Description: The page replacement that can be done only at its best replaces the page that will be required for usage later after the largest amount of time in the time of Archon. It results in achieving the least possible page fault rate; however, it is inapplicable in the real-time system since it assumes that future references are unknown.
Pros: The best form of performance is provided. These have the benefit of being used as a reference when comparing other algorithms.

Cons: Cannot be realized in real systems, because it presupposes the knowledge about the forthcoming memory references. Holds definite theoretical and computational advantage within the confines of simulation.

3.Least Recently Used(second chance):

Description: LRU removes the page that has not been accessed for quite a some time. Second Chance is just an addition to the standard FIFO method. It provides each page with a second opportunity with a reference bit. When a page is discussed for replacement, if the reference bit of that page is set to 1, then that page is allowed again after flipping the reference to 0.
Pros: In most cases, the specific identification method yields better results compared with the use of FIFO. The planning enables tracking of the actual usage patterns necessary for decision-making.
Cons: More than one usage counter usually has to be maintained, which adds more overhead.

4.Least Frequently Used:

Description: LFU replaces the page that has used least frequently in the system. What it fails to consider is that sites which are accessed few times are likely to remain so in the future.
Pros: The patterns of use that are long term can be tracked well. Effectively used in a scenario where statically frequently accessed pages are frequent motivated.
Cons: High overhead because of the count frequency maintainers. May cause the aging problem in which infrequently accessed old pages have a high frequency count and cannot be replaced.

5.Most Frequently Used:

Description: Deleted the most frequently used page as the theory is that in most cases the visitors are unlikely to revisit the page if it was accessed most often by the site users.
Pros: It is used as a solution to LFU because this policy can be effective in certain conditions when frequently accessed pages are useless after some time
Cons: High overhead necessary for upkeep of the equipment, namely the frequency counters.

Background:

Page replacement algorithms form the core of memory management in the current operating systems. In such systems the processes or programs run as if they have ample amount of memory, when in actual sense, memory is restricted. The operating system does this by partitioning the memory in defined sizes known as pages. A page fault occurs when a program goes to the operating system to say that it requires a page that is not in main memory. To solve this problem, the operating system reads the wanted page from second level storage, such as a disk memory, into the main memory.
But physical memory can accommodate only a certain amount of pages. When we reach the level of memory update, it is time for the operating system to free up one page and replace it with a new one. But this decision of which page has to be evicted is made with the help of page replacement algorithms. The goal of a good algorithm will be to reduce page faults while optimizing memory usage at the least computational cost.

FIFO
Reference String: 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
Frame Count: 3
Total No. of Page Faults for FIFO is = 15, Total No. of Page Hits for FIFO is= 5

3	3	3	2	2	2	2	3	3	3
	1	1	1	5	5	5	5	5	2
		4	4	4	4	1	1	1	1
Fault	Fault	Fault	Fault	Fault	Hit	Fault	Fault	Hit	Fault

3	1	1	1	1	1	1	5	5	5
2	2	2	2	2	3	3	3	0	0
0	0	0	0	0	0	4	4	4	1
Fault	Fault	Hit	Hit	Hit	Fault	Fault	Fault	Fault	Fault

Optimal
Reference String: 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
Frame Count: 3
Total No. of Page Faults for optimal is= 11, Total No. of Page Hits for optimal is= 9

3	3	3	2	5	5	5	5	5	2
	1	1	1	1	1	1	1	1	1
		4	4	4	4	4	3	3	3
Fault	Fault	Fault	Fault	Fault	Hit	Hit	Fault	Hit	Fault

2	2	2	2	2	3	4	5	5	5
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
Fault	Hit	Hit	Hit	Hit	Fault	Fault	Fault	Hit	Hit

LRU
Reference String: 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
Frame Count: 3
Total No. of Page Faults for LRU is = 16, Total No. of Page Hits for LRU is = 4

3	3	3	2	2	2	1	1	1	2
	1	1	1	5	5	5	3	3	3
		4	4	4	4	4	4	5	5
Fault	Fault	Fault	Fault	Fault	Hit	Fault	Fault	Fault	Fault

2	2	2	2	2	2	2	5	5	5
0	0	0	0	0	0	4	4	4	1
5	1	1	1	1	3	3	3	0	0
Fault	Fault	Hit	Hit	Hit	Fault	Fault	Fault	Fault	Fault

LFU
Reference String: 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
Frame Count: 3
Total No. of Page Faults for optimal is= 11, Total No. of Page Hits for optimal is= 9

3	3	3	2	2	2	1	1	5	5
	1	1	1	5	5	5	3	3	2
		4	4	4	4	4	4	4	4
Fault	Fault	Fault	Fault	Fault	Hit	Fault	Fault	Fault	Fault

5	1	1	1	1	3	3	3	0	0
2	2	2	2	2	2	4	4	4	1
0	0	0	0	0	0	0	5	5	5
Fault	Fault	Hit	Hit	Hit	Fault	Fault	Fault	Fault	Fault

MFU
Reference String: 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
Frame Count: 3
Total No. of Page Faults for optimal is= 15, Total No. of Page Hits for optimal is= 5

3	3	3	2	2	2	2	3	3	3
	1	1	1	5	5	5	5	5	2
		4	4	4	4	1	1	1	1
Fault	Fault	Fault	Fault	Fault	Hit	Fault	Fault	Hit	Fault

3	1	1	1	1	1	1	5	5	5
2	2	2	2	2	3	3	3	0	0
0	0	0	0	0	0	4	4	4	1
Fault	Fault	Hit	Hit	Hit	Fault	Fault	Fault	Fault	Fault

Implementation:

Solution Overview:

The page replacement algorithms discussed in this paper were coded in C++ and tested on Linux/Ubuntu.

FIFO and Optimal Algorithms:

These algorithms developed by Shreya Sri Bearelly address page replacement under basic order (FIFO) and future predicting (Optimal) strategies.

LRU second chance Algorithms:

These algorithms are designed by Divya Sree Dandu to emphasize the usage history and while building upon the usual FIFO, Second Chance offers improvements.

LFU and MFU Algorithms:
These algorithms are developed by Ganesh G. all of them count the frequency in order to evaluate the results of page replacement in the situation of varying access patterns.
Testing and Analysis:
The reference strings, test cases as well as the measurement of page fault rates and statistical examinations were compiled and cross-checked by Vikas Varala.

Implementation Details:
Programming Language: C++.
Operating System: Linux/Ubuntu.
Test Cases: Includes different frame count.
5 files with 10 frames.
5 files with 20 frames.
Example Test Input Files:
referenceString1.txt, referenceString2.txt….referenceString10.txt etc.

FIFO Implementation:
The program is based on First-In-First-Out (FIFO) page replacement technique to find out page faults if given a string of references and number of frames respectively.t. the function named firstinfirstoutpagereplacement algorithm employs the FIFO concept with the help of passing the page arrive time in the form of queue, and the unordered set implement the identification of currently available pages in the frame. This occurs when a page is referred from such a memory location that such a page is not currently stored in the memory. To accomplish the above, we keep deleting the old page to create room for the new page if the memory is full. The processTestCase function takes two parameters as integer arrays: a reference string, and the frame count; extracts this data and uses the firstinfirstoutpagereplacement function to determine the total page faults. The main function takes the name of the files as command line arguments and processes each file with printing out the reference string, frame count, and total page faults for the input. This modular implementation guarantees simplicity in organizing the file handling routine and algorithm structures, respectively.

Optimal Implementation:
This program which simulates the Optimal Page Replacement algorithm finds the least page faults for a given sequence of page requests (reference string) and frame count. Its central job is optimalPageReplacement which identifies which page should be replaced based on the future usage, the page that is not expected to be needed for the longest time is chosen. If a page is not in memory then it is said to lead to page fault. If memory is full, the algorithm then goes through the current pages in the memory and identifies which page of memory is least likely to be used next or has not been used again), before writing to this newly identified page of memory. The function processTestCase takes input files, extracts the string reference and the frame count while the optimalPageReplacement computes the total page faults. The main routine takes one or more filenames as command line arguments and processes each file, generating the relative string, the total number of frames and total number of page faults for each of the cases. Such an implementation successfully incorporates concepts such as file handling and the optimal page replacement algorithm, which are demonstrated to have excellent comprehensibility and modularity.

LRU Implementation:

This program provides an implementation of the Second Chance Page Replacement Algorithm which is similar to the LRU algorithm, especially through the use of a reference bit. A vector contains the pages that are monitored with Page number and reference bit. Whenever a page is requested, the program looks for the same page in the memory (hit), as well as in the cache. It put its reference bit to 1 if it is found. Inability to find needed data in the cache or buffer memory results in what is called a page fault. In other words, if there is an opportunity in memory, the page is added directly. Otherwise, the algorithm continues round-robin for the circular method the reference bit will determine which page will be replaced. The pages, which have a reference bit of 1, we make them accept another chance by setting its bit to zero and then moving the "pointer" forward, until we get that of a page which refers to a zero bit in order to replace. Input parameters are reference strings and frame count which are entered through command line arguments with names of input files The pro-gram processes each file and calculates the sum of page faults using the given algorithm. The outputs include the reference string, frame count and the total page faults for every file shown as an indication of proper memory management simulation**.**

LFU Implementation:

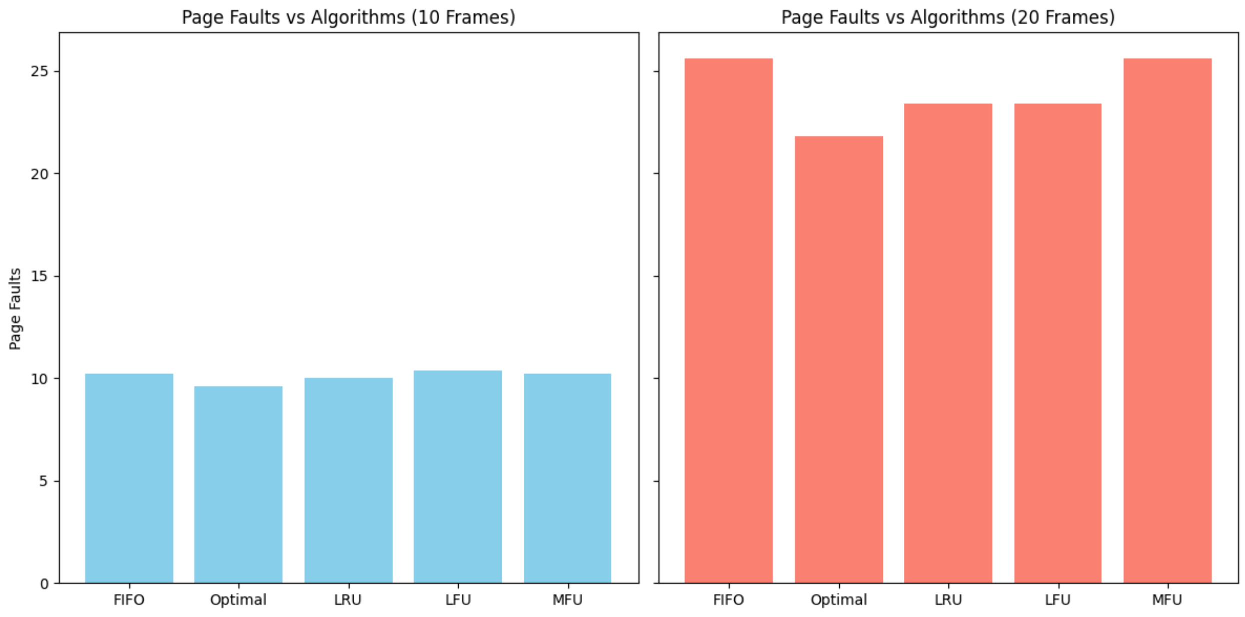
In this program the Least Frequently Used (LFU) Page Replacement Algorithm is applied to mimic the condition of a paging system. An unordered map is employed keeping the frequency of page references and another vector is used to store the currently in memory pages. In each access carried out for the page of a process, the process does not have the page fault if the page is already available in memory. Otherwise a page fault is incurred and if the memory is full, then the algorithm replaces least accessed page based on the frequency map. The function findLeastFrequentlyUsed helps to find the least frequently used page in case of replacement is required. The processTestCase function works with input files which contain a reference string and frame count parameters The function extracts these parameters for calculating total number of page faults using lfuPageReplacement. For each file being processed the result in console consists of the reference string, frame count and the total page faults. The program is aimed to process one or more input files passed to the program via command line argument, which contributes to the modularity of the implemented LFU page replacement.

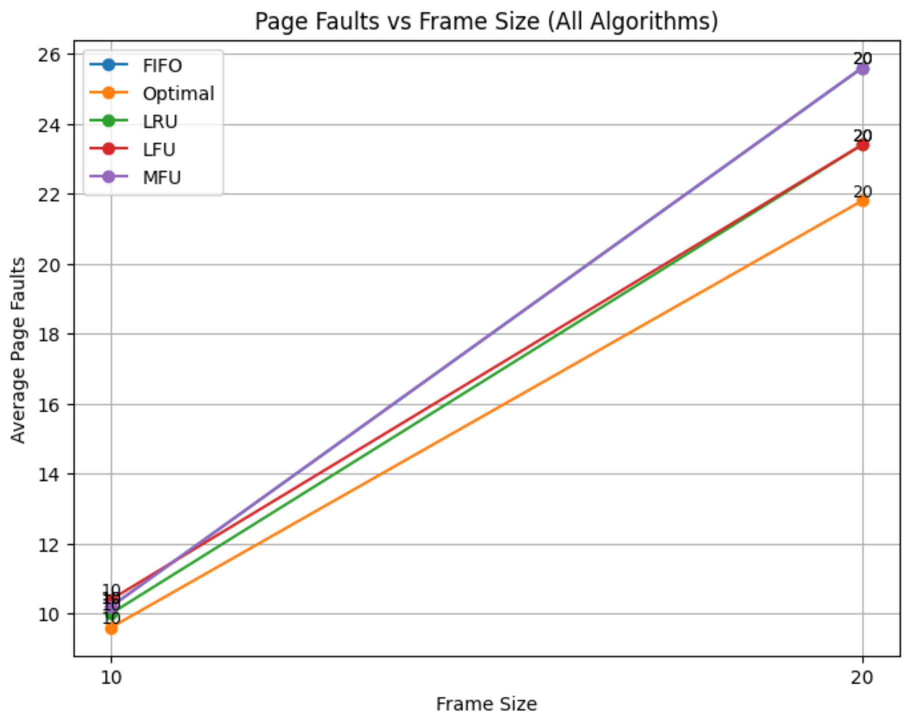
MFU Implementation:

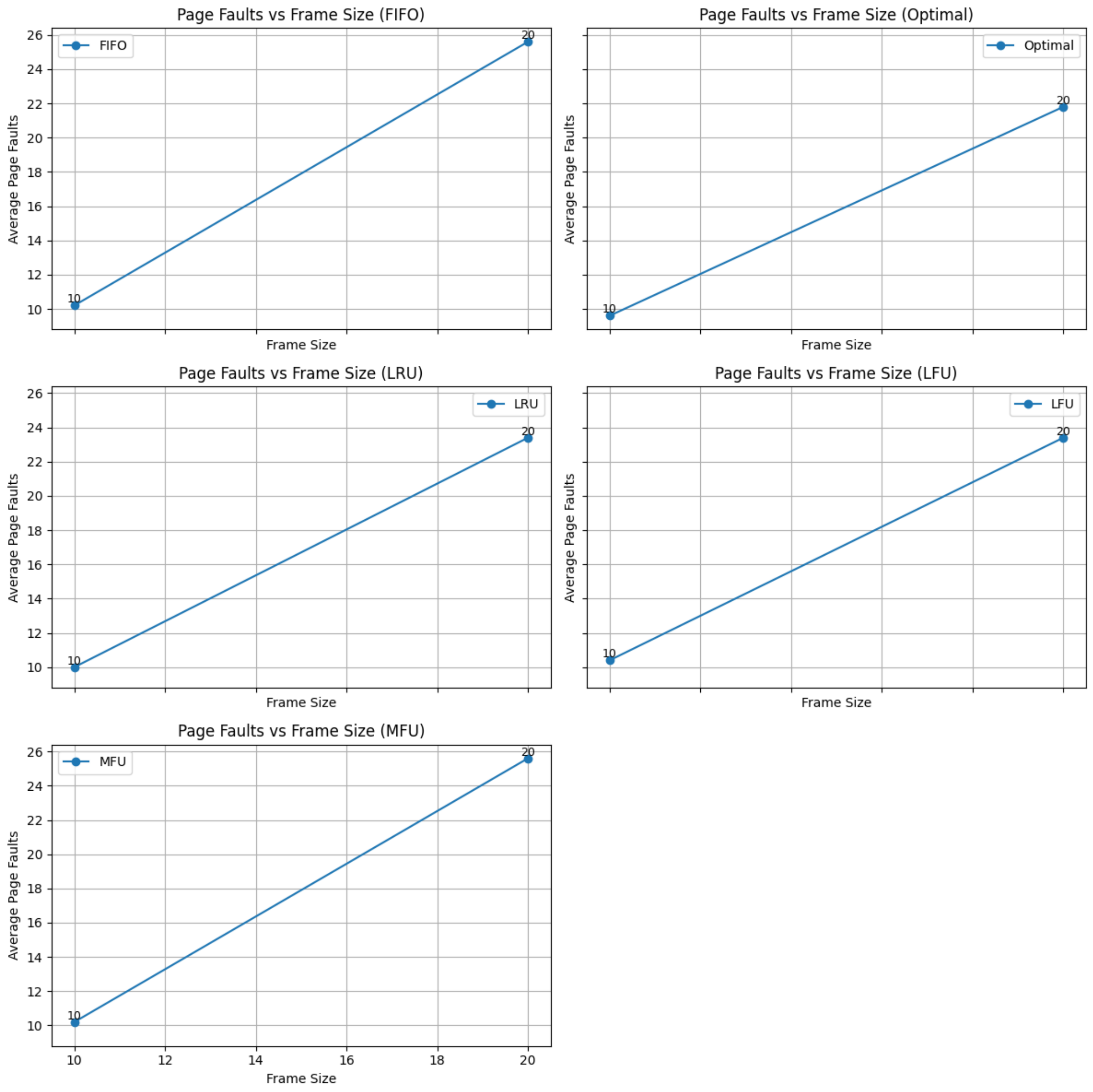
In this program Most Frequently Used (MFU) Page Replacement Algorithm is employed, it will help to simulate a memory management whose replacement strategy focuses most at the most frequently used page. The frequency of which references each page is computed and stored in an unordered map while the current familiar pages in storage are kept in a vector (frame). Another vector, insertOrder, is used to keep record of the order of the pages' insertion in case of a tie that is solved by FIFO (First-In, First-Out). During every page request, the frequency of the page is counted using a counter. It is only when a page is not present in memories that we speak of a page fault. In any other case, a page fault is considered to have occurred. In the case of memory being full, the most frequently accessed page is removed and in the occasion that there is a tie the first page to be introduced (FIFO) is ejected. The findMFU function indicate the MFU page for replacement. A possible example of the program input is a reference string and frame amount retrieved from a source file, the total of page faults, given by most Frequently Used Page Replacement As outputs the program gives the reference string, frame count and the total of page faults. In the modular design, file handling, frequency tracking, and the MFU algorithm are designed with cohesion.

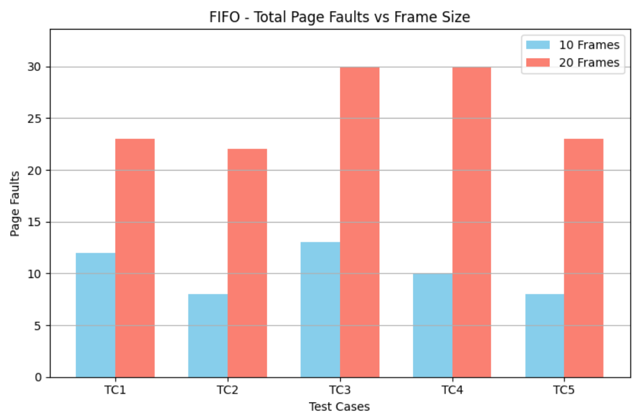
Experimental Results:

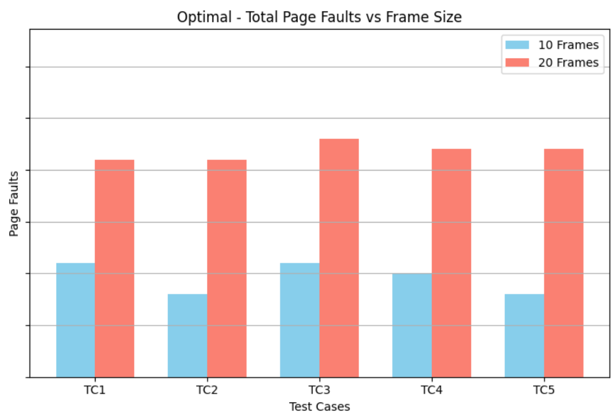
Outputs plotted in graphs:

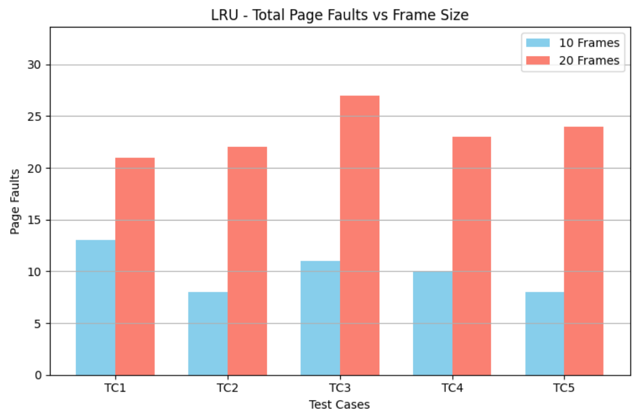


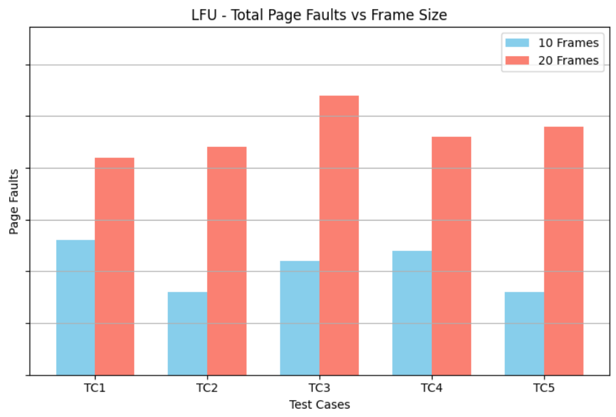


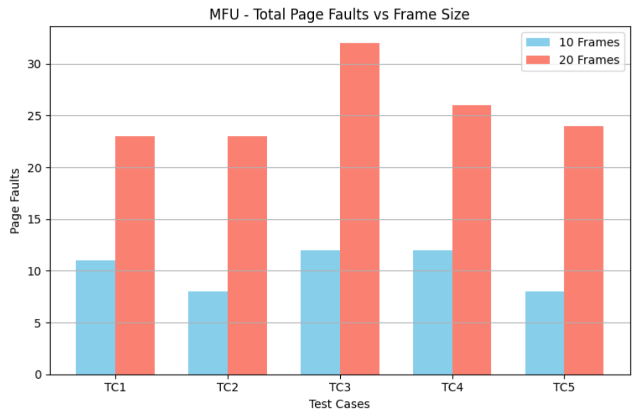

Page Faults vs Frame Size (FIFO)
FIFO
Page Faults vs Frame Size (Optimal)
Optimal
Page Faults vs Frame Size (LRU)
LRU
Page Faults vs Frame Size (LFU)
LFU
Page Faults vs Frame Size (MFU)
MFU
Average Page Faults
Frame Size
10
20
10
12
14
16
18
20
22
24
26

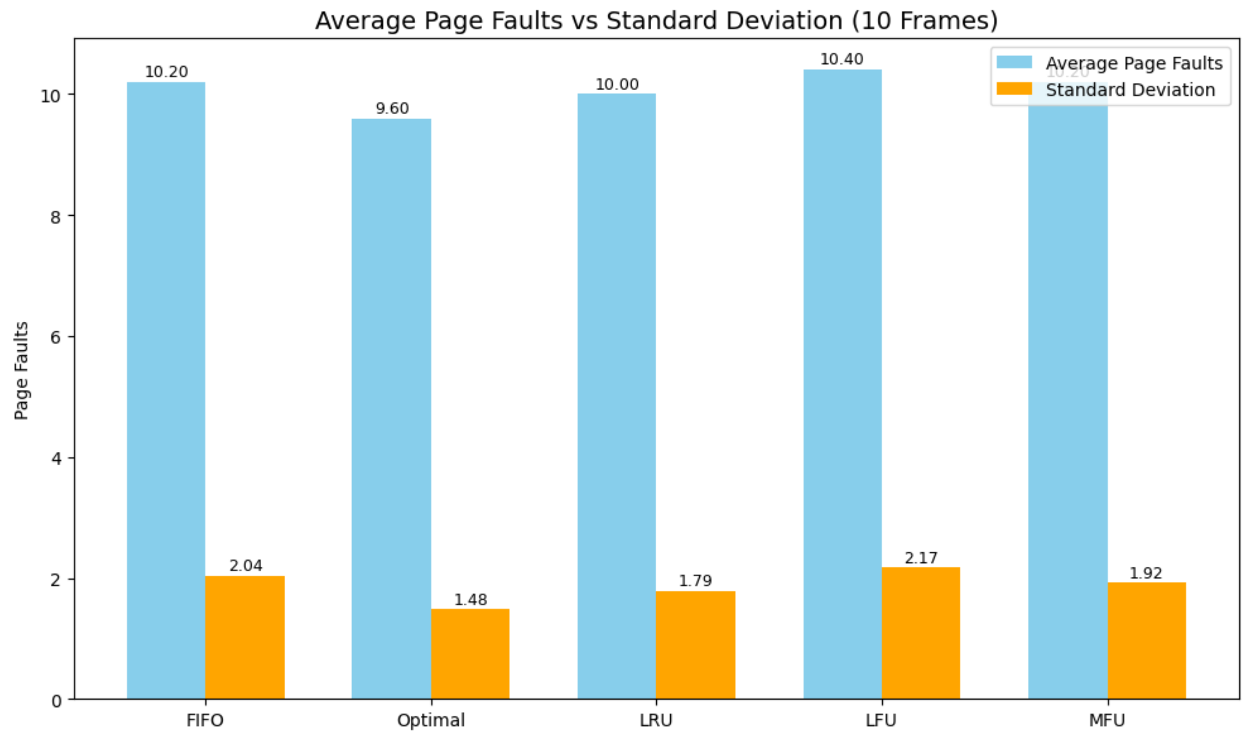

FIFO - Total Page Faults vs Frame Size
10 Frames
20 Frames
Page Faults
Test Cases
TC1
TC2
TC3
TC4
TC5


Optimal - Total Page Faults vs Frame Size
10 Frames
20 Frames
Page Faults
Test Cases
TC1
TC2
TC3
TC4
TC5


LRU - Total Page Faults vs Frame Size
10 Frames
20 Frames
Page Faults
Test Cases
TC1
TC2
TC3
TC4
TC5


LFU - Total Page Faults vs Frame Size
10 Frames
20 Frames
Page Faults
Test Cases
TC1
TC2
TC3
TC4
TC5


MFU - Total Page Faults vs Frame Size
10 Frames
20 Frames
Page Faults
Test Cases
TC1
TC2
TC3
TC4
TC5



Tools used to generate graphs:

a. We have used matplotlib to generate graphs.
b. We have used Numpy to generate graphs.
c. We have used Pandas to generate graphs.

Insights to be gained from graphs:

Algorithm	5 Test cases with Frame Size 10, Avg weight	5 Test cases with Frame Size 20, Avg weight
FIFO	**10.2**	**25.6**
Optimal	**9.6**	**21.8**
LRU	**10.0**	**23.4**
LFU	**10.4**	**23.4**
MFU	**10.2**	**25.6**

Results Overview:

Data Interpretation:

Conclusion:

This project was able to assimilate and demonstrate the following page replacement techniques – FIFO, Optimal, LRU, Second Chances, LFU, and MFU. The evaluation emphasized the ideas of system simplicity and cost, with Optimal described as the ideal type. The results underscore the fact that choosing the right algorithms for specific workload is the key determinant of performance. The present work offers basic knowledge on page replacement techniques in managing memories.

Accomplishments:

Implemented six page replacement algorithms with logical and concise approach. Carried out a thorough experimentation on different specified reference strings and frame dimensions. Provided comparative outcomes generated in form of graphs and statistical analysis. Discussed cases that involve qualitative performance of the certain algorithms. Delivered all the action plan regarding implementation, outcome of Result and Key observation in a formal written report.

Future Work:

Study more different algorithms, such as Clock Replacement, and different types of Hybrid algorithms. Study potential use-cases, where the number and type of active transactions are not fixed. Improve when and how to implement in order to limit data tracking processes. Interleave the properties of a more accurate causality in logics, involving ‘reference bits’ for simpler hardware enforcing it. Expand the above research to also encompass distributed systems and memory virtualization environments.

References:

O'neil, E. J., O'Neil, P. E., & Weikum, G. (1999). An optimality proof of the LRU-K page replacement algorithm. Journal of the ACM (JACM), 46(1), 92-112.