**National University of Computer & Emerging Sciences Karachi Campus**



CS2006: Operating System

*Project Report*

***Page Replacement Algorithms***

Group Members:

1. Muhammad Saad 21K-3267
2. Syed Muhammad Taqi Baquer 21K-3175

***Introduction***

Our project implements the concept of memory management through Paging. We have programmed different Page Replacement Algorithms such as FIFO, LRU and Optimal to generate their detailed working and their Page Fault and Page Hit results of each. These algorithms are evaluated by running them on a particular string of memory reference and computing the number of page faults and hits. Moreover, we have compared the results of each algorithm by generating a graph of page hit and miss results to evaluate the most efficient algorithm.

***Background Context***

The memory management service provided by the operating system holds immense importance and role in efficient working of our systems. It is the basic function that manages primary memory; the operating system moves processes backforth between main and secondary memory.

Operating systems use several techniques for memory management to ensure smooth running of our systems; to keep track of used memory space by processes, minimize fragmentation issues, ensure proper utilization of main memory and maintain data integrity while process execution.

Paging is a technique used by operating systems for memory management. There are different techniques that determine how a particular page is selected to be swapped out upon page fault, that is, to create space for a new page. These techniques are called page replacement algorithms. Numerous page replacement algorithms exist such as Optimal Page Replacement, First in First out, Least Recently Used, Least Frequently Used, Most Frequently Used, Not Frequently Used, Second Chance Page Replacement, Clock Page Replacement etc.

Considering the rapid growth in research and invention of new techniques for memory management and its countless benefits, we feel memory management is essential for every system, and, so, we were motivated to conduct our project on the **Page Replacement Algorithms**. In this project, we have fulfilled our objective by implementing three algorithms i.e. Optimal Page Replacement, First in First out and Least Recently Used and have learned about the core of memory management.

***Problem Description***

A computer operating system that uses paging for virtual

memory management, **page replacement algorithms** decide which memory pages to swap out, or write to disk, when a page of memory needs to be allocated. Page replacement happens when a requested page is not in memory (page fault) and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.

When the page that was selected for replacement and paged out is referenced again it has to be paged in (read in from disk), and this involves waiting for I/O completion. This determines the *quality* of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm. A page replacement algorithm looks at the limited information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimize the total number of page misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself.

***METHODS***

Description of the algorithms we have implemented:

FIFO Page Replacement Algorithm

In this the page is replaced which is in memory for the greatest period of time.

LRU Page Replacement Algorithm:

Replaces the page in memory that has not been used for the longest period of time

Optimal Page Replacement Algorithm:

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

***Tools (Programming Language, Systems***

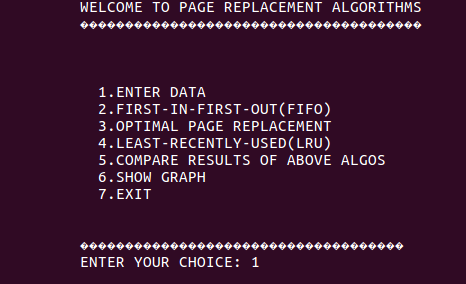
1. C language
2. Ubuntu
3. System is 64 bit
4. Libraries: #include <stdio.h> #include <stdlib.h>

***Solution Design***

**Main Menu:**

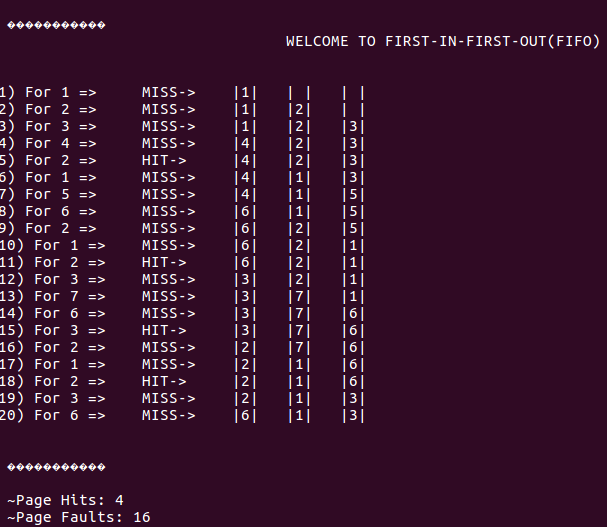
This is the first screen of our program which displays different menu options for the user to choose from. First and foremost, the user is supposed to enter the size of the Page Frame and the Page Reference String length and then the whole string one by one. After that the user can choose from the three Page Replacement

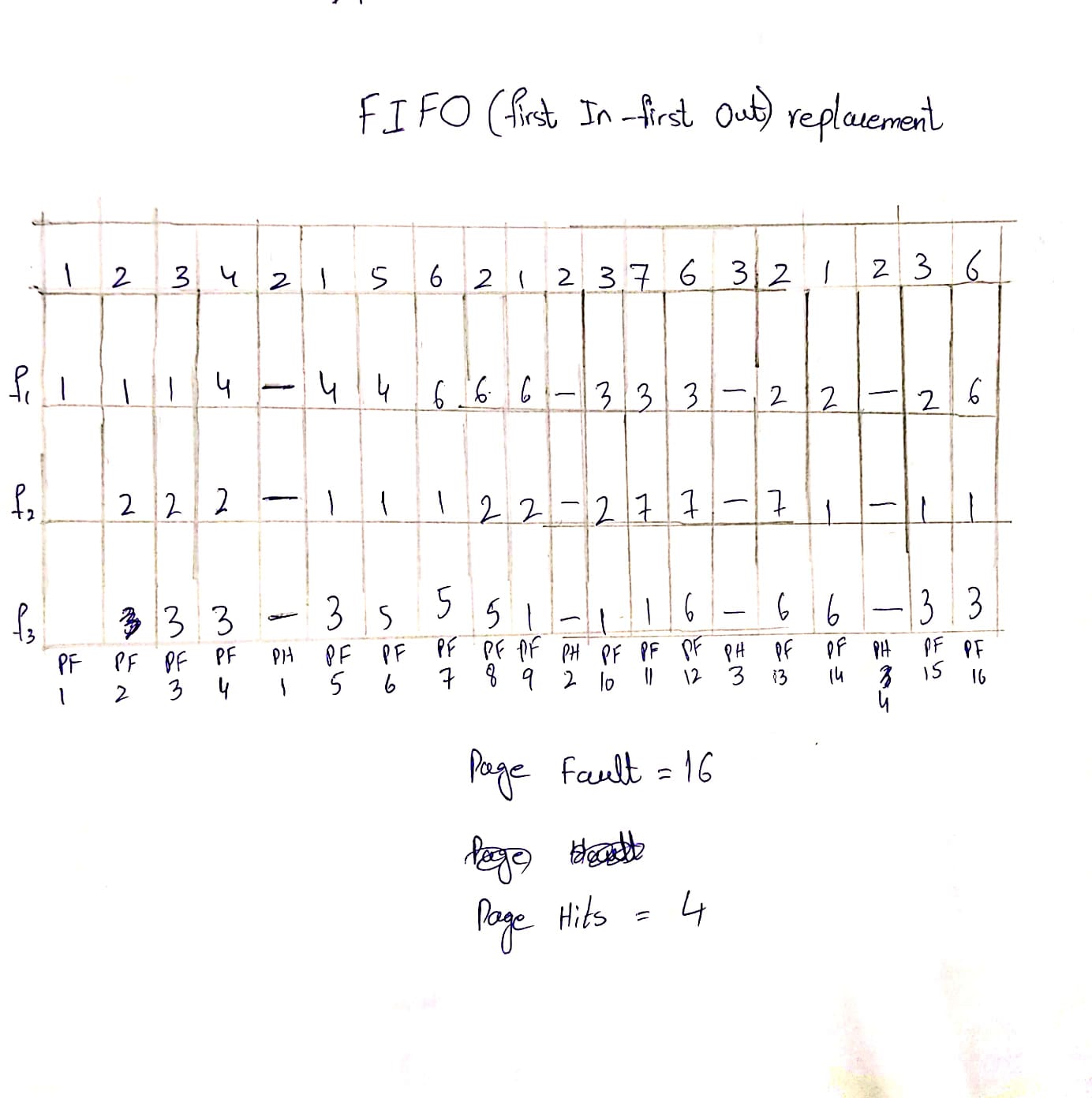
Algorithms and then evaluate each paging method by choosing the compare and/or graph option. The user also has the option to exit the program.



**FIFO Algorithm:**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a frame, the oldest page is in the front of the frame. When a page needs to be replaced, the page in the front of the frame is selected for removal.

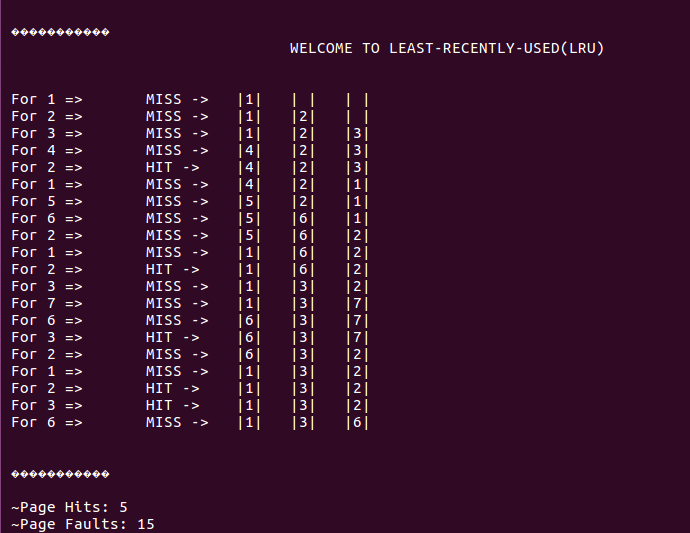


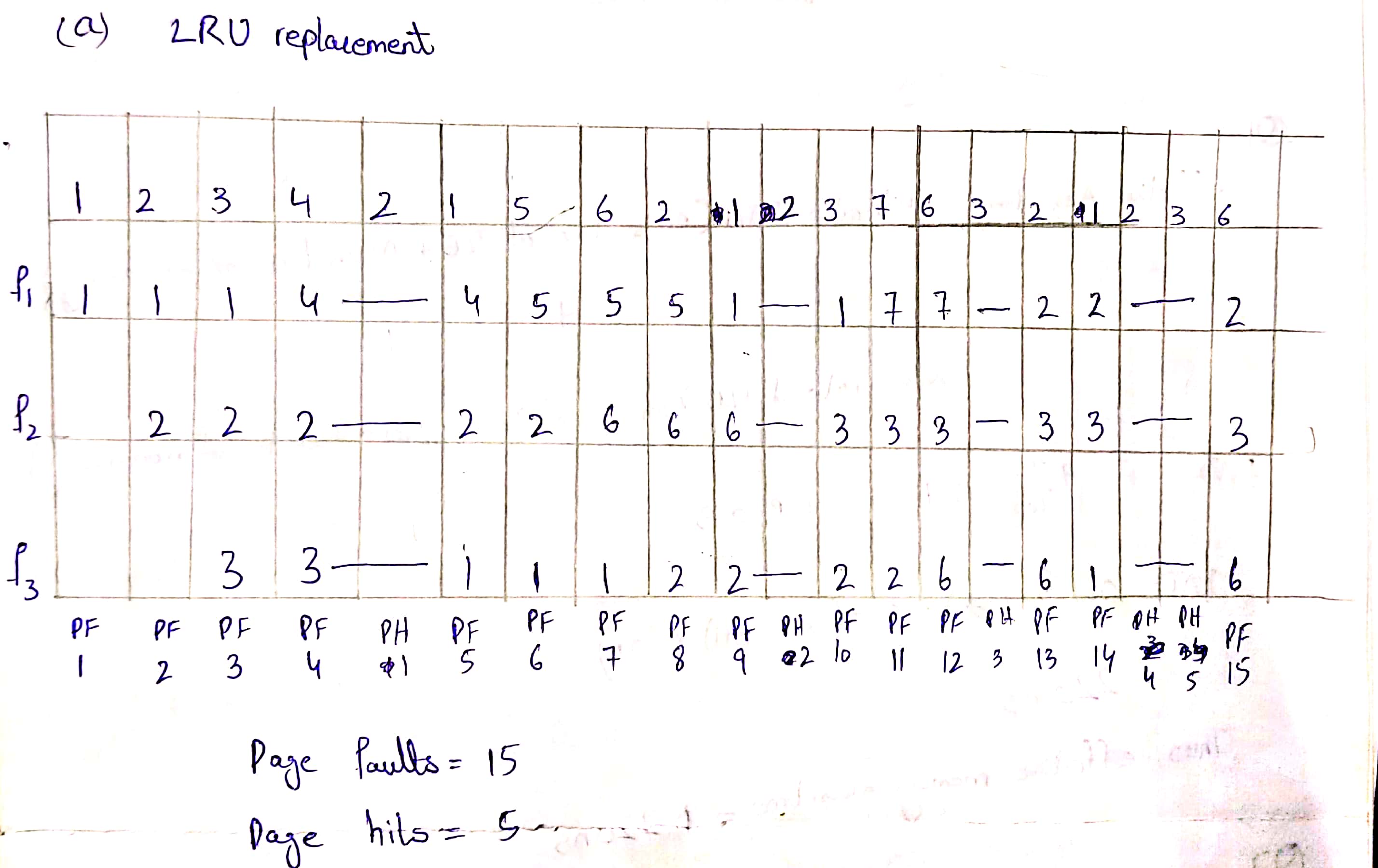


We have implemented this in our code by taking a string as an input. The page frame has been implemented using an array queue. Our program will then check the frames which are 3 in this particular example and then it will check whether they are in the frame already or not and then make an increment in page hit and page fault accordingly.

**LRU (LEAST RECENTLY USED):**

Least Recently Used (LRU) algorithm is a page replacement technique used for memory management. According to this method, the page which is least recently used is replaced. Therefore, in memory, any page that has been unused for a longer period of time than the others is replaced.

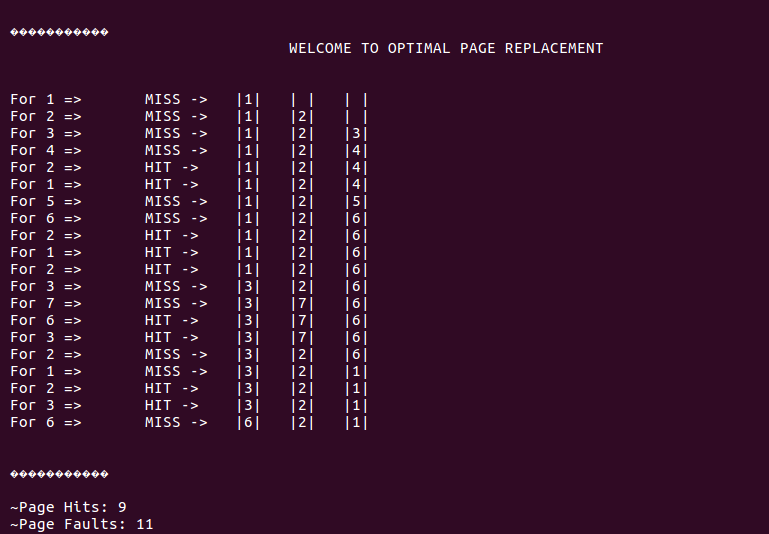


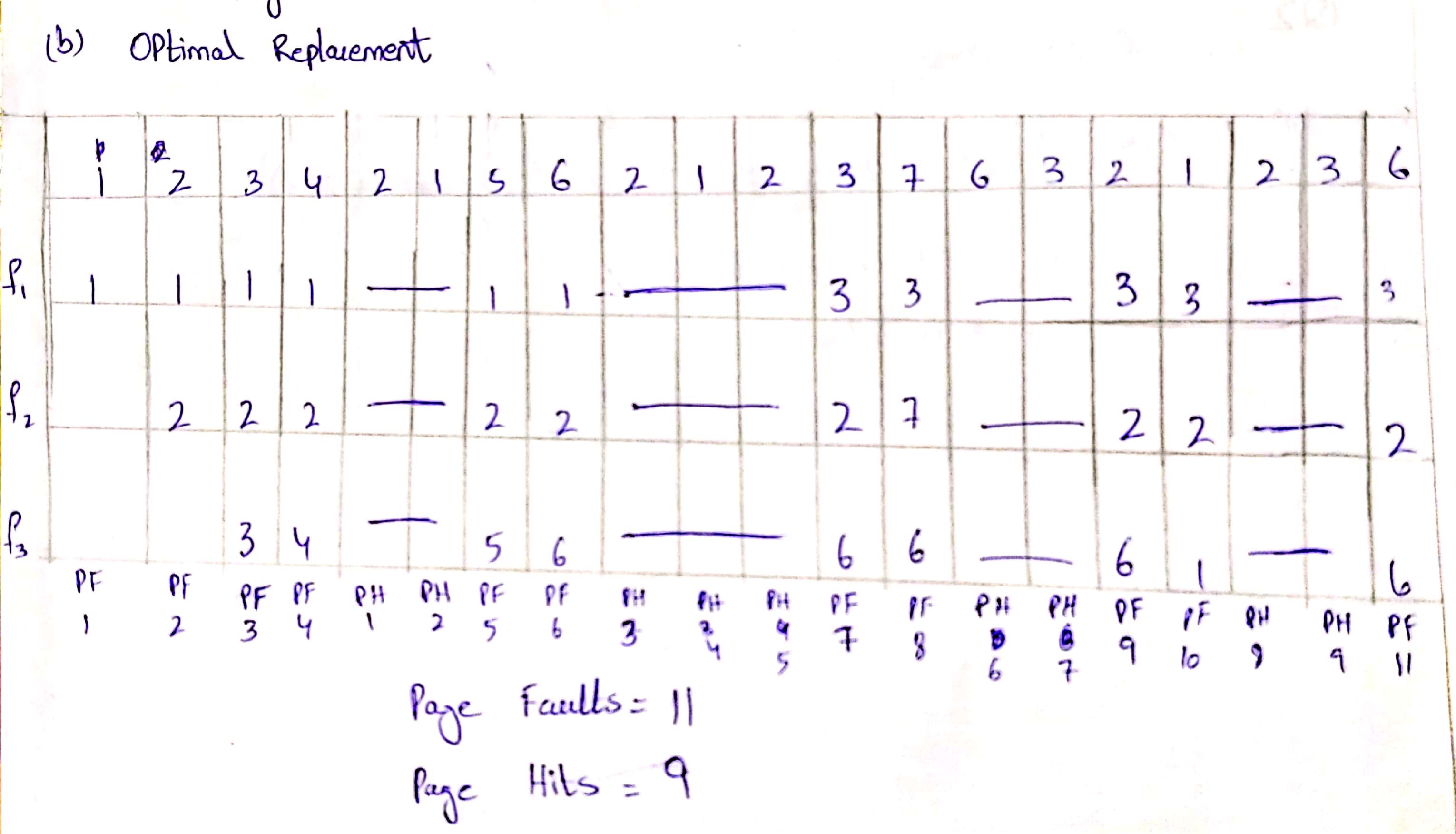


We have implemented LRU in our code by checking the value through the previous hit to see if it is far from the current value, if that is true then it will replace, otherwise it would not replace and then incrementing page fault and page hit accordingly.

# OPTIMAL:

The theoretically optimal page replacement algorithm (also known as OPT, clairvoyant replacement algorithm, or Belay’s optimal page replacement policy) is one of the most complex algorithms out of the ones implemented by us. It works as follows: when a page needs to be swapped in, the operating system swaps out the page whose next use will occur farthest in the future.



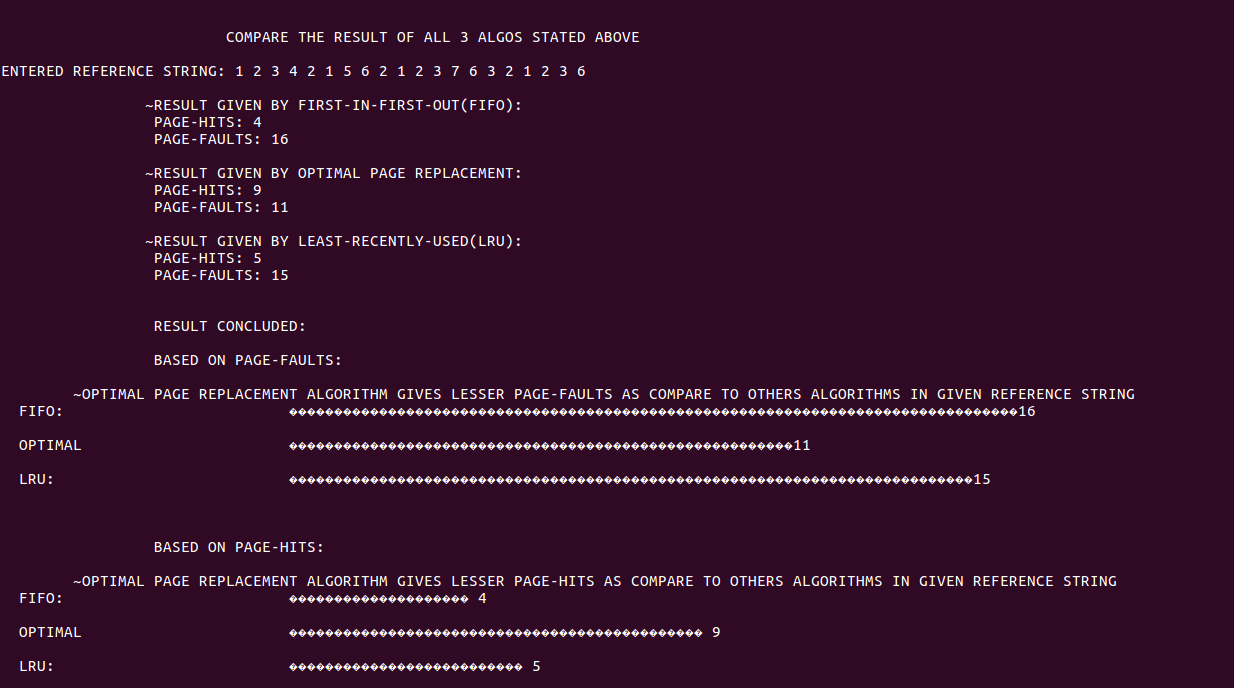


Although this algorithm is comparatively complex, we have coded it using the logic of checking whether the next variables which are in the frames are going to occur soon or later by checking the reference string that we have stored in an array against the pages in the frame and then resulting in incrementing options of page hit and page fault accordingly.

# Compare function:

It is a very important function of our program as it forms the basis of our evaluation of these paging algorithms; it displays the algorithm with the highest page fault and the one with the lowest page fault and as well as the algorithms with the highest and lowest page hits.

# Graph:



Our program also has the option for evaluating the most efficient algorithm through displaying a bar chart. The bars represent the number of page faults and page hits in each algorithm to allow the user to differentiate and result in an easier visualization.

***Conclusion***

The results provided by our project: the implementation of Page Replacement Algorithms suggest that OPT is the most efficient page replacement algorithm because it is easy to implement, and can be implemented using simple data structures. However, it requires future knowledge of the program and is fairly time consuming.

Our program also shows that FIFO is easy to implement and has a low overhead, however, it is not the most efficient out of these three because it doesn’t consider the frequency of use or last used time, simply replaces the oldest page and it suffers from Belady’s Anomaly(i.e. more page faults when we increase the number of page frames). Moreover, LRU is also efficient as it does not suffer from Belady’s Anomaly but is fairly complex to implement.

Therefore, we can say that we have learned immensely from this project and consider it an important learning experience.