Report Laboratory - 8

- 200010024 (Karthik K)

Introduction

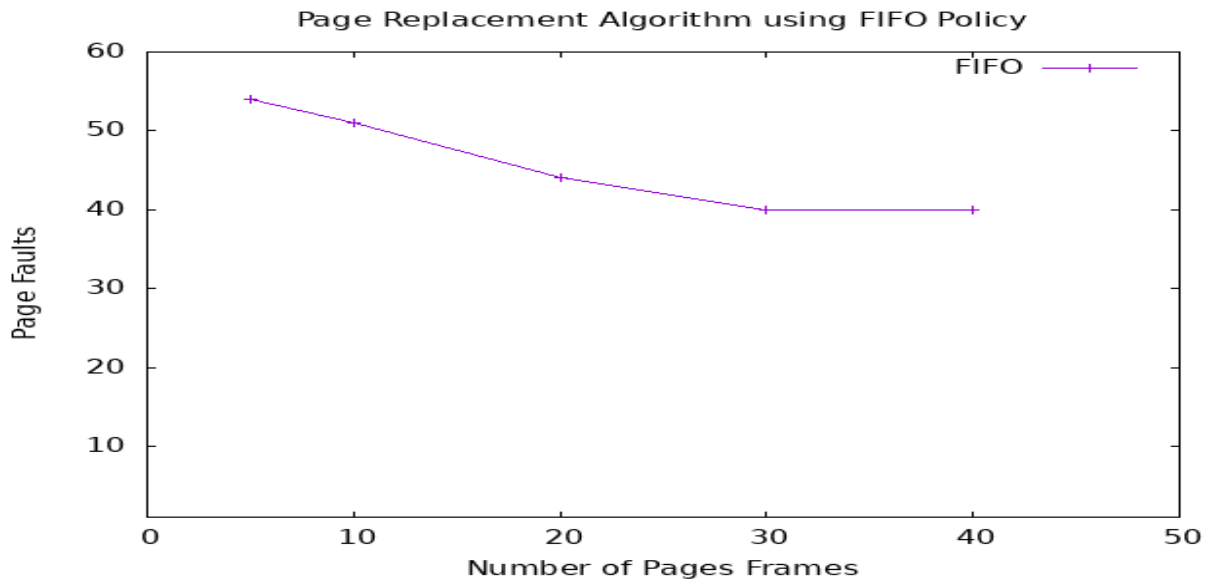
The goal of this assignment was to implement three different page replacement algorithms: FIFO (First In, First-Out), LRU (Least Recently Used), and Random Algorithm. These algorithms are used in operating systems to manage memory when there are more logical pages than physical frames available. The page replacement algorithms decide which pages to remove from physical memory to make space for new pages that are needed. The assignment required implementing these algorithms in C++ and testing them with different configurations (page sizes and frame sizes) and page request patterns. The input files for the program were provided, and they contained the page request patterns for testing. The implemented code reads the command line arguments for the number of pages, frames, and swap size, and the name of the file containing the page request pattern. The program then reads the page request pattern and applies each of the three-page replacement algorithms to it.

FIFO Page Replacement Algorithm

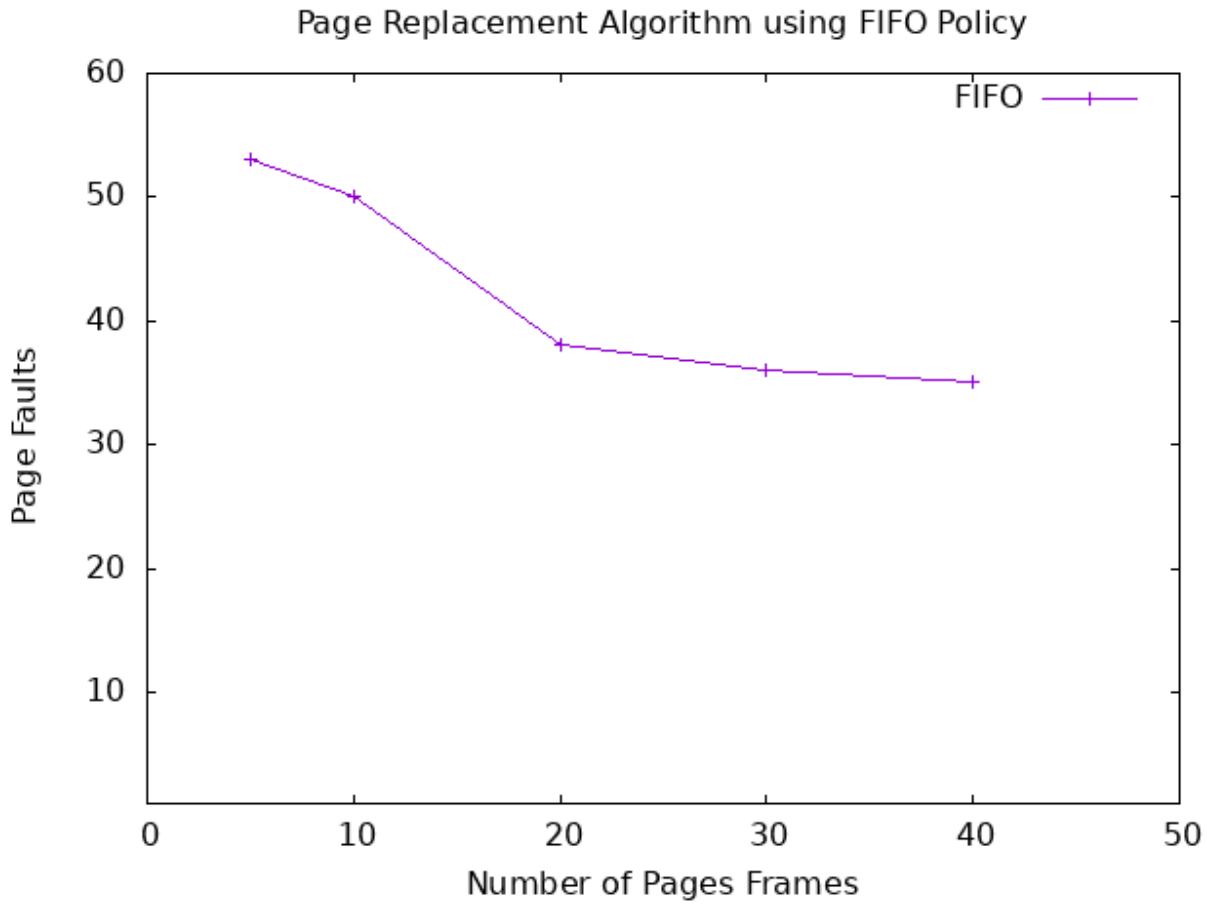
The FIFO algorithm works by replacing the page that arrived first into the frames, without considering how frequently it has been accessed. The implementation of FIFO in the code uses a queue to store the frames and checks if the requested page is already in the queue. If not, it adds it to the end of the queue, and if the queue size reaches the number of frames, it removes the first element from the front of the queue.

I have used 5 request.dat files with 5 different configurations.

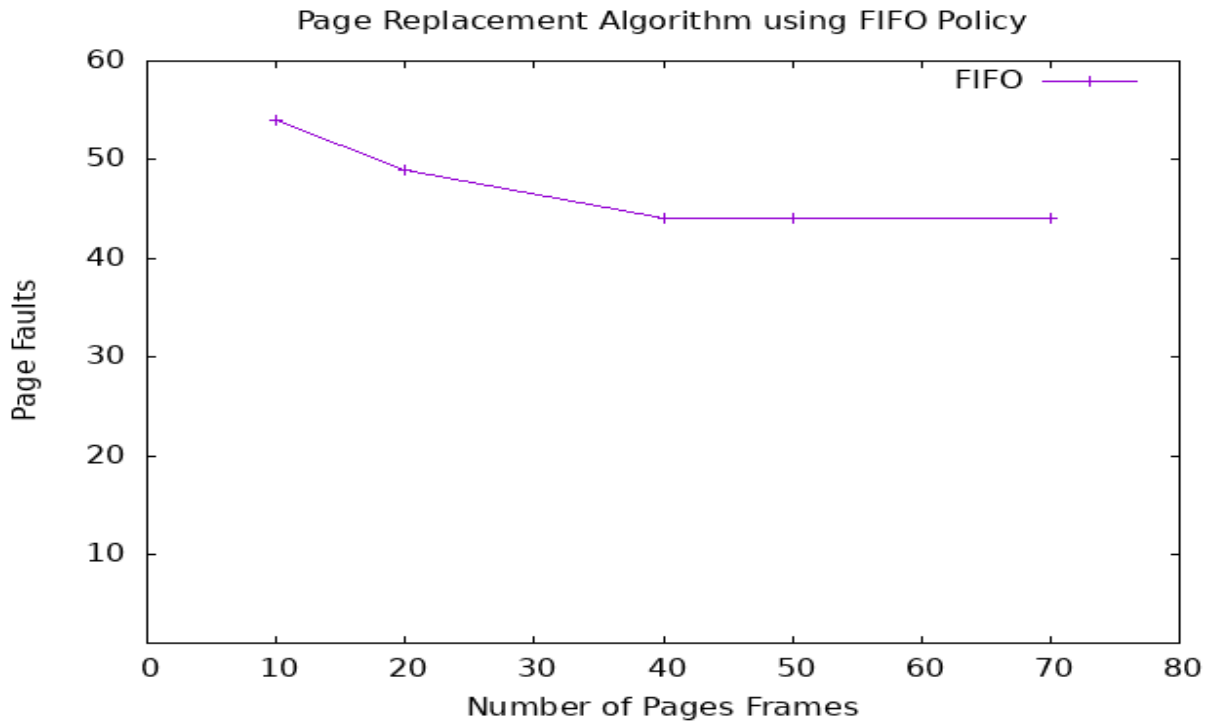
req1.dat



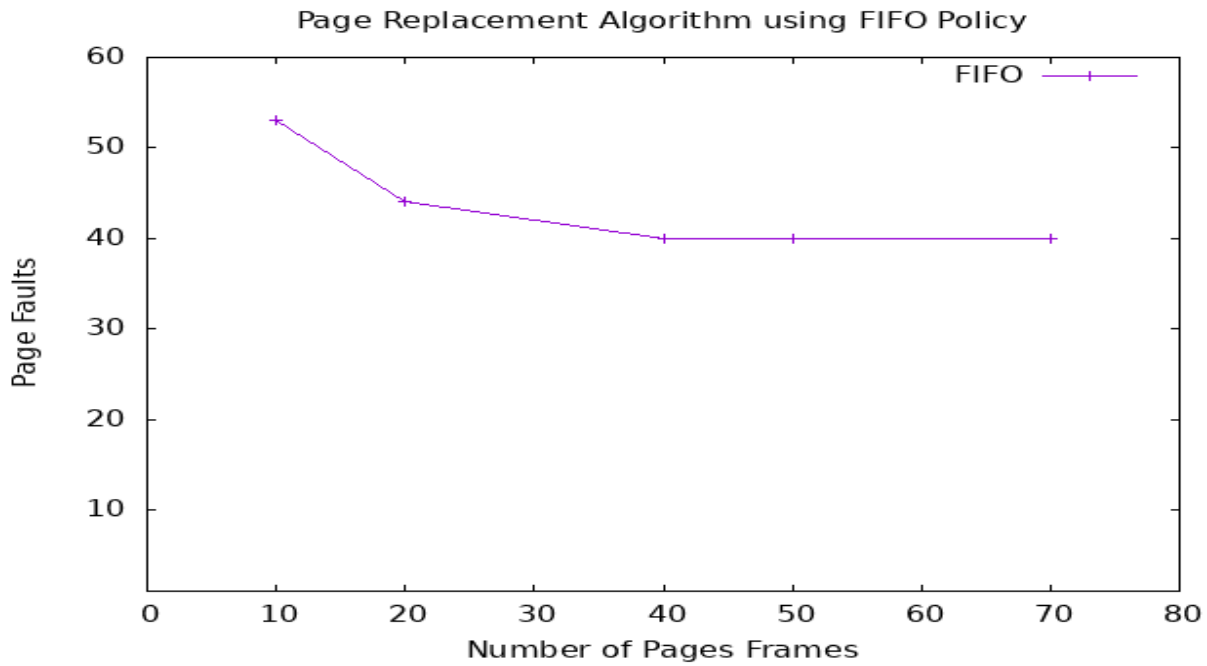
req2.dat



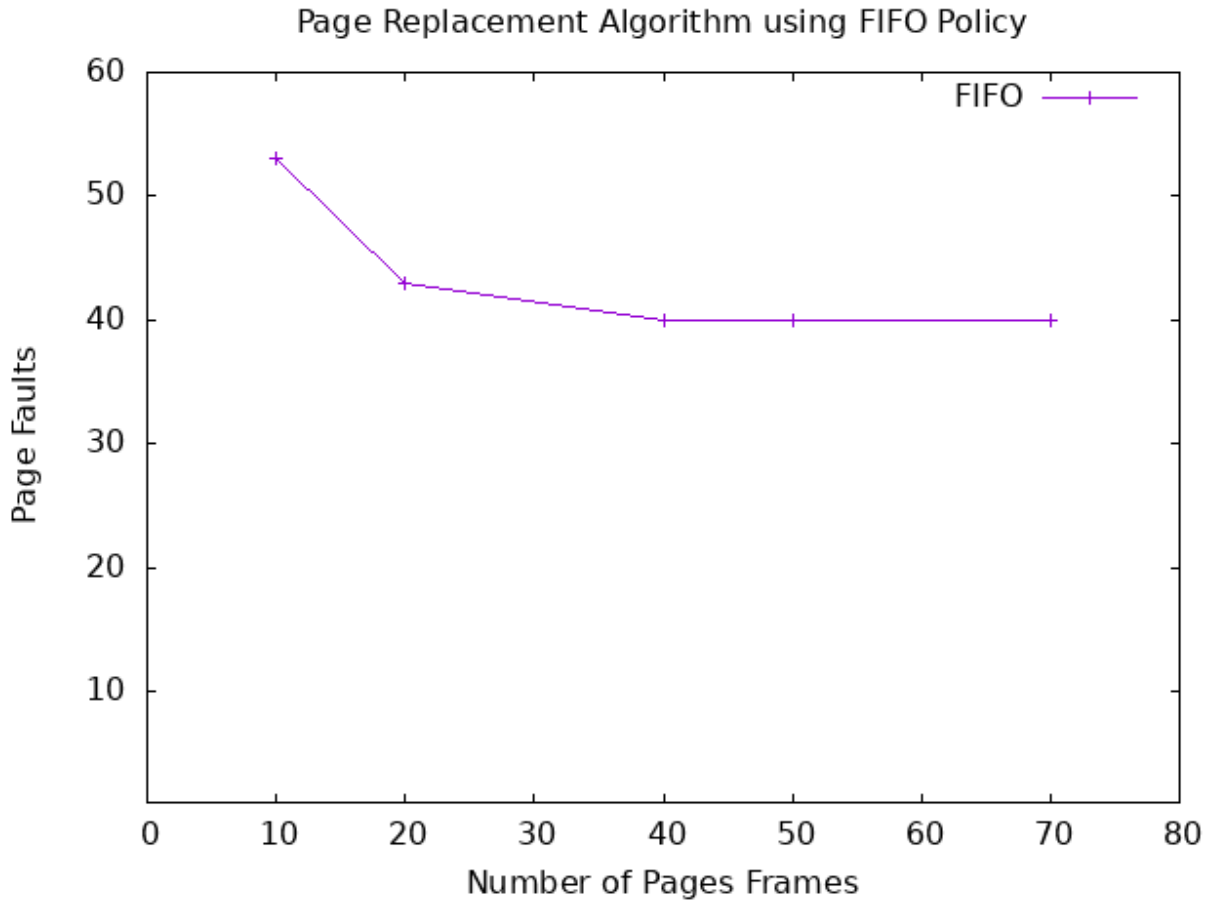
Req3.dat



Req4.dat



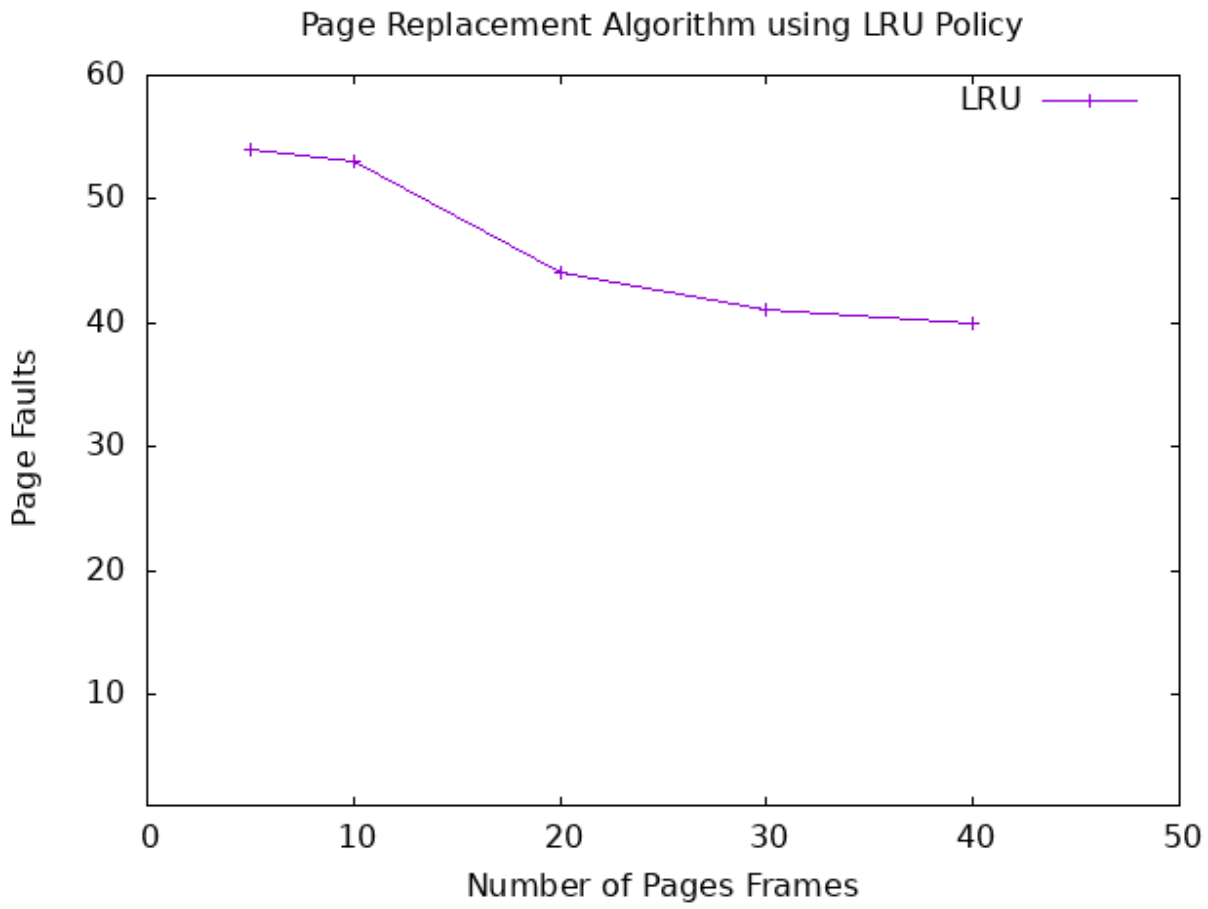
req5.dat


Page Replacement Algorithm using FIFO Policy
FIFO
Page Faults
Number of Pages Frames
60
50
40
30
20
10
0
10
20
30
40
50
60
70
80

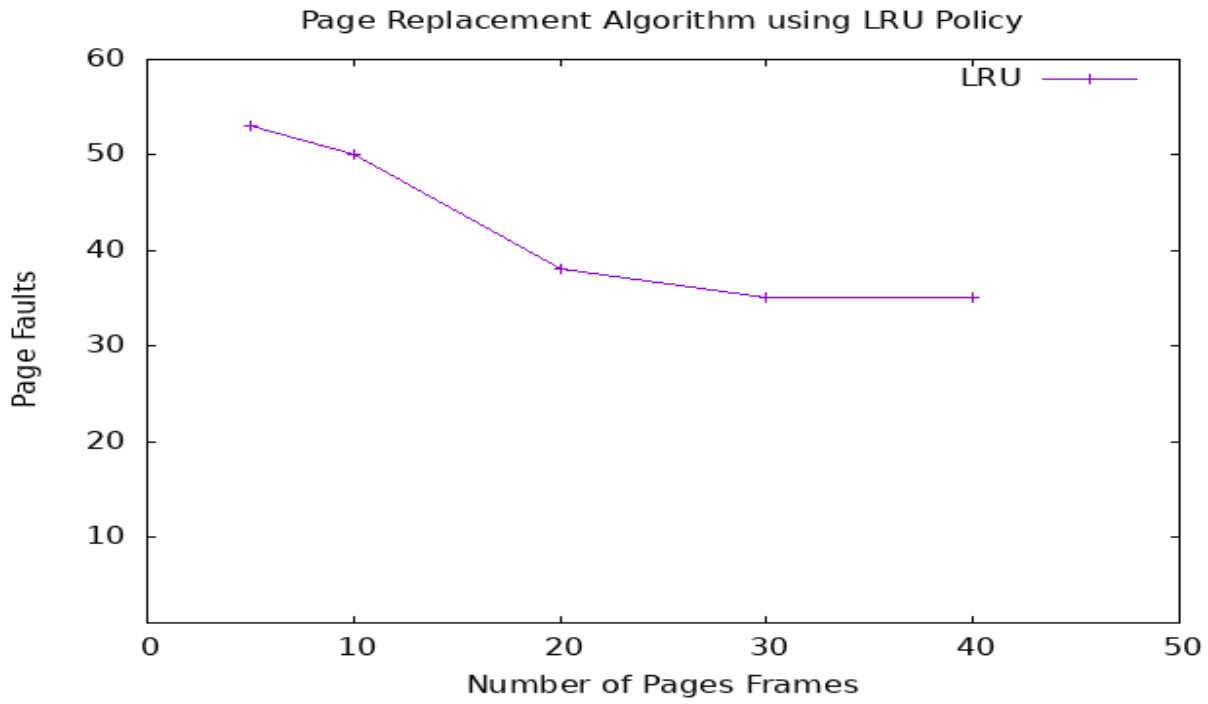
LRU Page Replacement Algorithm

The LRU algorithm works by replacing the page that has not been accessed for the longest time. The implementation of LRU in the code uses a vector to store the frames and checks if the requested page is already in the vector. If not, it adds it to the vector, and if the vector size reaches the number of frames, it removes the first element. If the page is already in the vector, it is moved to the end to indicate that it has been accessed most recently.

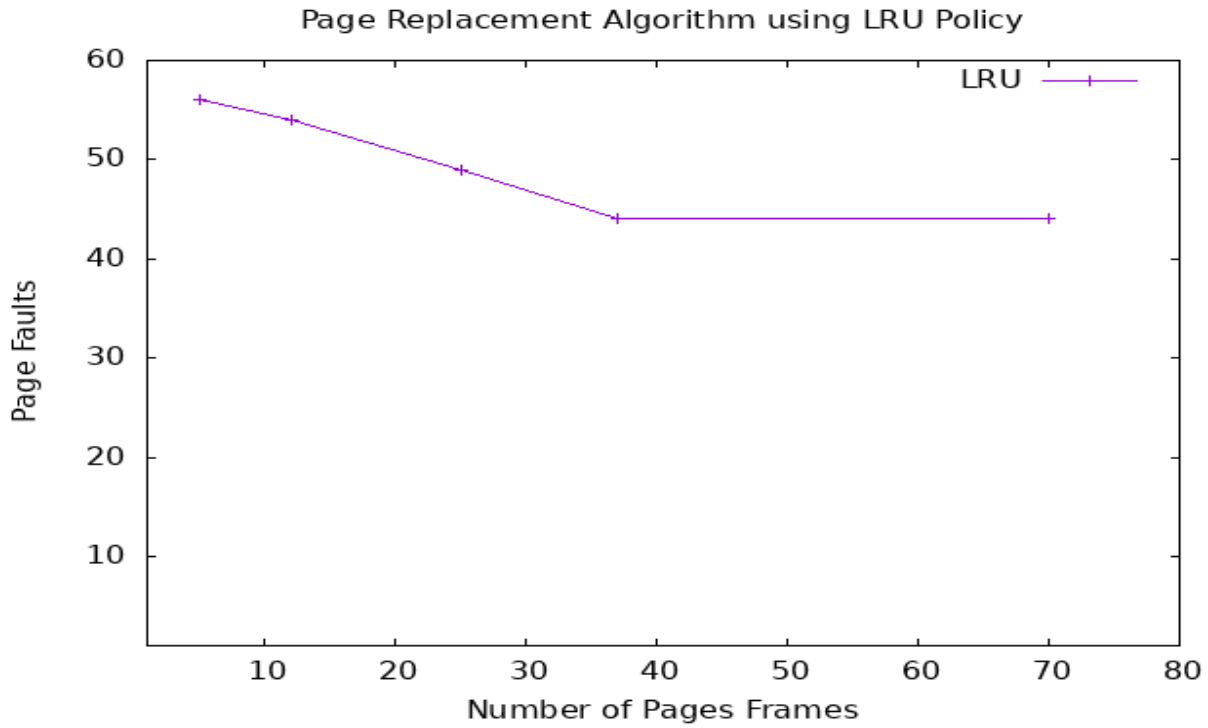
Req1.dat



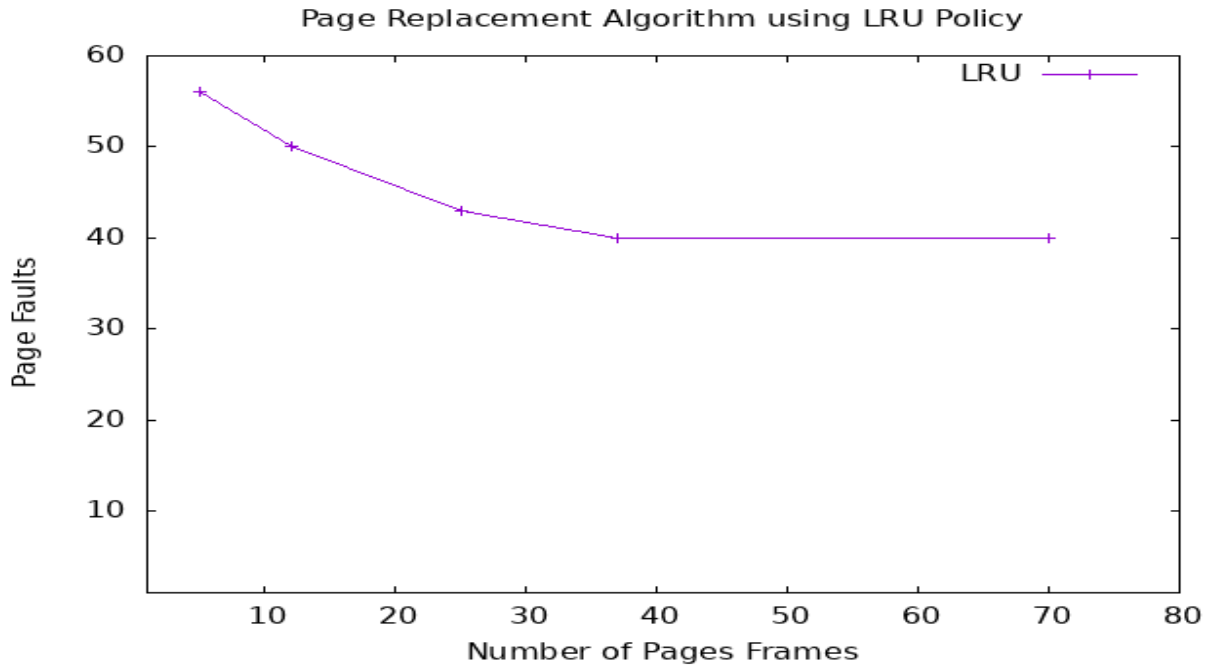
Req2.dat



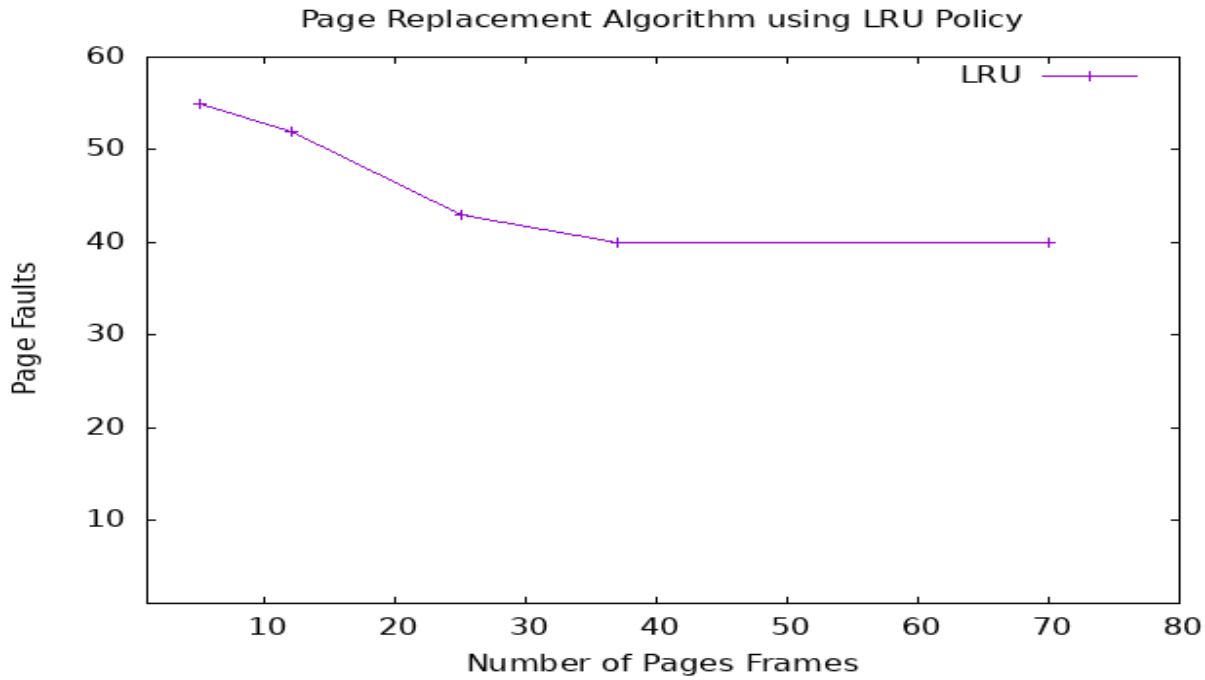
Req3.dat



Req4.dat



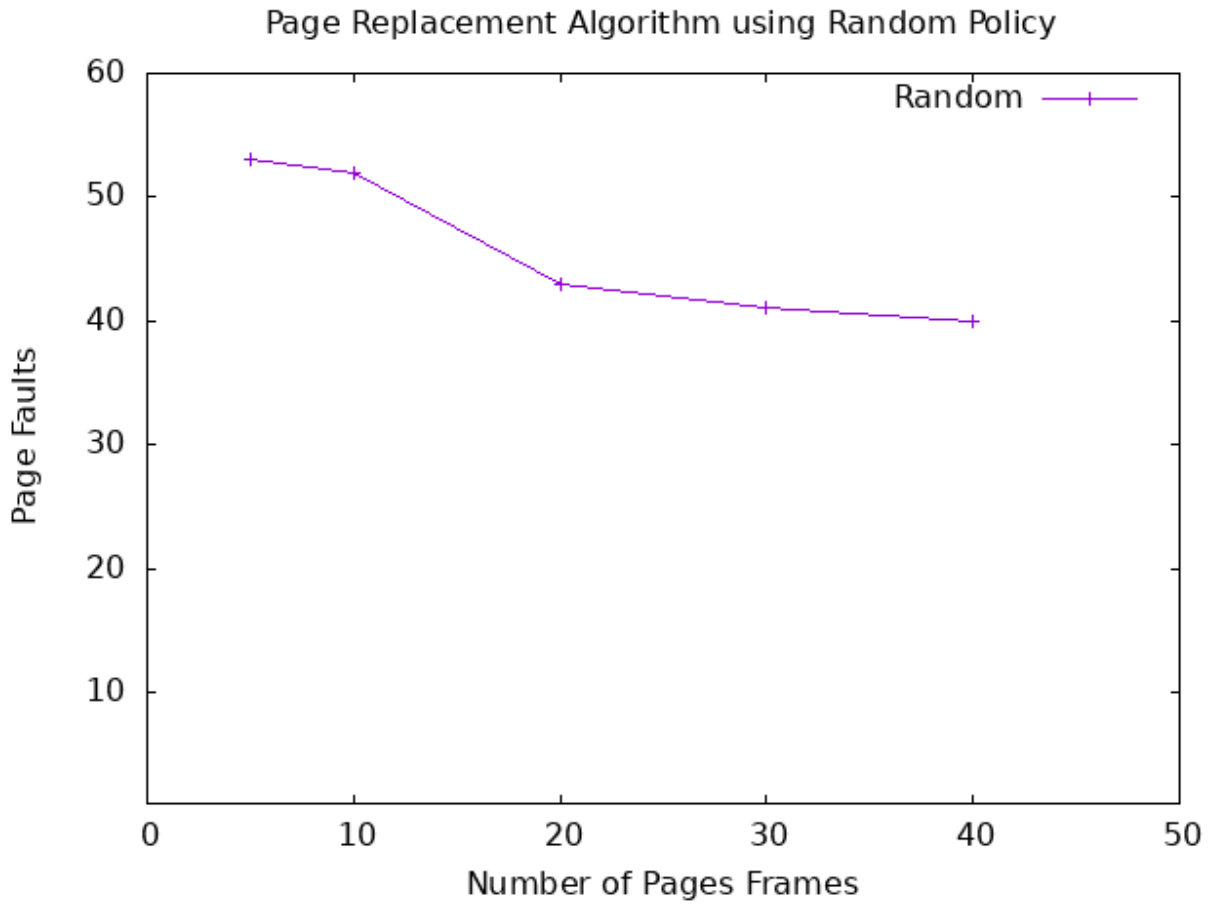
Req5.dat



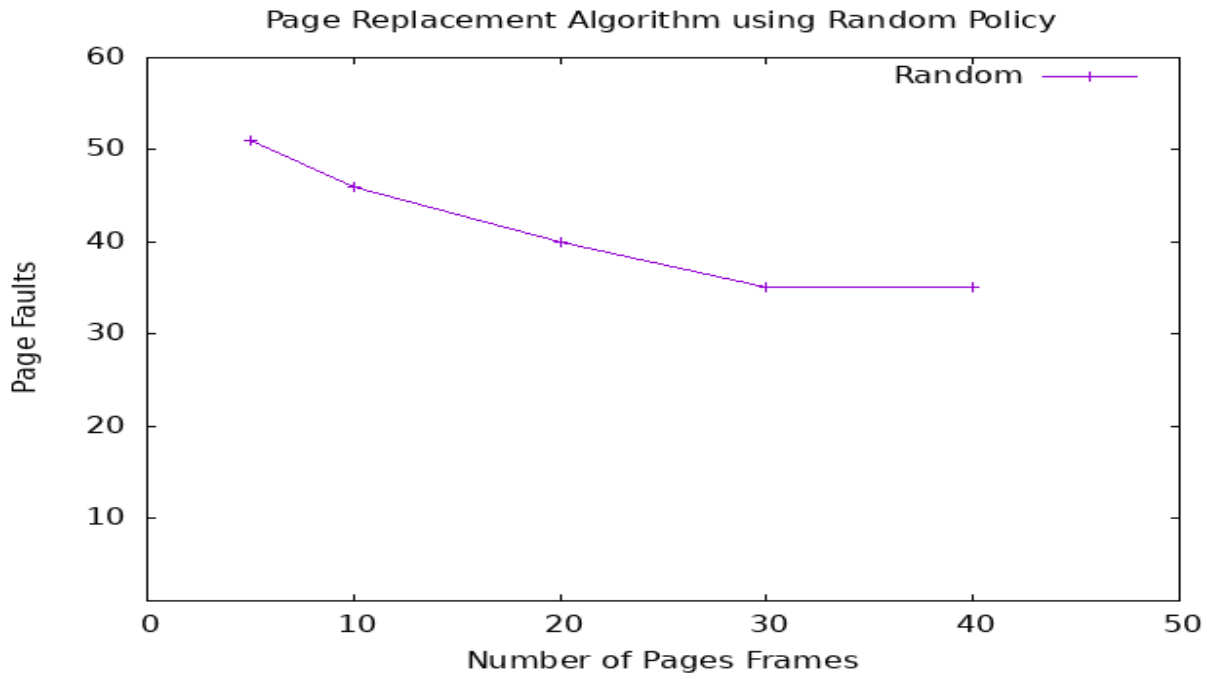
Random Page Replacement Algoithm

The Random algorithm works by randomly selecting a page to replace. The implementation of Random in the code uses a vector to store the frames and checks if the requested page is already in the vector. If not, it adds it to the vector, and if the vector size reaches the number of frames, it selects a random page from the vector to remove.

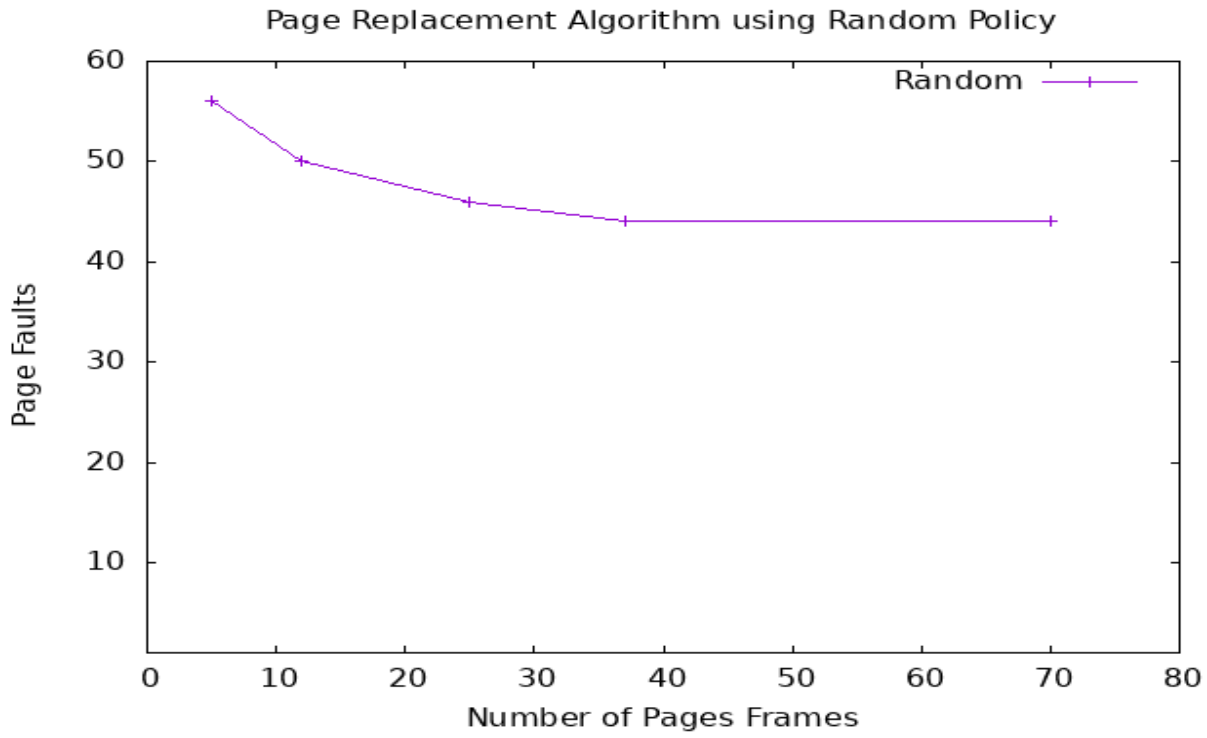
Req1.dat



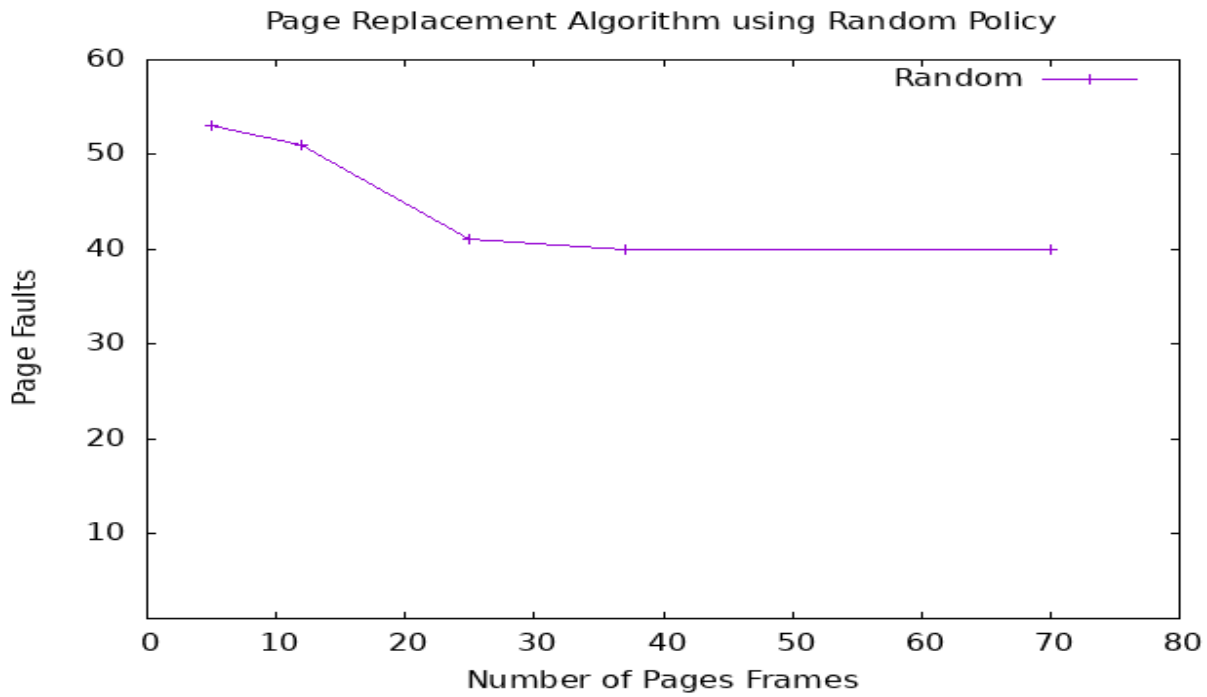
Req2.dat



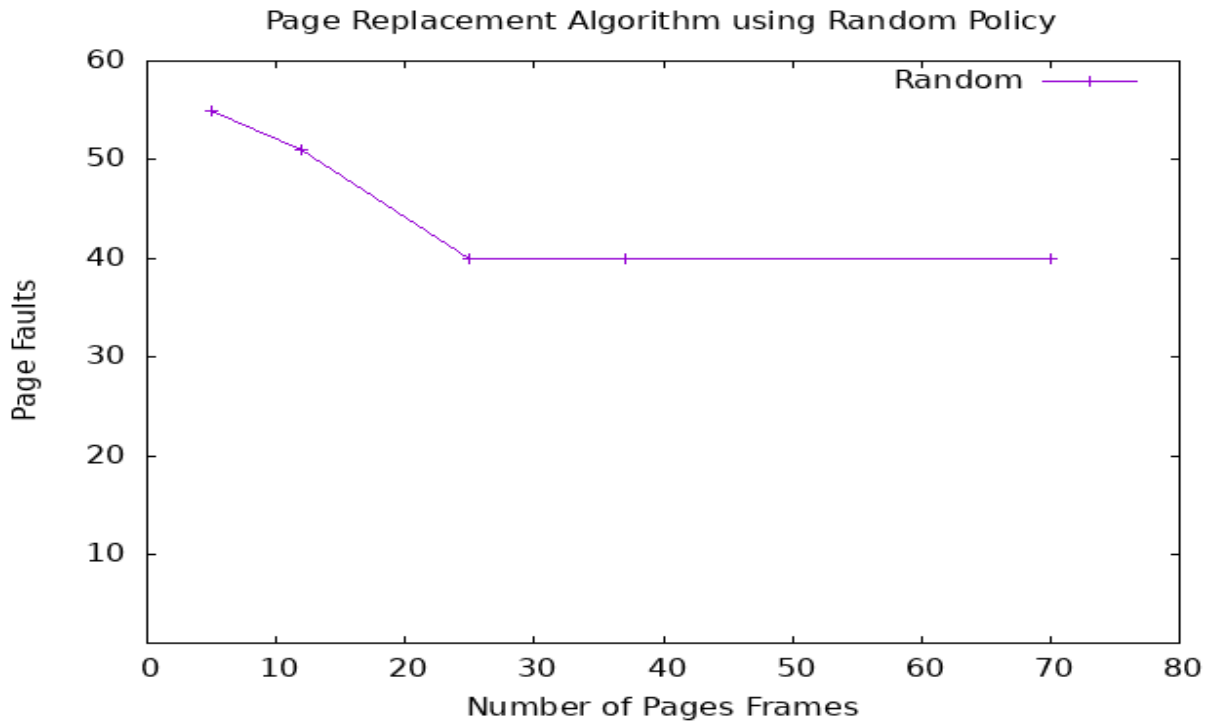
Req3.dat



Req4.dat



Req5.dat



Analysis:

The code also keeps track of the number of page faults for each algorithm and outputs them at the end of the program. To test the implementation, five different input files were created with different page request patterns and frame sizes. For each input file, the program was run with different frame sizes, and the number of page faults was recorded. The results were then plotted on a graph to compare the performance of the three algorithms with different frame sizes.
If the number of frames increases gradually the number of page faults becomes constant because all the pages requested can be fit into memory without any replacement.

Req1.dat

The performance is FIFO>LRU>Random

Req2.dat

The performance is LRU>FIFO>Random

Req3.dat

The performance is Random>LRU>FIFO

Req4.dat

The performance is LRU>FIFO>Random

Req5.dat

The performance is LRU>Random>FIFO

Conclusion

Overall, the type of workload and expected access patterns in the system determine which page replacement algorithm should be used. The LRU technique might perform better if the system has a high level of locality. However, if the access patterns exhibit a significant amount of randomness, then the

A more effective algorithm might be FIFO.
LRU functioned more effectively than FIFO in the req*.dat files. Random occasionally outperformed FIFO and LRU by a wide margin..