CS314 – Operating Systems Lab 8

210010062

210010038

This assignment's objective was to put three distinct page replacement methods into practice: First-In-First-Out, LRU (Least Recently Used), and Random. Operating systems employ these algorithms to manage memory when there are more logical pages available than physical frames. The web page Algorithms for replacement determine which pages should be deleted from physical memory to create room for newly required pages.

The task involved the implementation of these algorithms using C++ and examining them with various setups, including different page and frame sizes, as well as diverse page request patterns. The program utilized input files provided, which held the page request patterns for testing purposes.

Upon execution, the code parsed the command line arguments specifying the number of pages, frames, swap size, and the file name containing the page request pattern. Subsequently, it read the page request pattern and applied each of the three-page replacement algorithms to it.

1. FIFO Page Replacement Algorithm

The FIFO algorithm replaces the page that arrived first into the frames, without considering how frequently it has been accessed. Implementing FIFO in the code uses a deque to store the frames and checks if the requested page is already in the deque. If not, it adds it to the deque, and if the deque size reaches the number of frames, it removes the first element.

2. LRU Page Replacement Algorithm

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

1.3 Random Page Replacement Algorithm

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

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.

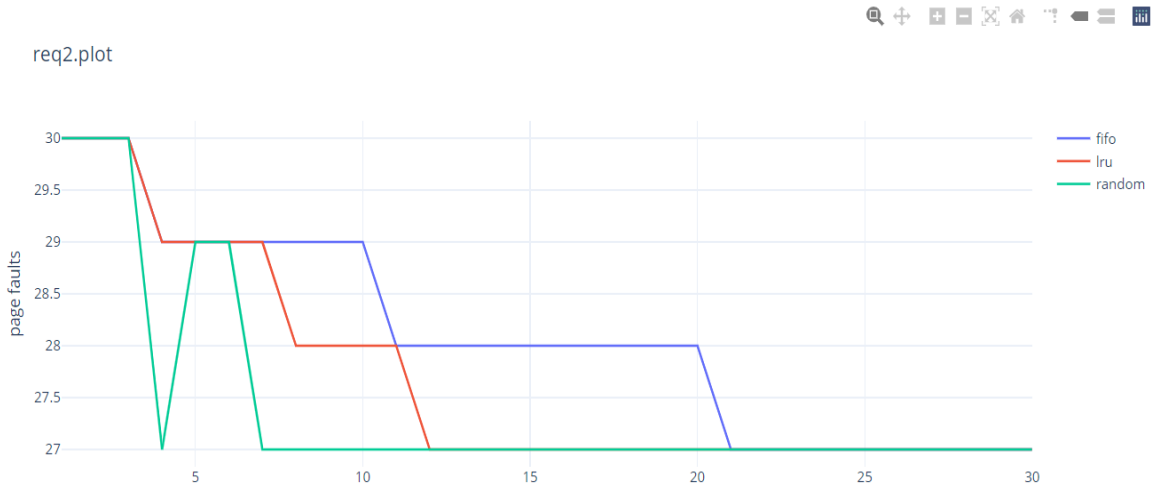
Req1.dat plot :

This is the graph showing Number of frames vs Number of page faults.
Number of pages = 60, frame number = 30

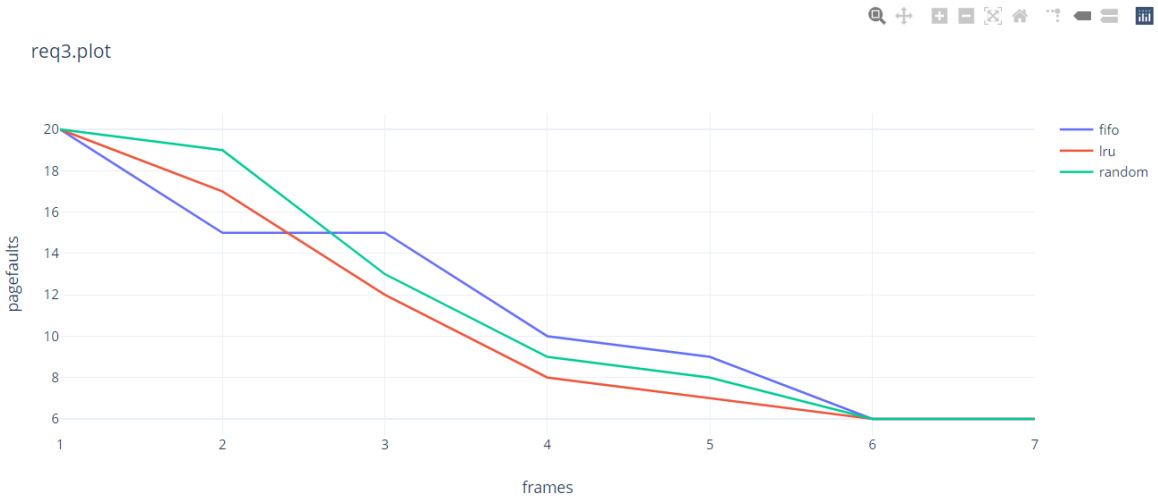
Req2.dat plot :

This is the graph showing Number of frames vs Number of page faults.
Number of pages = 30

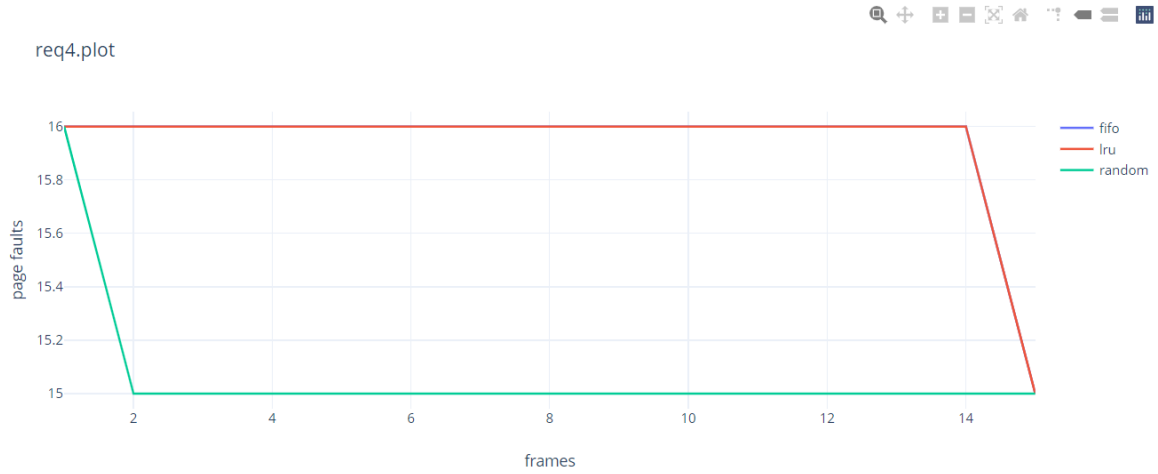
LRU performed better than FIFO in this case as least recently used page requests are not used again in the req2.dat file.



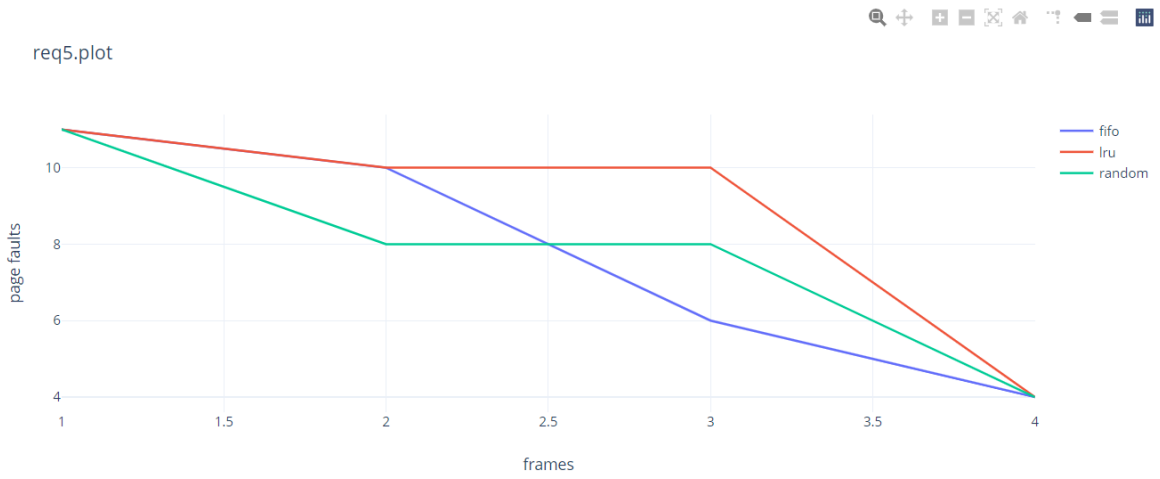
Req3.dat plot :
This is the graph showing Number of frames vs Number of page faults.
Number of pages = 7
In this case, as the number of frames increases, LRU performed better than FIFO. FIFO performed better when number of frames ae less.



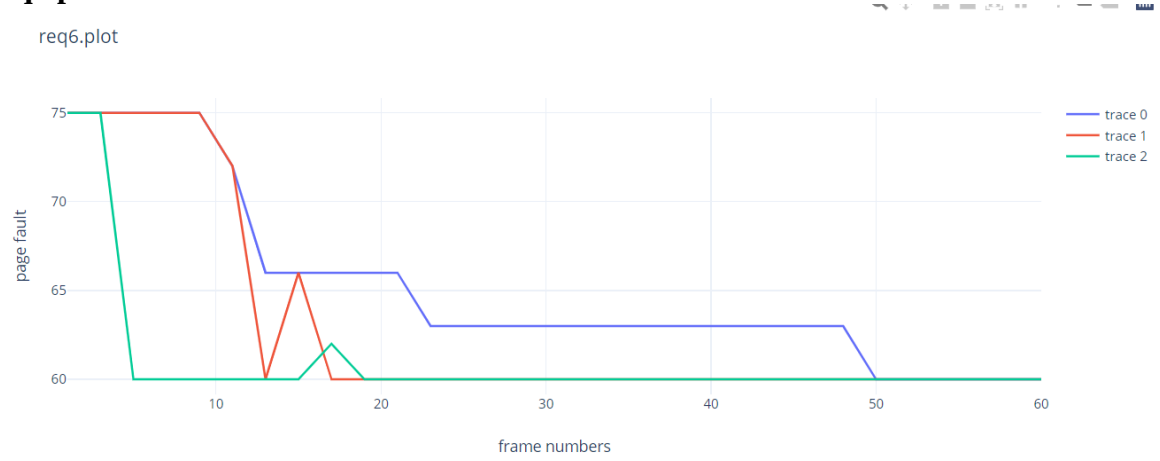
Req4.dat plot :
This is the graph showing Number of frames vs Number of page faults.
Number of pages = 15
Both LRU and FIFO performed equally in this case as page requests are not repeated again in the req4.dat file.



Req5.dat plot :
This is the graph showing Number of frames vs Number of page faults.
Number of pages = 5
FIFO performed better than LRU in this case as least recently used are used again in page req6.dat file.



Req6.plot



Conclusion

Overall, the choice of the page replacement algorithm depends on the type of workload and access patterns expected in the system. If the system has a high degree of locality, then the LRU algorithm may perform better. However, if there is a high degree of randomness in the access patterns, then the FIFO algorithm may perform better.
In the req*.dat files, LRU performed better than FIFO. Sometimes, random performed much better than FIFO and LRU.