CSC 345-01 Project #3: Virtual Memory Manager

Name: Chris Toala & Eric Rivera
Section: 01 w/ Dr. Yoon

1. Introduction

In this lab, we explored virtual memory by creating three versions of a memory manager that translates logical to physical addresses. The first version employs a complete set of frames with no need for page replacement, while the second and third versions use fewer frames and implement FIFO and LRU-based replacement, respectively. All three rely on a page table and TLB. This allows us to observe how different memory constraints and replacement strategies impact page faults and TLB effectiveness.

2. Implementation Details

The basic virtual memory manager, main1, processes logical addresses sequentially using a page table and a TLB. It reads each address from the input file, extracts the page number and offset, and checks for a TLB hit before consulting the page table. On a page fault, the corresponding page is loaded from the backing store into the next free frame. This approach uses FIFO for TLB updates and avoids page replacement using full-sized physical memory.

The FIFO-based approach, main2, handles address translation with limited physical memory. It uses 128 frames, later varied, and applies FIFO page replacement once memory is full. Each address is masked and broken into page numbers and offset, and resolved through the TLB or page table. On a page fault, the program loads a new page into a free frame or replaces the oldest page in memory. The TLB is updated using FIFO, and entries are invalidated if their page is evicted.

The main3 implementation uses the same 128-frame that main2 uses, but instead of FIFO, it utilizes LRU (least recently used). Each logical address is masked and is divided into page numbers and offset. The TLB is then checked for a hit, and if there is no hit then the page table is contacted. When a page fault occurs and if there are still free frames available, the needed page is loaded into the next frame until physical memory is capped. When physical memory is capped then the program uses currentTime to start the LRU process where it then dictates the least recently accessed frame as the victim which is then overwritten and afterwards the page table and TLB are updated accordingly. Entries for recently removed victims are then invalidated, LRU is proven to be better in terms of reducing faults more than FIFO in locality.

3. Experimental Results

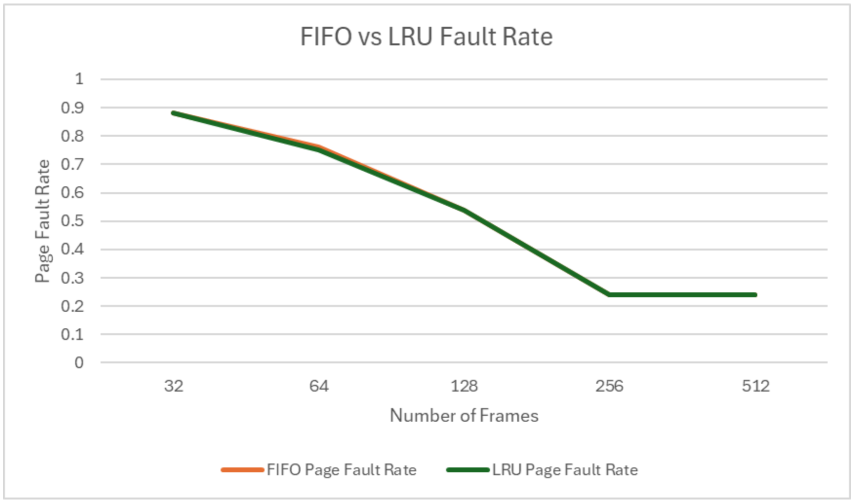
The experimental results compare the behavior of the virtual memory manager under different conditions. We track key metrics like TLB hit rate and page fault rate to evaluate the performance of each implementation. The following are the results:

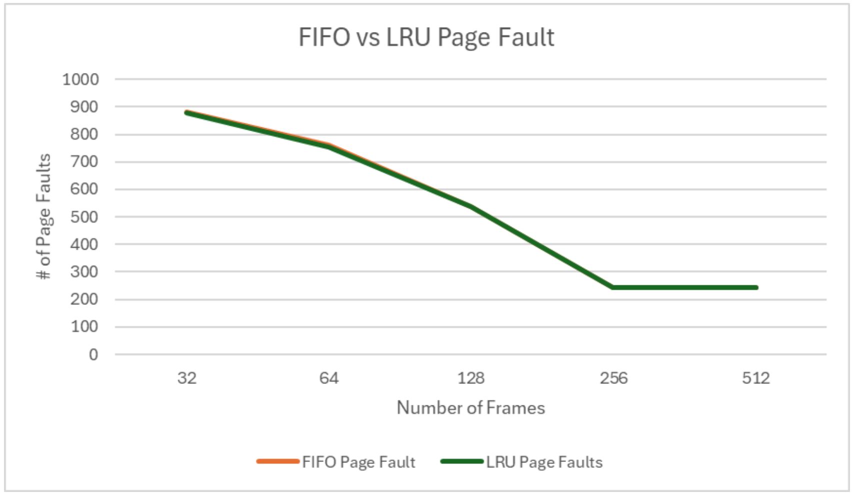
For main1.c, for 256 frames:

Total Addresses:	1000
TLB Hits:	54
Page Faults:	244
TLB Hit Rate:	0.05
Page Fault Rate:	0.24

4. Statistical Analysis

Number of Frames	FIFO Page Fault	FIFO Page Fault Rate	LRU Page Faults	LRU Page Fault Rate
32	882	0.88	879	0.88
64	762	0.76	754	0.75
128	538	0.54	539	0.54
256	244	0.24	244	0.24
512	244	0.24	244	0.24





5. Conclusion

This project explored how virtual memory is managed through paging, TLBs, and page replacement strategies. By comparing main1, main2, and main3, we observed how increasing the number of frames can improve page fault rates, to a point. The group also found that using LRU was slightly more efficient than using FIFO-based replacement. Overall, the results demonstrate the importance of efficient memory management when physical memory is limited.