Cedric Nicolas
Qua'on Thomas

COP4600 Project Two Report

I. Introduction

For this project, we were tasked with reproducing three Physical Memory replacement algorithms with the use of a simulated memory management system. The algorithms in question are as follows: First In First Out (FIFO), Last Recently Used (LRU), and the VMS Second Chance Algorithm. A relevant question to ask is "where do these algorithms come into play with memory management?" A computer can only hold so much information at a time in RAM, which is kept there in order to improve access times and overall computing performance. When a process fills up it's allocated space in the RAM, and requires a different page (or grouping of addresses), it must remove something from RAM. Now another question, and the theme behind this experiment, is raised: "How do we choose what to remove from RAM?". These are where the replacement algorithms come into play. As it is impossible for the computer to know which things it will and won't need throughout the lifetime of a process (it is not future-seeing), we have to choose an algorithm that will operate in such a way that seems it's performing as close to that as possible. This is why this experiment is important for us, as students, to gain a deeper understanding of why we would choose a certain replacement algorithm for memory management over another.

II. Methods

This project took the course of a week to complete. Averaging around four hours of work per day, the first two days were spent going over the project and determining what data structures would be needed, and how the work would be split up. We came to the conclusion that we would recreate the memory of a computer using these data structures:

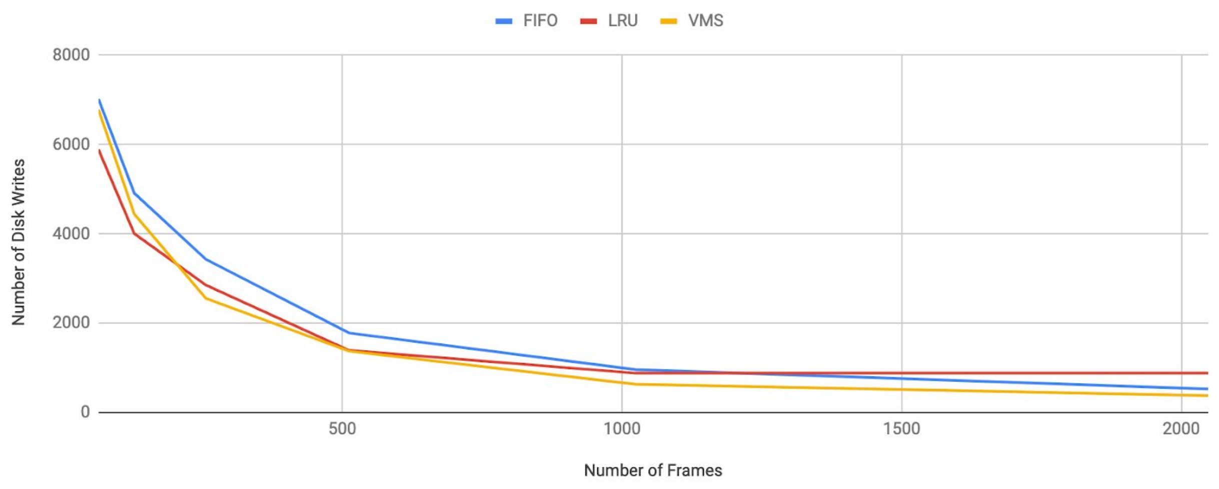
- An array of integers of size 2^20 to represent the page table
- A Queue to represent the RAM's pages for FIFO
- A Memory Reference Structure that holds an address, the access type, and it's position for LRU
- An Array of Memory Reference Structures to represent the RAM
- An Array of Memory Reference Structures to hold all traces from the text file
- Two Queues to represent each FIFO list for two separate processes in the VMS policy
- Counter variables for the amount of disk reads and writes

These structures in conjunction would allow us to recreate the memory of a computer in order to count the disk reads and writes that each replacement algorithm would need.

The reason we decided that the size of the page table needed to be 2^20 was due to the fact that the addresses in all trace files were 32 bits long, and each page was 4KB large. As such, it was determined that there could be a maximum of 2^20 pages, and that we only needed to do operations on the first 5 numbers in the hexadecimal address. Each address was represented as an unsigned data type, whereas all Queues were pointers to specific places in memory, so that we would be assured that the operations were done on the correct data.

As for testing, for FIFO and for LRU we had decided to do all of our tests of <nframes> size 128, only increasing it when we had fully implemented VMS in hopes to see the expected trends.

III. Results



As can be seen in the figure above, the results of our experiment are shown comparing the number of disk writes to the amount of frames provided at the start of the simulation. These tests were run on the gcc.trace file.The reason that we do not compare the number of disk reads to the number of frames, is due to the fact that we had implemented a page table large enough to read in all possible unique page numbers. Therefore, the only reason it would be necessary to perform a disk read is if a new virtual address needs to be translated. The number of frames used in our tests are listed below:

- 64
- 128
- 256
- 512
- 1024
- 2048

Strangely, LRU performed the same on 2048 frames as it did 1024. After going further, we had seen that it 881 disk reads on gcc.trace was not it’s best performance, as it got down to 868 with 4096 frames.

IV. Conclusions

Our results were pretty consistent with what had expected. As we did not have the ability to go through the traces and calculate true answers, we had based our answers off of the example given in the project description. We had known FIFO to generally be the worst of the three algorithms, and that was made apparent in our results. Following FIFO was VMS (at least as we increased the amount of frames available in the RAM). It was interesting to see the trend that VMS performed closer and closer to LRU as the number of frames increased. For instance, starting at nframes = 128, we see that LRU has a total of 4007 disk writes and VMS has a total of 4446 writes, the difference being 439 disk writes. As we double the number of frames to 256, we see that gap lower to 298.

We also noticed that number of reads stays consistent through all of the algorithms. We knew this was due to our implementation of a page table of size 2^20. Due to this, every page read in from the trace file was stored in the page table. With this, all pages would eventually be stored in the page table. As such, the disk read count variable represents the total amount of unique pages represented in the trace file. The page table did not affect the amount of disk writes, as disk writes only happened when a frame that was written to in RAM was replaced.

Overall, the experiment has definitely deepened our understanding of these replacement algorithms, as well as how the Virtual Memory and Physical Memory works in conjunction with the use of the TLB, Page Table, Disk, and RAM. That being said, I think if we had more time, it would've been nice to see how to implement the TLB as well.