Project 2 Report: COP 4600, Operating Systems

Salma H. Eid - U81194273

Julie Reyes - U7661122

1. Introduction

In order to study three page replacement policies, Least Recently Used (LRU), First In First Out (FIFO) and second chance replacement policy (VMS), this project simulates the virtual memory and analyses the memory behavior by calculating disk reads, disk writes and the number of hits obtained when simulating the three policy replacement algorithms. For this to be as close to actual memory behavior the algorithms read in trace files that contain real memory accesses, containing addresses and whether it is a read or write operation. This report also compares the number of hits with cache size (number of frames) and examines how those values vary using different policies.

2. Methods

Due to the possible large number of frames, our team decided on simulating memory using linked lists. Using a linked list as a base of the LRU, FIFO and VMS structures allowed us a constant removal time. Therefore, it was easier to implement removing a frame from memory and adding another. Because all the polices have different functionality, the linked list was used accordingly.

LRU

The LRU algorithm was implemented using a doubly linked list, with the head, first node, of the list representing the most recently used frame and the tail, last node, representing the least recently used frame. If a frame is found in memory it moves the node of that frame to the head of the list. If it is not found, the algorithm checks if the LRU list is full. If memory is full remove last node and add new frame at the head of the list. If memory is not full only add at the front.The first in first out method functions like a queue, which evicts pages from a page table based on if it has been in the page table the longest.

FIFO

To implement the the FIFO replacement policy for this project, a singly linked list data structure was utilized.Once the table is full of pages, the method simulates a FIFO memory page replacement policy. The method also keeps track of whether the memory accesses are of type read or write.

VMS

The VMS policy is implemented in order to deal with process that are competing for memory space. To create a policy for the competing process, each process contains a page table as a FIFO list that is half the size of the total memory, known as RSS. To maintain efficiency, second chance lists clean and dirty are also implemented. We used the pseudocode conditions provided in the project description and FIFO list to implement.

To measure the response of the three replacement policies we decided on number of hits as a reasonable, measurable variable that would allow us to compare algorithms. The results of this experiment and reasons we think causes such behavior is discussed below.

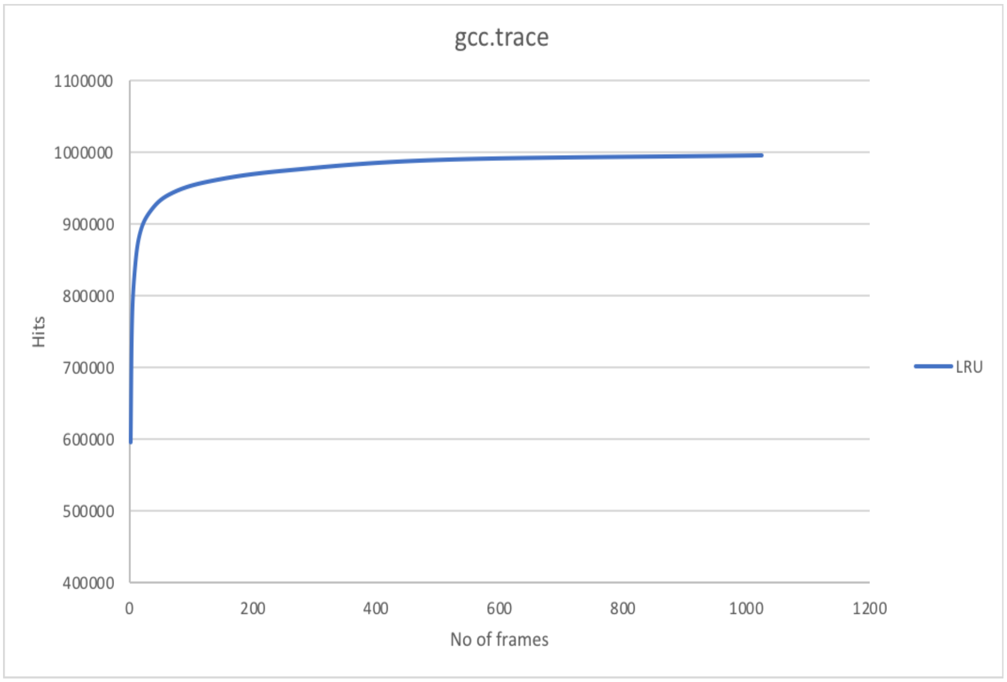
3. Results

Recorded number of hits received running each algorithm. Number of hits calculated such that the variable would be incremented when the frame was found in the cache/data structure = Events in trace - Disk reads.

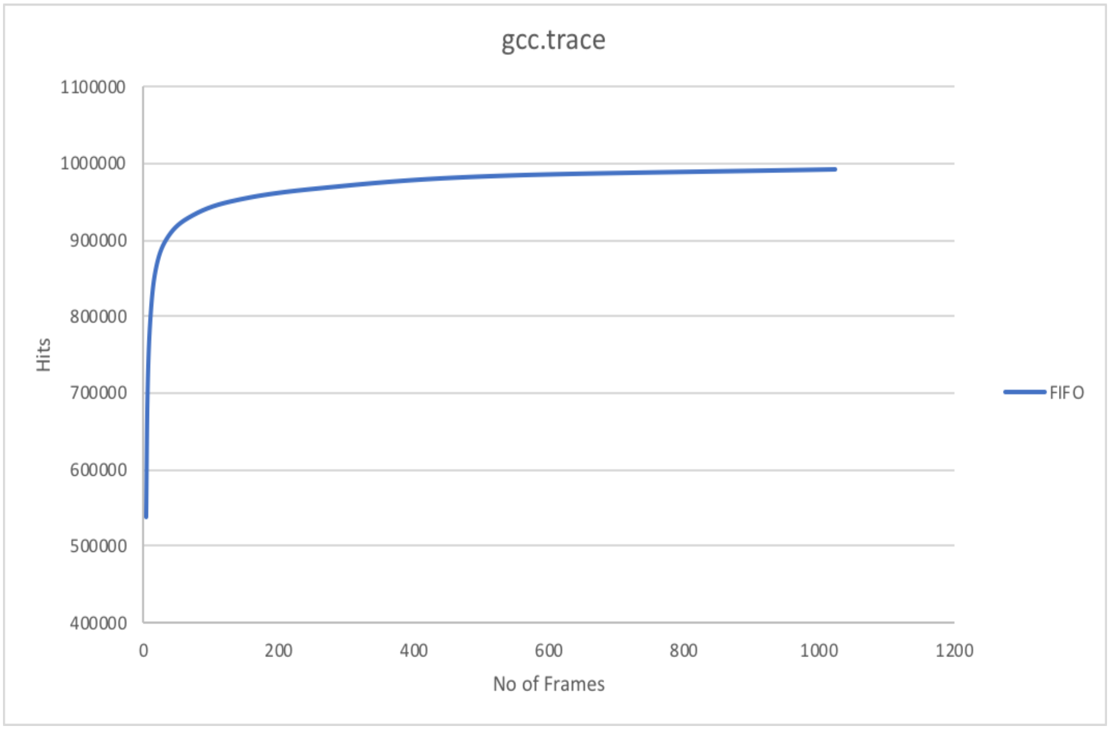
Results for the gcc.trace:

	LRU		FIFO		VMS	
No. Frames	Hits	Disk Writes	Hits	Disk Writes	Hits	Disk Write
2	596045	74158	539088	76432	454358	77389
4	756191	43187	697140	55013	643772	55899
8	828814	23983	794632	37524	766591	38638
16	883396	14749	861461	24043	844838	26328
32	915599	11737	901933	16759	893905	17961
64	940911	8863	929685	12053	924921	12720
128	959179	6131	951474	8487	948957	8879
256	974692	4231	968302	5945	968351	6021
512	989575	2173	984391	3425	983370	3635
1024	995609	1024	993327	1747	995751	589

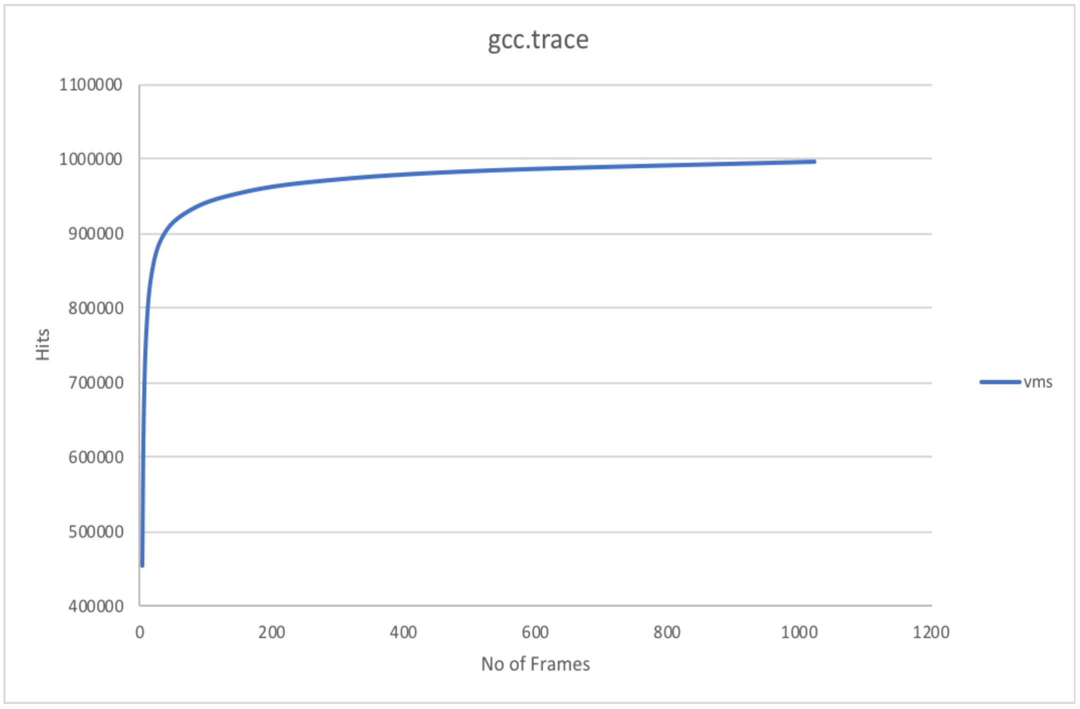
LRU:



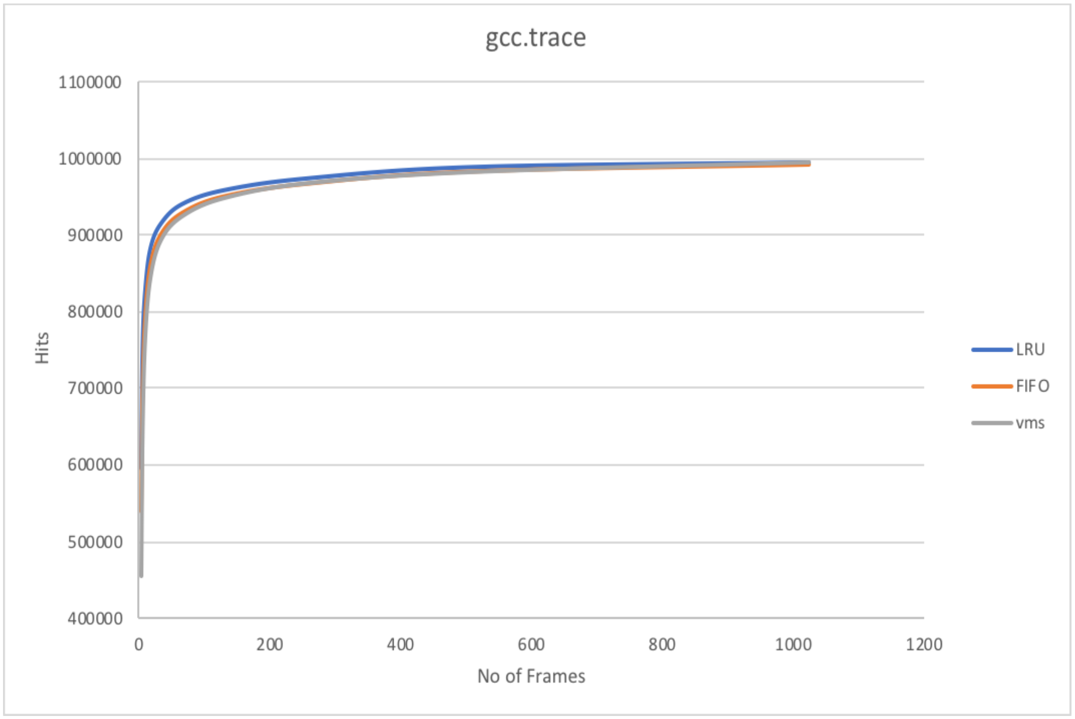
FIFO:



Vms alone:



LRU, FIFO and VMS graphs compared:



As shown above in the graph the number of hits for the LRU, FIFO and VMS algorithms are very close. Both numbers increase exponentially a small increase in the number of frames leads to a huge surge in the number of page hits received until the experiment reached approximately 300 frames in memory. At that point the both numbers for LRU, FIFO and VMS increased slightly with the increase in number of frames. At 600 frames and above, increasing memory allocations does not improve memory performance significantly. Although all three behave similarly with this trace file, LRU hits are slightly more that those of FIFO and VMS. With the behavior shown above we can conclude that the trace file, gcc.trace makes the LRU function as a FIFO. For example a pattern of memory accesses where the least frequently used frame is accessed first, or accessed more recently that the other frames in the list. As for the disk writes, the trend shown by the gcc.trace file suggests that as the page number increase in memory a larger decrease in page writes is obvious in VMS than in FIFO and LRU.

4. Conclusion

After completion of this project, our team was able to analyze virtual memory performance in terms of space and time efficiency. This enabled our team to understand the possible trade offs when implementing a memory replacement policy. This project allowed our team to delve into the specifics of how each policy operates under real conditions. Our team found the most challenging aspect to be the specifics in VMS implementation. Initially our team hypothesised that FIFO, would be the most inefficient policy. Under the conditions simulated in this project, that proved to be correct. A interesting observation that we made was the trade off in having too many memory frames in our page tables, as the search times increased proportionally. The most interesting aspect of this project coming into experimentation, was how VMS would perform under the simulation. It was interesting to delve into the intricacies of how a page replacement policy deals with process completing for memory resources, while performing in a more efficient manner than just FIFO. Overall, our team learned in detail the pros and cons of different policies.