Project 2 Report

Introduction:
Due to the constraints we encounter in computing we use page replacement in order to utilize the virtual memory. This process is performed to gain the ability of storing more processes in the virtual memory at the same time.

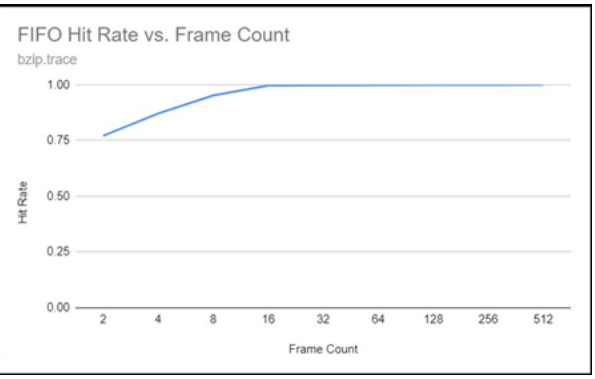
The approach is simply by checking that if no frame is available (free) then find a frame that is not currently used, followed by freeing it. The three methods we implemented for page replacement in this project are FIFO (First in first out), LRU (Least recently used) and SFIFO (Segmented first in first out).

In the LRU algorithm, the page being replaced from memory during a page fault is the one that has been least recently used. And the page table stores frames based on how recently they have been referenced or used. In the case of our implementation, the most recent pages are pushed to the front of the page table (cache).

One of the constraints faced is the inability to allocate more frames than what we have available. In our case, the number of frames will determine the size of the page table. This in turn affects the number of page hits and faults. Definitely, the performance will be affected due to the difference in case handling for each algorithm.

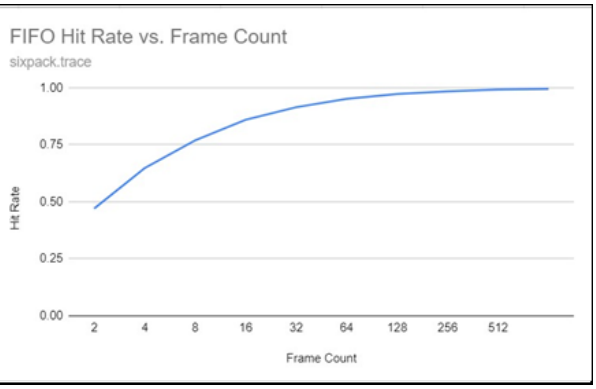
FIFO:

Log(frames)	Num of Frames	Writes	Reads	Hit ratio	FIFO
1	2	54321	228838	0.771162	BZIP.TRACE
2	4	38644	128601	0.871399	
3	8	18797	47828	0.952172	
4	16	1335	3820	0.99618	
5	32	851	2497	0.997503	
6	64	514	1467	0.998533	
7	128	305	891	0.999109	
8	256	125	511	0.999489	
9	512	0	317	0.999683	RUN TIME SUFFERS



The FIFO algorithm run using various amounts of memory size upon the bzip.trace file. Around 16 frames, diminishing returns starts heavily setting in, and with a hit ratio of 99.62%, each further added memory provides less and less benefit, while coming at the heavily increased cost of runtime. Notably, around 512 frames, runtime begins to increase exponentially, and the number of writes that are performed decreases all the way to 0. This might be a result of the program not being able to properly function when given that much memory to work with.

Log(frames)	Num of Frames	Writes	Reads	Hit ratio	FIFO
1	2	136484	529237	0.470763	SIXPACK.TRACE
2	4	94710	351810	0.64819	
3	8	57121	230168	0.769832	
4	16	31314	140083	0.859917	
5	32	18805	85283	0.914717	
6	64	11936	48301	0.951699	
7	128	8346	27778	0.972222	
8	256	5426	15440	0.98456	
9	512	3353	8069	0.991931	
10	1024	2368	5492	0.994508	RUN TIME SUFFERS



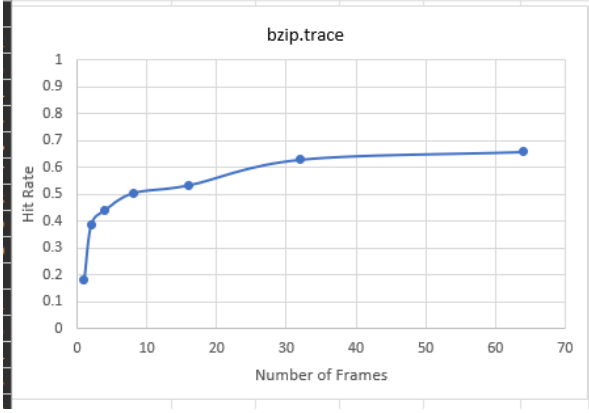
With the sixpack.trace file for the FIFO algorithm, it notably has a much less sharp increase in hit rate proportionally compared to the frame count, and also seems to have lower overall hit rates, starting at 50% whereas bzip.trace usually started at around 80%. For this file, it only starts to have heavily diminishing returns at around 256 frames, much higher than the 16 of the other file. Interestingly, the runtime only begins to suffer at 1024 frames, and at no point do the writes end up being reduced to 0. This could indicate that our implementation of our paging algorithms are better lent to the data contained within the sixpack file.

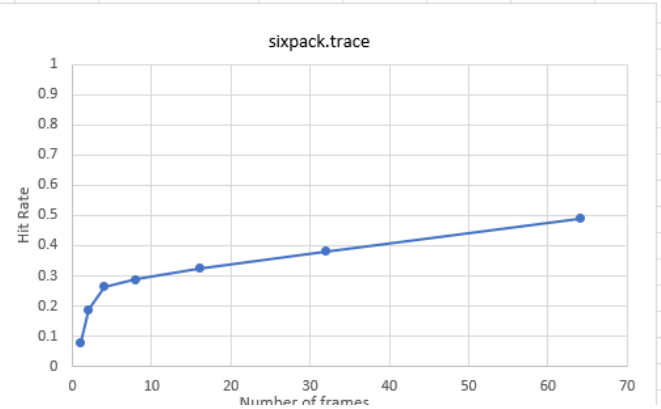
LRU:
For the LRU algorithm, the data has been recorded and plotted in the charts below. There is a general trend that can be observed in the two trace files. If we increase the number of frames then the hit rate is likely to increase. This can be explained due to the increasing chance of finding a page in the page table (cache) when the number of frames that the cache can hold is higher.

The biggest increase in the hit rate came before reaching 32. After reaching 32 frames, the rapid increase of the hit rate starts to decline. And even though it is still going to increase, the difference in hit rate is barely noticeable.

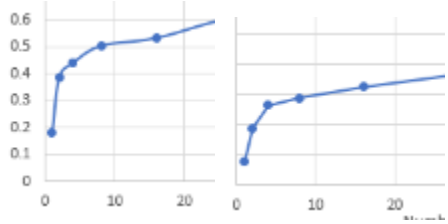
Number of frames	Hit Rate	Hits
1	0.183334	183334
2	0.388124	388124
4	0.440954	440954
8	0.503994	503994
16	0.533358	533358
32	0.628544	628544
64	0.657353	657353

Number of frames	Hit Rate	Hits
1	0.080214	80214
2	0.188521	188521
4	0.263741	263741
8	0.287421	287421
16	0.325413	325413
32	0.381698	381698
64	0.489475	489475





Conclusion:
For the LRU algorithm:
Both the trace files have had similar numbers when the number of frames and hit rate were put against each other. Increasing the memory size indeed increases the performance of the algorithm by raising the hit rate.



Both the trace files reacted very similarly to the threshold of when the hit rate starts to rapidly increase in the beginning. Followed by the decline slightly after the number of frames have been multiplied a few times.