Joel Shaju

Measuring the Performance of Page Replacement Algorithms on Real Traces

COP 4600

Introduction

Paging allows for fixed-sized, small chunks to be mapped between physical memory and virtual address space via use of pages; it helps eliminate external fragmentation and can help grow segments as needed. In this project, we simulate a page table, that loads pages until the maximum number of frames specified have been reached; when the number of pages fills up the simulated physical memory (reaches number of frames specified), then we put into place algorithms for page replacement. Choosing the correct algorithm becomes a problem, as it can greatly affect the cost and system performance; For our project, the created page replacement simulator should be able to perform the First-In First-Out, Least Recently Used, and Segmented-FIFO algorithms efficiently; results of each algorithm on various frame sizes will be analyzed.

For FIFO, when the page table is empty or the maximum size (number of frames) has not been reached, a page is pushed back to the table. Once maximum number of frames is reached, any time a page comes in that's not already in the table, it gets pushed to the back and the oldest page (front of page table) gets popped. For LRU, the same with FIFO applies, but anytime a page is referenced, it gets moved to the back of the page table (becomes latest). For Segmented-FIFO[1] (also known as VMS), a specified portion of physical memory is used as a secondary buffer; we used two page tables for the implementation: one for the primary buffer and one for the secondary buffer. Our primary buffer mimics the actions of FIFO, while the secondary buffer mimics the actions of LRU; the oldest page from the primary buffer is pushed on top of the

[1] Rollins Turner and Henry Levy. 1981. Segmented FIFO page replacement. In Proceedings of the 1981 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '81). Association for Computing Machinery, New York, NY, USA, 48–51. DOI:https://doi.org/10.1145/800189.805473

secondary buffer when the primary buffer is full, and the bottom of the secondary buffer is popped off. If a page is found in the secondary buffer, it moves to the top of the primary buffer. Anytime a page is not in the memory/page table, it will cause a page fault, which entails a disk read; if the dirty bit of the page to be discarded is 1, then it becomes seen as a disk write.

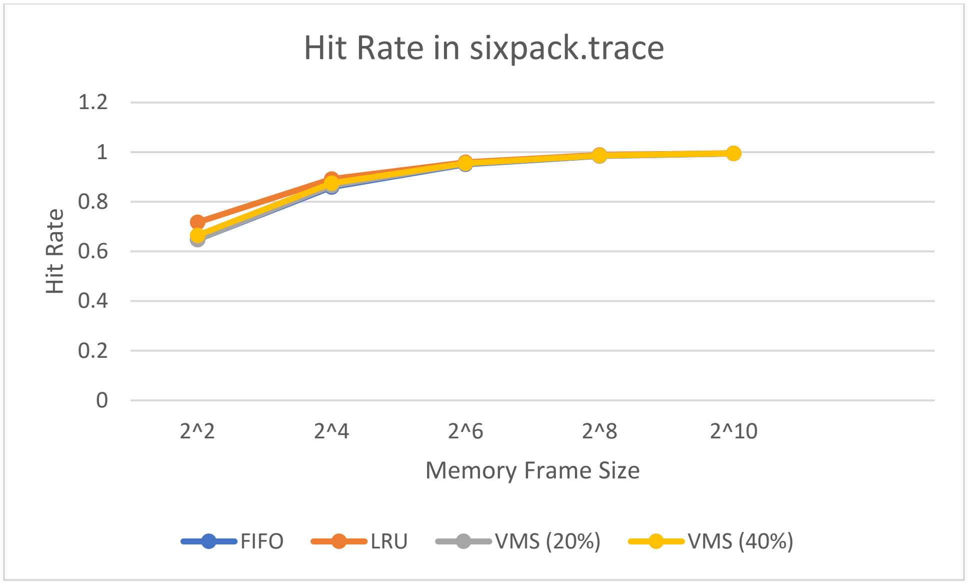
Methods

For our project, we are using trace files called sixpack.trace and bzip.trace. Each line in the trace file represents a trace event, with an 8-bit hex virtual address (5 bits for page number and 3 bits for page offset) and R/W permission ('W' sets dirty bit of page).

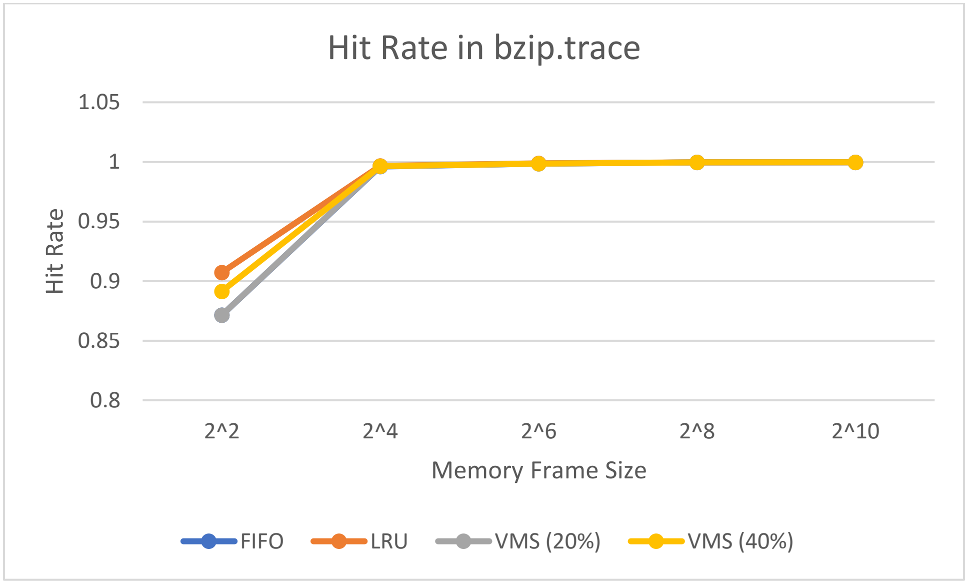
To measure performance of the page replacement algorithms, we ran the ran the simulator through various memory frame sizes (2^2, 2^4, 2^6, 2^8, and 2^10) for each algorithm (using VMS/Segmented-FIFO with parameters 20 and 40 each as separate algorithms). These sizes were chosen, as they demonstrate an excess of memory, a shortage of memory, and memory sizes close to what each algorithm needs; we believed that using over 2^10 would be impractical and would allocate too much memory for the process, while using natural numbers (such as 1,2,3…,10) would not show the depth of the algorithm and results would not be too differentiable from others. The hit rate was measured for each algorithm, for both sixpack.trace and bzip.trace.

Results

Hit Rate in sixpack.trace					
	2^2	2^4	2^6	2^8	2^10
FIFO	0.64819	0.859917	0.951699	0.98456	0.994508
LRU	0.71738	0.891318	0.958814	0.98876	0.995532
VMS (20%)	0.64819	0.864275	0.953837	0.985429	0.994877
VMS (40%)	0.665101	0.874579	0.955594	0.986355	0.99517



Hit Rate in bzip.trace					
	2^2	2^4	2^6	2^8	2^10
FIFO	0.871399	0.99618	0.998533	0.999489	0.999683
LRU	0.90723	0.996656	0.998736	0.999603	0.999683
VMS (20%)	0.871399	0.996281	0.998592	0.99951	0.999683
VMS (40%)	0.89142	0.996419	0.998621	0.99956	0.999683



Our results show that for all algorithms (regardless of tracefile being used), as the memory size (number of frames) used increases, the hit rate tends to increase (almost reaching 1). When looking at both tables (one for each trace file), it appears that after a certain size, there is not much of a change in hit rate. It appears that bzip.trace had an overall higher hit rate for each algorithm and less range in the hit rates when changing memory size.

Conclusion

In conclusion, it appears that LRU is the most efficient algorithm, as it has yielded the highest hit rate for each test, and FIFO is the least efficient, as it has yielded the lowest hit rate for each test. After a certain memory size, there are not many differences in performance; for sixpack.trace, not much difference is noticed after 2^6 frames, and for bzip.trace, not much difference is noticed after 2^4 frames. Using SFIFO of 20% and 40% are significantly better than FIFO, as well.

References

1. Rollins Turner and Henry Levy. 1981. Segmented FIFO page replacement. In Proceedings of the 1981 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '81). Association for Computing Machinery, New York, NY, USA, 48–51. DOI:https://doi.org/10.1145/800189.805473