Nicholas Everekyan, Kevin Gallagher

CS433: Operating Systems

Professor Zhang

November 19th, 2024

Programming Assignment 5 Report

Problem Description:

We were tasked with implementing a page table and 3 separate page replacement algorithms: First In First Out (FIFO), Last In First Out (LIFO), and Least Recently Used (LRU). Everything would be abstracted into their own separate implementations, providing for a clean and concise way to operate on each algorithm individually.

Program Design:

The designs of both FIFO and LIFO were relatively simple, using a queue and a stack respectively to make the algorithms work as intended. Adding and replacing a page in these implementations was simple as well, simply checking if the page already existed within the set, and returning a page fault if it did. Should a page fault occur, the page would either be added or replaced into the page table in the respective order of their algorithms. For LRU, it was a bit more nuanced. We used a list to store the pages, adding the most recently used page to the front of the list, and we also used a map to keep track of each page in the order that they were added to the list, which accounted for a slight increase to our overhead. From there, adding and replacing pages came

relatively similar to the previous implementations, simply updating the list and the map respectively.

Analysis:

As we used std implementations of containers for all three of our implementations (queue for FIFO, stack for LIFO, and the list / map combination for LRU), our implementations remain relatively concise and efficient. The extra overhead incurred by our LRU implementation is almost inconsequential when placed next to the scale of our program, as we are using hundreds of thousands of pages in the tests. Thus, efficient containers are required due to the scale of operations at play.

FIFO

Page Size	Memory Size	Page Faults	Time
1024	2^32	141266	0.37s
1024	2^16	178790	0.40s
4096	2^32	99811	0.32s

LIFO

Page Size	Memory Size	Page Faults	Time
1024	2^32	444862	0.61s
1024	2^16	541385	0.67s
4096	2^32	374021	0.54s

LRU

Page Size	Memory Size	Page Faults	Time
1024	2^32	132844	1.21s
1024	2^16	164938	1.24s
4096	2^32	89723	1.15s

What is interesting to see is the relations between the algorithms and different page sizes and memory sizes. As a baseline, we will use the 1024 byte page size with 2^32 bytes of memory. For this, the FIFO algorithm ran the fastest, with LIFO being about double the runtime of FIFO, and LRU being double the runtime of LIFO. However, LIFO had many more page faults than FIFO. What is interesting to note is that while LRU ran longer than the FIFO algorithm, it had fewer page faults, which is commendable. Looking into how different page sizes and memory sizes affect things, a smaller memory size overall makes the algorithms run slower, with more page faults, while a larger page size lessens page faults and makes the algorithm run faster.

Conclusion:

Our program successfully implemented these three algorithms for page replacement, as well as a properly functioning page table. One possible area of improvement would be to potentially decrease the overhead required in the LRU implementation by finding (or creating) a better container to store the least recently used page in, perhaps a modified priority queue of sorts, which would take a factor of how many times the page has been used into account for the sorting mechanism.