CS433 Programming Assignment 5 Lab Report

Lucas Birkenstock and Connor Toro

Problem Description

Our task in this assignment was to implement a page table, with multiple replacement algorithms. These include the first come first serve, last in first serve, and least recently used (LRU) algorithms. Implementing this required a deep understanding of how paging tables work.

Program Design

Our program was fairly structured and organized. For each replacement algorithm, there was a header, and .cpp file. Each algorithm had its own implementation of touch_page(), replace_page(), and load_page(). The first in, first out (FIFO) and last in, first out (LIFO) replacement algorithms had virtually identical load_page() function implementations.The least recently used (LRU) replacement algorithm had a somewhat similar implementation, but instead had to account for a K,V map. The first in, first out (FIFO) and last in, first out (LIFO) replacement algorithms also had identical touch_page() functions, where the number of pages was simply incremented. The least recently used (LRU) algorithm also does this, but also updates the value in the LRU map to equal touches + num faults + num replacements. All three algorithms have different replace_page() implementations, since they are completely different algorithms.

Starting with first in, first out (FIFO), this algorithm was one of the most simple. Its replacement implementation keeps a queue of pages, because a queue is FIFO. When a victim must be selected, pop the front page. After this, get the victim's frame number, set its valid bit to false, and update the page table's information at the function's input. Increment the number of replacements and return the victim.

Secondly, the last in, first out (LIFO) replacement algorithm. The simplest data structure to maintain LIFO logic is a stack, so we used that to store the pages. When a victim must be selected, simply pop a page off the top of the stack. Then, like the FIFO replacement algorithm, get the victim's frame number, set its valid bit to false, and update the page table's information at the function's input. Increment the number of replacements and return the victim.

Lastly, the least recently used (LRU) algorithm. It has the most complex replace_page() algorithm. First, increment the num_replacements variable, and update the map to reflect that value at the key provided by the function parameter. Then, iterate through the entire K,V map, updating a variable to track the least recently used page. This is done by comparing the V associated with every K. The value is incremented each time it is accessed. Once done iterating through the map, the victim is selected as the one with the smallest V. Remove that victim from the map, get the frame information, and update the page table. Return the victim page.

System Implementation

This assignment didn't take a particularly long time to implement, but it did require a large amount of technical knowledge. The different replacement algorithms are simple in concept, but actually adapting them to work for a page table was very technically heavy.

Results

Our code meets all requirements except for the assignment on gradescope, except for the extra credit of having a runtime less than one second. It will correctly use first come first serve, last in first out, and least recently used (LRU) algorithms for replacing pages in a page table.

Conclusion

To conclude, all assignment requirements and functionality were met, and served as a good learning experience for understanding how page tables and page table replacement algorithms work. We were unable to keep the runtime under one second, but wanted to devote more time towards studying for finals.