Ben Foltz-Miranda

Dr. Zhang

CS 433-02

02 May 2024

# Assignment 5 Program Report

## Problem Description

The problem that I am solving is being simulate the process of three different page-replacement policies: FIFO, LIFO, and LRU. This program is able to keep track of pages in the memory and free frames by maintaining a page-table data structure that’s size is equal to the number of pages in the logical memory. For each memory reference in the page-table, the page number is calculated and the program checks if this page exists in main memory, if it does not then a page fault is generated and then that page is loaded into main memory. If the main memory is full, then a page is selected to be removed, depending on the replacement algorithm, it chooses either the first page added, the last page added or the least recently used page.

## Program Design

The Replacement class is a base class used to simulate a general page replacement algorithm. The other replacement algorithms use the replacement class as a parent class so that the logic for which function be called. If the page is valid, it calls the touch\_page function, if the page is not valid but free frames are available, it calls the load\_page function and lastly if the page is not valid and there is no free frame, it calls the replace\_page function. Each child of the Replacement class has a different implementation of these functions. The FIFOReplacment class is designed to simulate the FIFO page replacement algorithm. This is achieved by storing the page number in a queue and keeping track of the next free frame with an int variable. The FIFOReplacement load\_page function creates a new page entry variable and then sets the page to valid, the frame number to the next available frame number and then sets the dirty bit to false. The page is then pushed onto the FIFO queue, the number of free frames is decremented and then the page table is updated to reflect the newly added page, lastly the next available frame number is incremented. The FIFOReplacement replace\_page function creates a new page and then sets the page to valid, the frame number to the next available frame number and then sets the dirty bit to true. The victim page is then retrieved from the front of the queue and then removed from the queue, the new page is then added to the FIFO queue and then added to the page table. Lastly the victim page is set to invalid in the page\_table and then the victim page number is returned. The LIFOReplacement functions work basically the same as the FIFOReplacement functions except instead of using a queue, it uses a stack as the data structure and then removed the page on top of the stack. The LRUReplacement class is designed to simulate the LRU page replacement algorithm. This is achieved by storing the page number in a doubly linked list that is acting as a stack and then adding each node of the linked list to a hashmap with the page number as the key. Each node in the linked list has a page number, and a node pointing to the previous and next nodes. The LRUReplacement touch\_page function creates a new node object by retrieving the node from the hash map, it then calls a remove helper function that removes the node from the linked list and then calls another helper function called add that adds the node to the front of the linked list. The LRUReplacement load\_page function creates a new page entry and then sets the page to valid, the frame number to the next available frame number and then sets the dirty bit to false. Then the page table is then updated to reflect the newly added page. A new node is created to represent the page that was added to the page table, it’s page number is set and then it is added to the hashmap and then added to the linked list using the add helper function. Lastly the free frames int is decremented and the next available frame int is incremented. The LRUReplacement replace\_page function creates a new node to store the victim page from the linked list, it then removes the victim page from the map and then removes the victim page from the linked list using the remove helper function. The victim page number is then stored in an int so that the victim node can be deleted. A new page is then created and then the page is set to valid, the frame number is set, and the dirty bit is set to true. Then a new node is created with the page number to be added as it’s page number and then it’s added to the node map and then added to the front of the linked list with the add helper function. Lastly the page table is updated to store the new page that was created, and the victim page is set to invalid and the page number of the victim page is returned. The LRUReplacement add helper function creates a node to store the node after the head, and then sets the new node’s next to point to that node and previous to point to the head, it then sets the head’s next to point to the new node and the old first node’s previous to point to the new node. The LRUReplacement remove helper function creates two nodes, one that stores the previous node to the node that is being removed and another to the node that is next to the node being removed. It then sets the previous node’s next to point to the next node and then next node’s previous to point to the previous node.

## System Implementation

I did run into some problems during my implementation, one issue was figuring out how to remove to remove the page from the stack in the LRUReplacement class. I originally implemented the data structure as a std::stack, but then quickly realized that there was no way to remove the page that was in already in the physical memory and add it back to the top of the stack without taking a really long time. I was able to figure it out by reading the section in the book that suggested using a doubly linked list to always have a pointer directly to each page in memory so if I needed to remove it from somewhere in the stack and add it back to the top of the stack I could just remove it instantly without having to search through the whole stack to find the correct page.

## Results

I believe that all the features that were required are included in my submission. I think that something that I would like to improve on in this project would be potentially switching out the raw pointers for smart pointers so I don’t have to worry about memory leaks from pointers being allocated but not actually pointing to anything. Another I would like to improve on would be to add the virtual memory simulation using LIFO replacement algorithm and LRU replacement algorithm to Test 1 so that the user could look through and see how the algorithm is working in more detail.

## Conclusion

I was able to solve the intend problem successfully. The program is able to successfully simulate the FIFO, LIFO and LRU replacement algorithms. A lesson that I’ve learned from this assignment think about the data structure that I am going to use for a specific implementation. An issue that I have is just starting with whatever idea I have first and then running with it and then going back later and fixing it when I realize that there is a better way to do it.