Jordan Sinoway & Michael Ralea
CSC345-02
April 9th, 2021

Project 3: Virtual Memory Manager

This project centered around developing a program that translates logical to physical addresses for a virtual address space with a size of 2^{16} bytes. The program was required to read the set of logical addresses from an input file and, utilizing a TLB and page table, translate each of the logical addresses to its corresponding physical address, and output the value of the byte stored at the translated physical address. Through the development, testing, and debugging of this project, we learned the steps involved in translating logical to physical addresses, including resolving page faults using demand paging, managing a TLB, and implementing a page-replacement algorithm. For our implementation the following requirements were implemented:

Reading Logical Addresses From File:

The fscan() function was first utilized to scan each line of the input file, addresses.txt. Each line was scanned and saved into the program as an integer, and then passed to the getPage() function for later calculations. getPage() masked the logical addresses to find the page and offset variables, as described in the lab handout, and outputted the logical address to out1.txt. Using the feof() function, the program stops reading the addresses once it reaches the end of the input file.

Translate Logical Addresses to Physical Addresses:

To calculate the physical address, the variables derived from the logical address and other global variables are used, in the equation “frame * page_size + offset”. The frame number is the frame which holds the page number taken from the logical address. The program initially checks the TLB for the page, and if it is found, returns the associated frame number. If it is not found in the TLB, each entry of the page table is checked. If the frame is still not found, the program then loads the page number from the BACKING_STORE.bin file, and sets a new frame number. After all the variables are found, the physical address is calculated using the equation, and the result is placed in out2.txt.

Retrieving Values Stored in Physical Addresses:

The values in the physical addresses are retrieved from the physical_memory variable in the logical address page's associated frame. Every frame in the physical memory has a size of 256 bytes, so the program uses the logical address' offset value. After retrieving this value, the program outputs it to "out3.txt".

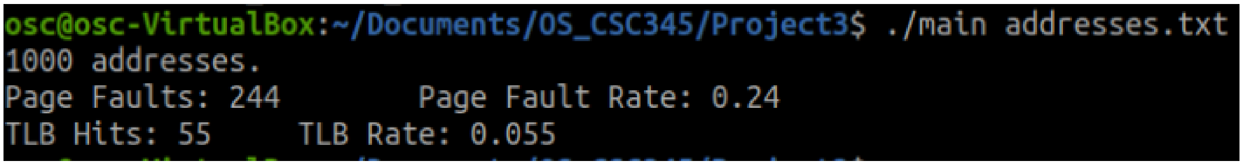
FIFO Based TLB Updating:

After the frame number is set, the TLBInsert() function is called to add the page number and frame number to the TLB. The first step is to check whether the page number is already present in the TLB. If it is not, the new page frame is added to the TLB. As the TLB only contains spots for 16 entries, a FIFO based replacement algorithm is used to make room for the page frames. Once the TLB becomes full, the replacement algorithm is used and the program replaces the oldest page oldest entry, which is at the top of the TLB. However, if the page frame is already in the TLB, if page replacement is enabled, an LRU based algorithm is used to move the position of the current page frame to the back of the TLB.

Calculating Page Faults and TLB Hits:

The main desired outputs of this program are the calculation of page faults and TLB hits. First, for the calculation of the page faults, a global page fault variable is incremented when a frame number corresponding to the logical address' page number cannot be found in either the TLB or page table. As part of theis, the program uses the readStore() function to read the 256 byte page from the BACKING_STORE.bin and stores it into an open page-frame in the physical memory. The next time this same page is called, it is stored within either TLB or the page table and a page fault will not occur. After the program is finished, it prints the page fault count and ratio of faults to total addresses.

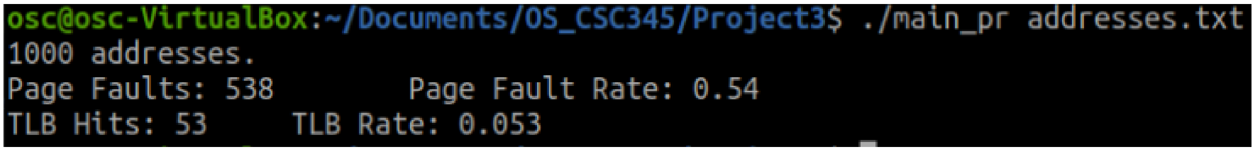
Additionally, the TLB hits are calculated, starting with acquiring a page frame through the 16 entry deep TLB. If the page frame is in the TLB, the frame variable is set and the program does not have to go any further. If not , the program will search through the page table and BACKING_STORE.bin. If found, the global TLB hit variable is incremented, and at the end of the program the total TLB hit count and ratio is printed. A sample output without page replacement for the calculation of the page faults and TLB hits can be seen below:



```
osc@osc-VirtualBox:~/Documents/OS_CSC345/Project3$ ./main addresses.txt
1000 addresses.
Page Faults: 244        Page Fault Rate: 0.24
TLB Hits: 55     TLB Rate: 0.055
```

Page Replacement for main_pr.c:

An alternate version of main.c was created, main_pr.c, which utilized a LRU-based page replacement algorithm. As long as the physical memory and page table have not been fully filled, the program will continue to add new page frames without any problems. After they have been filled however, the main_pr.c uses a LRU-based page replacement algorithm, which pushes out the oldest memory component and adds the new data to back of the memory location. A sample output with the page replacement algorithm for the calculation of the page faults and TLB hits can be seen below:

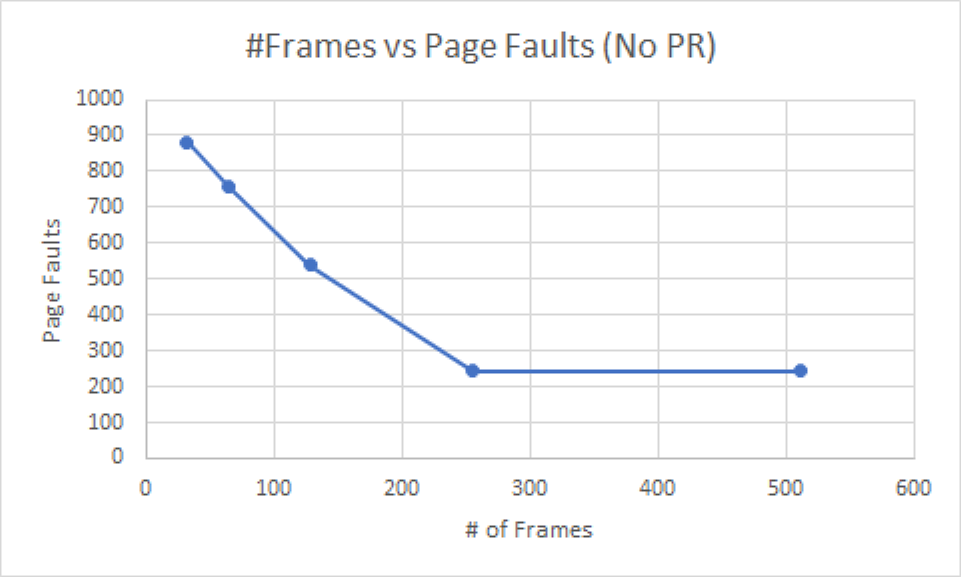


```
osc@osc-VirtualBox:~/Documents/OS_CSC345/Project3$ ./main_pr addresses.txt
1000 addresses.
Page Faults: 538        Page Fault Rate: 0.54
TLB Hits: 53     TLB Rate: 0.053
```

Varying Frames Graph:

As directed in the lab handout, multiple values for the frame number were tested for main.c, without page replacement. The absolute number of page faults and the page fault rate (out of 1000) was recorded for these five cases. The table showing these values is shown below, along with a graph showing the trend as the frame number increased. The page faults and rate flattened out after 256, presumably because at that point, the memory becomes saturated.

# Frames	Page Faults	Page Fault Rate
32	881	0.88
64	760	0.76
128	538	0.54
256	244	0.24
512	244	0.24


#Frames vs Page Faults (No PR)
Page Faults
1000
900
800
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
# of Frames