COMP304
Project 3 Report

Murat Güç - 68967 Yakup Can Karacaoğlu - 69063

Part 1 -

In this part we implemented the virtual memory manager according to the 20 bit masks. To do that **PAGE_MASK** gets the value of (1111 1111 1100 0000 0000) -> 0xFFC00 and **OFFSET_MASK** gets the value of (0000 0000 0011 1111 1111) -> 0x003FF.

These masks are to calculate the offset and logical page number as integer values derived from logical addresses.

Then, we implemented the helper methods which are

search_tlb: Iteratively, it checks the tlb table where the parameter of the logical page is located and returns the corresponding physical address.

add_to_tlb: In the fifo replacement policy, it replaces logical and physical addresses one by one according to the tlbindex. We reached a specific location by tlbindex % TLB_SIZE.

Lastly, we handled the page faults that happen when the TLB miss occurs thanks to the memcpy function which is about to swap memory between main memory and backing store. the contents of the page from the backing store to a free page in main memory. It then updates the page table to indicate that the logical page is now located at the physical page in main memory. The physical page in the pagetable corresponding to and coming from search_tlb becomes free_page. The variable "free_page" is updated to the next free page in main memory, and the page_faults variable is incremented to keep track of the number of page faults that have occurred.

Part 2 -

FIFO -

In this part of the project, the size of the virtual memory which is 1024 is bigger than the actual physical memory which is 256. This normally requires handling of the swapping operations between main memory and hard drive (It is used as BACKING STORE.bin in this project). It also requires some page table updates. For the scope of this project, we did not implement the rewrite of the content back into the hard drive but the page table and TLB updates are handled according to FIFO.

The implementation is started with downsizing the array of the main memory. The page size should be the same as the first part, but the number of pages is divided by 4. After that, the implementation of the page fault area is changed. There is a for loop to clear out the old instances of pointers to a page in physical memory. If there is an old instance, it means that we

are trying to overwrite a physical memory, so a swapping operation is occurred, and the content is changed. The old pointer should not reach the new content, so we clear the old pointers. Same procedure should be applied for the TLB. To clear the TLB, two new functions are created. “**search_tlb_physical**” is used to check that there is any logical memory pointer to a specific physical memory in TLB, it returns the logical memory address. Then “**update_tlb”** is used to change the TLB content according to the result of the “**search_tlb_physical**”. The update operation is to clear the old logical memory from the TLB and put the new one as matching with the old physical memory. Lastly, the free page counter variable is updated as circulating on 256.

In that way, we have completed the implementation by cutting the connections of old logical memories to updated contents.

LRU -

The only difference between Least Recently Used (LRU) and FIFO parts is choosing the next free memory. As implied in the name, the algorithm is based on choosing the physical memory which haven’t been used for a while. To achieve it, we created an array with 256 size. The array is used to keep the information of the time of the memory reaching.

We initialized the array 255 in the first index and 0 in the last index. To choose the next memory to swap, we always choose the memory which has the value of 255. The 255 means it is the least recently used memory. After reaching the memory, we change the value of the index of the reached memory to zero. It means it is the last used location. Then increase the rest of the elements by one and take the modulus with 256 in the array. It is important to choose the next swap location. After increasing, the element which has the 255 is the next location.

We implemented the algorithm when the page fault occurs, but we couldn’t handle updating the LRU array when the page table hits and TLB hits occurs. So, it is not working as desired, but it is the brief explanation of the system that we used.

To compile:

./a.out ../BACKING_STORE.bin ../addresses.txt -p 0 - for FIFO

./a.out ../BACKING_STORE.bin ../addresses.txt -p 1 - for LRU

Github repo: https://github.com/gucmurat/COMP304_Fall22_Project_3