Group 16: Blake Johnson, Alan Hoffman, Pamela Cardoso

Project 5 Report

Solution Description

Our Virtual Memory Manager consists of 4 main parts:

1. Welcome/Initialization – The function "welcome()" prints a welcome message, displays the constants of the program, asks the user which TLB replacement strategy to use and whether to display the physical addresses or not. Then the initialization of the page table array and TLB array is done. The initialization consists of filling the page table with all -1's, this is how the system will determine if a certain position in the page table is being used or not (-1 indicates the position is not used). The same is done for the TLB. The mainMemory_counter is initialized to 0. This is used to keep track of which index in the main memory is to be filled next with the new frame that is read in.

2. Read Input File – Once the needed information has been gathered from the user the input file is read and the logical addresses are saved in an array. The function "ReadInputFile()" takes care of this step.

3. Main Loop – The main function contains a "for loop" that simply iterates through all of the logical addresses that we read from the file. Each iteration of the loop processes one of the logical addresses individually by finding the value at that location and printing it. This loop takes care of the basic logic behind the memory manager and uses several functions to accomplish its purpose. "GetPageNumber()" and "GetOffset()" use bit shift operators to extract the page number and offset from the current address. Once the page number is established the memory manager will search the TLB array and the page table array using the functions "SearchTLB()" and "SearchPageTable()." If the page number cannot be found in either of those arrays the function "HandlePageFault()" will access the "BACKING_STORE" file and read a frame into the main memory at the index specified by the mainMemory_counter. The main memory is represented by a 2 dimensional array. The first dimension represents the frame number and the second dimension represents the offset. This allows the main memory to be easily accessed once we find the frame number in the TLB table, the page table or the "BACKING_STORE" file. When a page fault occurs "HandlePageFault()" will call a function called "TLB_Replace()." This function is passed the variable that tells the system to use FIFO or LRU based on what the user selected in the "Welcome" section. Once the system has the frame number and offset "AccessMainMemory()" is a very simple function that will use the frame number and offset as indexes to the 2 dimensional array and will return the output value at that location. "WriteOutput()" will then print the current address that is being processed and the value at that location to the screen and to the output file.

FIFO Replacement Strategy: Simply shifts the entries in the TLB array up and removes the entry in the first index. Then the new entry is added in the last index of the array, so it will be the last one to be replaced.

LRU Replacement Strategy: Each TLB entry has a timestamp field called "last_used." Every time this entry is used its "last_used" field is updated to the current timestamp. When it comes time to replace an entry the system searches for the entry that has the smallest "last_used" value and replaces that entry in the TLB array.

4. The last step is handled by the function "WriteStats()" and it simply prints out the counters that were keeping track of the page faults and TLB faults while the main loop was executing.

Questions:

(1) How did you guarantee that each logical address is translated to the correct physical address?

By using bit shifting operators we find the page number and offset and use those values as the "physical address" because they are used as indexes to the main memory 2D array. The physical address is made up of the frame number and the offset, so in our design the frame number is the first index of the 2D array and the offset is the second index.

(2) How did implement the page table, physical memory, and TLB?

The page table is a simple array. The TLB is an array of structs and each struct has a page number, frame number and a last used counter that is used by the LRU algorithm.

(3) Does your program realistically and accurately simulate a virtual memory system?

Our system realistically simulates a virtual memory system because it follows every step that a real system would take except a page replacement strategy because that was unnecessary due to the fact that our virtual space was identical to our physical space.

(4) Did you use the Java operators for bit-masking and bit-shifting?

We used the operators: ">>" and "&" to handle our bit-masking and bit-shifting.

(5) When a TLB miss occurs, how do you decide which entry to replace?

We have a function that replaces the entries using a FIFO strategy and we have a function that uses a LRU strategy. Depending on which strategy the user selects at startup we call one of these functions to handle the replacement.

Generality and Performance Criteria

Our Virtual Memory Manager is general and easily configurable. We used constants and a very function-oriented approach to accomplish this. Our main function is short and to the point, it simply makes a number of calls to certain functions and passes those functions the needed parameters. Almost all the processing of data is done in those functions and then returned to the main function. This is a function-oriented approach and makes the code very readable and reliable.

Questions:

(1) How general is your solution?
Our Virtual Memory Manager is very general and flexible. We accomplished this by using many constants; this way by simply changing a constant value the entire code could still be used.

(2) How easy would it be to change parameters such as the size of the TLB?
It would be extremely easy to change the parameters of our program. The only thing that would have to be done is changing a value of one of the constants.

(3) Does your program only load pages from the backing store when they are needed?
Yes. The function "HandlePageFault()" is the only function that access "BACKING_STORE" and it is only called when the given page number is not found in the TLB or page table (i.e. when the searching functions for the TLB and page table return -1's).

(4) Does your solution allow the physical address space to be smaller than the virtual address space?
Not in its current state. It is designed to deal with a physical address space that is the same size of the virtual address space, but with some tweaking and more extensive testing it could make this possible.

<u>Miscellaneous Factors</u>

Our Virtual Memory Manager is written in a way that the code is very readable, general, and modular. As you can tell from the description we used a strict function-oriented approach. Our functions have very low coupling because they do not depend on each other to accomplish their goals. Every function has a single job and the main function uses each function to accomplish its purpose. We commented every single function to make the code readable and human friendly and the use of constants make it configurable and general.

Questions:

(1) Is your code elegant?
Yes. Our code is well thought out and elegant. Some functions are just a few lines of code because they accomplish a simple task. We did not use a single global variable, instead we pass parameters to each of the functions and they return a certain value. Global variables do not lead to elegant code.

(2) How innovative is your solution? Did you try any ideas not suggested here (e.g. a choice of replacement policies for the TLB)? Innovative ideas that go beyond the requirements could receive extra credit.

(3) Did you document all outside sources?
No outside sources were used.