Anu Khatri

CSCE 410-500

Professor Da Silva

11 April 2021

Design Document – P4

vm_pool.H

I added the variables that corresponded with the parameters passed into the constructor. I made a variable mem_region_count to keep track of how many regions there at any given point in my mem_regions, which is an array of the struct mem_region. This struct is just helpful to keep the address and the size in one place for any given region (kind of like nodes when making linked lists). I also made mem_region_limit, which is the largest number of regions allowed, so that I don't exceed that number when allocating.

vm_pool.C

Constructor

I assigned the parameters' values to the respective variables. I then put mem_region_count's value at 0 since this is the constructor, and no regions have been allocated. I also put mem_regions in a frame of frame_pool by calling frame_pool's get_frames(). I then did register_pool to register the newly made VMPool object for my vm_pools object in PageTable.

allocate()

For this one, I first start with checking if the passed in _size is equal to 0 (since then there is nothing to allocate), or if the mem_region_count has exceeded mem_region_limit (which means that no more can be allocated). For these two conditions, I returned 0. If it passes past that check, then I will start the allocation process. If mem_region_count is 0, then that means that we have to start at the base address, so I make logical_addr equal to that. Otherwise, I will use the address and size of the previous region to calculate the address of the new address. I put logical_addr in the respect address value and the passed in _size for the size of the mem_region just made. Lastly, I increment mem_region_count to make sure I am keeping track correctly of how many there are.

release()

I created the variable index to see where the given _start_address is in the regions. I went through the mem_regions object, and broke the loop once the index was found. I then used that in another for loop to free up each page. I incremented the address by the machine PAGE_SIZE in order to get the address of the next page in that region. Lastly, I had to fix mem_regions and move the data down to a lower index of mem_regions so that there wasn't a hole in the array and so that mem_region_count can continue to place new regions at the correct index.

is_legitimate()
I went through mem_regions (using mem_region_count as the limit for the index), and I used the address as the beginning of where _address could be, and then the address + size as the end of the range. If _address was between that in any given region, that meant it was a legitimate address and returned true. If after going through all of mem_regions and there wasn't _address, that means it was not legitimate and consequently returned as false.

I could not solve an error that I had in Part II involving check_address (even when is_legitimate always returned true), so I was not able to completely check the correctness of Part III.