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.