Implementation of the VM Pool

Vishruth Raghuraj Gollahalli

1. **Part I: Support for large address apces:**

   For the first part, have made use of recursive page table lookup to manage larger address spacing using free mapping. This was done by adding the set address of the page directory into its last entry. This was done in the PageTable::PageTable() constructor.

2. **Part II: Registering VM Pools:**

   a. Implementation of the register_pool:
   This function first checks whether the head pointer to the vm pool is not NULL. If it is, then the given vm pool is added as the head of the list. If not NULL, then we traverse through the list and register the pool by adding it to the end of the list.

   b. Implementation of the free_page:
   First we obtain the page directory and page table entries to find the correct frame to release. This makes use of the existing logic to find the page directory and table index as used in the handle_fault function.

   c. Implementation of the is_legitimate:
   A simple function, checks whether the given address falls within the boundaries of the base address and size. If it does, then returns true, else returns false.

3. **Part III: Allocator for Virtual Memory**

   a. Implementation of allocate:
   First we make sure that there is enough memory available in the vm pool. If there is not then we break out of the function with error. If available, first, we calculate the number of pages required. Then, using the memory data structure, we calculate the base address using the base address and size of the previous entry. Once we have the new base address, it's just a matter of setting the size of the newly allocated memory by calculating size as the number of required pages times the page size. Finally we update the variables responsible for storing info about the memory pool data structure by setting available memory and the number of regions now occupied in memory.

   b. Implementation of release:
   This happens in three major steps.
   Step 1 here is to first identify the region that has to be released. By looping through the list, we can find the correct region to be released.

Step 2 is releasing the region. I have implemented this by simply shifting the list entries to the left at the location of the released region (delete element in linked list logic).
Step 3 is to finally update info about the memory data structure by adding the available memory and reducing the count of regions currently being used.

(Note: I have added only one file to the given set of files. This is the test_script.sh bash file which I added to run all the commands for testing together. No other additions have been made to the folder provided)