Hung-Hsueh Shih (UIN: 132006330) CSCE611 Operating System
Machine Problem 4 : Page Virtual Memory Management and Memory Allocation

In order to complete memory management I need to use recursive page table look up and virtual memory pool. This machine has three part, first part I set the page directory and page table page in to process memory pool and use recursive page table look up to modifies the entry. Second part is I assign a virtual memory pool to each page table and add two function register pool and free page to free the associated pages and frames. In the last part, I create a virtual memory pool same as what I did in the MP2.

Page_table.H
In the private variable I assigned a virtual memory pool pointer array to store vm pool object pointer. I also defined four additional functions, register_pool which is used to register a virtual memory pool with the page table, free_page which is used to release frame and mark page invalid,PDE_address which is used to get the page directory entry when paging turn on, and PTE_address which is used to get the page table page entry when paging turn on .

Page_table.C

In Page_table() I move the page directory and page table page into process memory pool by using process memory pool get_frames() and I assign the last entry in page directory to point to the head of page directory.

Pde_address() I use recursive page table look up to get the page directory entry when paging turn on. The recursive page table look up for page directory is 1023| 1023| X input is current logical address.

Pte_address() I use recursive page table look up to get the page table page entry when paging turn on. The recursive page table look up for page table page is 1023|X | Y input is current logical address.

handle_fault() I first check whether the memory is legitimate if is invalid then abort. If the memory address is legitimate then I execute page fault handler routine instead using the original page fault address, I use pde_address and pte_address I defined above to access the entry.

register_pool() traverse the vm pool maintain by this page table to find the first available vm pool and set the input to the vm pool.

free_page(): I release associated frames and set the page table page entry to invalid.

VM_pool.H I create a data type struct region for the vm pool object which contains base address and the size of each vm pool.

VM_pool.C
In constructor I initialize all data for vm pool object and point page table to this vm pool. I also initialize the first item in vm pool to the base address and one page size. In the end, I initialize the rest of item in vm pool array to 0

allocate()
In allocate I first to find out how many pages I needed, I divide the given size to page size to see how many pages I needed and set the the corresponding vm pool item base address and size

release()
I traverse the vm pool array to find the item which contain the same base address and release it

is_legitimate()
I traverse the vm pool array to see whether the memory is inside the region. If it is, return true else return false.