CSCE 611

Gautham Srinivasan (UIN: 927008557)

MP4 Design Document

Objective:

The objective of machine problem 4 is to complete the memory manager by making use of recursive page table look up and virtual memory pool.

Recursive Page Table:

As CPU issues logical address, we make use of recursive page table to access page directory and page table entry. The last entry of page directory (1023) is pointed to the first entry. The MMU thinks the address is of the format 10 bits – page directory index, 10 bits – page table index and 12 bits for offset and access the logical address.

We access the page directory entry by making the logical address of the form:

1023 | 1023 | X bit | 00 where X is indexed to page directory entry

We access the page table entry by making the logical address of the form:

1023 | X bit | Y bit | 00 where X is indexed to page directory entry and Y is indexed to page table entry

Detailed implementation:

a) **PageTable() –**
Allocate frames from the process pool. Set the last entry of page directory to point to the page directory start address and mark it as present.

b) **handle_fault() -** This function will be called whenever the CPU is unable to locate the page or the page table the error code 14.
   1. This function first checks the addresses is legitimate or not by calling legitimate function from vm_pool. If the address is not legitimate, then throw error.
   2. Check for the error code 14 which determines the page fault
   3. Check whether the page directory entry is present (**1023 | 1023 | X bit | 00**), if not then allocate a frame from process pool and update the page directory entry.
   4. Check whether the page table entry is present (**1023 | X bit | Y bit | 00**), if not then allocate a frame from process pool and update the page table entry.

c) **register_pool () –** This function keeps track of the vm_pools by storing the address in a list.
d) **free_page () –** This function first checks if the page table entry is valid. It then releases the frame and updates the page table entry. Lastly, it flushes the TLB by loading the page directory again.

VM POOL:

The purpose of vm_pool is to allocate and de-allocate memory whenever the user requests for it. vm_pool has its own data structure (region) where it keeps track of the start address of the memory region and size of the memory region. The first memory region is used to store the management information of other region just like what we did for contiguous frame pool. The vm_pool allocates a page or multiple pages whenever the user request for a memory block.

Detailed implementation:

a) **VMPool () –** This constructor is used to initialize its data structure. The first region is used to store the management information and hence its base address is the start address of the region. And rest of the regions is marked as invalid.
b) **Allocate () –** This functions allocates a page or multiple pages. It keeps track of the start address and the size of the memory in its region and returns the start address of allocated region.
c) **Release () –** This function finds the region corresponding the base address to be released. It then calls free frame of the page table to release the frames from the process frame pool. Later, it modifies the region to make it contiguous.
d) **Legitimate () –** This function checks if the given address is present in any one of the memory regions.