Machine Problem 4: Virtual Memory Management and Memory Allocation

Design :

In this programming assignment, we extend our page table manager implemented in MP3 to handle pages in virtual memory. Our system has a total amount of 32 MB memory. We have kernel space from 0 to 4 MB and the process space from 4MB to 32MB. There also is a 1MB hole of inaccessible memory starting at 15 MB. The kernel space will be direct mapped from virtual memory to physical and shared between process page tables. User space memory will be managed. Both the frames and the pages are 4 KB in size. Here we process a logical address space of 4GB. We use recursive page table lookup to manipulate the last frame of page directory content which will point back to the head of page directory entry itself again. This time instead of getting memory from kernel frame pool, we allocate directory frame and the page table page frames from the process frame pool and they are no longer directly mapped.

For the second part of the assignment, we modify the page table class to add some extra functions needed for virtual memory manager. The register pool function needs to maintain a list of memory pools which have been created till now. This uses a list of VMpool of certain size, defined in the PageTable class. We also maintain the number if VM pools register in another variable. Next, we add the check for address in the page fault handler. The fault in any address should be within the region of VM pool that we are maintaining. The same is implemented in the page fault function. If this fails, we assert the same and abort.

Finally, for the third part of the assignment, I tried to implement vmpool. The idea is to allocate regions of different sizes. Each region consists of two main information and I store them as a structure: (1) base address; (2) the size of the region. We always allocate regions in multiples of pages. Whenever a virtual memory pool releases a region, I will call the release_frames method to notify the page table that the pages can be released. While constructor initializes the variables, the allocator creates region. Regions in this case are similar to a linked list but in the form of array. The end of each region is the beginning of another. Referring to is_legitimate method, it just simply checks if the address is within the boundaries for all pools. Note that this method will be called in page_fault handling process make it really easy to identify the address satus. When a request of

allocating (new) is made, allocates try to find the nearest big enough region. On the other hand, release change the availability variable of a region and reconstruct the array.

Functions used in vm_pool.C/H :

- We define the following struct for region in vm_pool.H:

```
struct region {
unsigned long base_address;
unsigned long size;
};
```

- VMPool constructor

```
base_address = _base_address; // _base_address is the logical start address of the pool
size         = _size;         // _size is the size of the pool in bytes
frame_pool   = _frame_pool;   // _frame_pool points to the frame pool that provides the virtual memory pool with physical memory frames
page_table    = _page_table;    // _page_table points to the page table that maps the logical memory references to physical addresses
regions = (struct region*)(base_address);
region_no    = 0;
page_table -> register_pool(this);
```

- Allocate

```
regions[region_no].base_address  = address + Machine::PAGE_SIZE ;
regions[region_no].size          = num_frames*(Machine::PAGE_SIZE) ;
region_no++;
```

- Release

  We find the region with the particular start address and free pages

```
page_table->free_page(_start_address);
start_address += PageTable::PAGE_SIZE;
regions[i] = regions[i+1];
region_no--;
```

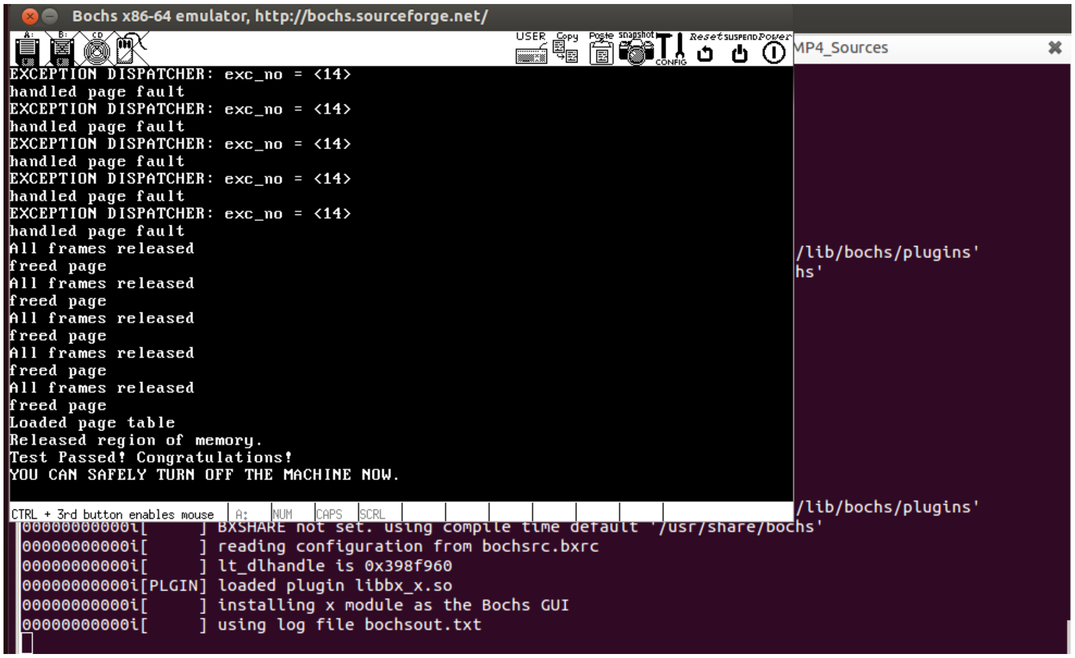
- Is_legitimate

```
unsigned long addr_limit = base_address + size;
unsigned long base       = base_address;
if ((_address < addr_limit) && (_address >= base)) { return 1 }
```

NOTE : No changes have been made to cont frame pool.H and cont frame pool.C from MP2.

OUTPUT :

Test for VM Pools:



Test for Page Table :