CSCE 611: MP4: DESIGN DOCUMENT
AUTHOR: SRIVIDHYA BALAJI
UIN: 827007169

PROBLEM STATEMENT:

To extend and implement a virtual memory Manager which is capable of supporting large address spaces, and a simple Virtual Memory Allocator.

PART:1 SUPPORT FOR LARGE ADDRESS SPACES

By Directly mapping the page entries, there is a limit in the size of the Page table that is used. Hence in order to extend the usage for larger address spaces, the Process Memory Pool is used to allocate memory to the Page Directory and Page Table. Since the CPU generates Logical address randomly, in order to map these address to the frames, the Recursive Page Table approach is used.

Recursive Page Table is where, the last entry of the Page Directory is made to point to itself. On implementing Recursive Page Table, the Page Directory stored in Virtual Memory can be manipulated as,

| 1023 : 10 | 1023 : 10 | offset : 12 |

and the Page Table is manipulated as ,

| 1023 : 10 | X : 10 | Y : 10 | 0 : 2 |

FUNCTIONS USED:

1) **PageTable::PageTable() :** This is the constructor. The direct mapping which involves mapping the addresses and setting the bits to be present and the Page directory entries to be not present happens in the constructor. The recursive setting of the Page Table is also done in the constructor, where the last entry (1023) is made to point to itself.

   PSEUDO CODE:

```
page_directory = kernel_mem_pool->getframes(1)*PAGE_SIZE
page_directory[1023] = page_directory |3                    //RECURSIVE SETTING
Page_table = kernel_mem_pool->getframes(1)*PAGE_SIZE
address = 0
For i=0 to 1024
   Pagetable[i] = address |3
   address = address + PAGE_SIZE
PageDirectory[0] = Page_table |3
For i= 1 to 1023
   PageDirectory[i] = 2  // Marking remaining entries as not present
```

2) **PageTable::load() :** Once the page_table is setup, here the current object's page directory index is extracted and stored in the PTBR ie CR3 register

PSEUDO CODE:

```
Current_page_table = this;
Write_cr3(current_page_table->Page_Directory)
```

3) **PageTable::enable_paging() :** Next the paging is enabled by setting a specific bit in the CR0 register.

PSEUDO CODE:

```
Write_cr0(read_cr0 | 0x80000000)
Paging_enabled = true
```

4) **PageTable::handle_fault(REGS * _r)**
   - Finds the faulty address
   - Find a free frame from the process Frame Pool to map the address in Page Directory and Page Table. Map the logical addresses based on the recursive page table strategy and map frames to resultant offset spaces.
   - Page Fault handler returns
   - Next time it doesn't raise a fault as the corresponding logical memory exists

PSEUDO CODE

```
    address = read_cr2();
* ptr_page_dir = current_page_table->page_directory;

 /*check legitimate : will be explained in PART 2 */

 obtained_page_dir_index = First 10 bits of address
 obtained_page_table_index = Next 10 bits of address

 if((ptr_page_dir[obtained_page_dir_index] & 1) == 0)
       page_table =  kernel_mem_pool->get_frames(1) *PAGE_SIZE
       directory_entry = set the first 20 bits as F. Indicating 1023 | 1023
       directory _entry[obtained_page_dir_index] = page_table |3;

page_table_entry = process_mem_pool->get_frames(1) * PAGE_SIZE
page_entry = of the form 1023 | x y | offset
page_entry [obtained_page_table_index] = page_table_entry |= 3;
```

The first if checks if the entry is not present in the Page Directory and allocates a frame . The second if checks if the entry is not present in the Page Table and allocates a frame for the Page_Table_entry.

TESTCASES:

The fault handler was evaluated for two cases.

1) Fault address = 4MB
2) Fault address = 8MB

For both these the given test suite had passed thus successfully managing the memory.

PART:2 REGISTERATION OF VIRTUAL MEMORY POOLS AND LEGITIMACY CHECK OF LOGICAL ADDRESS:

The Virtual Memory Pool contains a list of regions for virtual memory allocation. The Page table knows about the Pools registered with the Page Table, through the PageTable::register_pool() function. Under this function a Linked list of the VM pools is maintained. When a Page Fault occurs, it is checked if the fault address is legitimate , by checking if the fault address is within the base address and base address + limit range and if so handle page fault.

FUNCTIONS USED:

1) **PageTable::register_pool(VMPool * _vm_pool)**

   PSEUDO CODE :

   ```
   Register_Pool(VMPool *_vm_pool)
   If PageTable::VMPoolList_HEAD = NULL
       PageTable::VMPoolList_HEAD =vm_pool
   else
       for From HEAD to END of Linkedist,
     ptr->vm_pool_next_ptr = _vm_pool
   ```

2) **VMPool::is_legitimate(unsigned long _address)**

   PSEUDO CODE:

   ```
   is_legitimate(_address)
   if address >  base_address + size || address < base_address
    return false
   else
   return true
   ```

The is_legitimate function checks if the given address is between the base_Address and base_address+size of the corresponding VM_POOL. If so, it returns true and the Fault Handler handles the fault accordingly.

TEST_CASES:

1) Registering a pool such that the FAULT ADDRESS 4MB falls within the address range. The Legitimate function returns true and handles the fault and the corresponding address pool is registered.
2) Registering a pool such that it does not contain the FAULT ADDRESS 4MB in its range size . The Legitimate returns false and the code is asserted.

PART:3 ALLOCATION AND DE-ALLOCATION OF THE VM_POOLS

A simple virtual memory Manager is implemented which allocates and de-allocates memory in multiples of pages in the Virtual Memory. In the allocate function, a regions of virtual memory pool is allotted. Once allotted it returns the start address of the regions from which memory was allotted. Here, arrays are used for allocation and de-allocation of these regions. Each array element is of a class object type, which contains details regarding the base address and the length of the VM_Pool region allotted. The class is defined as follows :

1) DATA STRUCTURE USED FOR VM_POOL REGIONS

```
class virtual_memory_region{
 public:
        unsigned long base_address; // represents the base_Address
        unsigned long length;  //represents the size
 };
```

2) ALLOCATION OF VM_POOL REGIONS:

When a new region is requested to be allocated, a new element is added to the array. Each array element is of class object type, which contains details regarding the base_address and the length of theVM_Pool region allocated. The length of the region is the number of pages required multiplied by the Page_Size. Once a region is allocates, the remaining size of the Pool and the count of the regions is kept track of, in order to implement the next set of allocation.

PSEUDO CODE:

```
VMPool::allocate(unsigned long _size)
 If _size > remaining_size
     PRINT No sufficient space for region
Num_pages = _size/PAGE SIZE + (1/0) // Depending on _size %PAGE_SIZE
regions[region_count].base_address = regions[region_count -1].base_address + regions[region_count -1].length
regions[region_count].length = num_pages*PAGE_SIZE
```

region_count = region_count +1

remaining_size = remaining size – num_pages*PAGE_SIZE

return regions[region_count -1].base_Address

/* returns the base_address of the new region added in the array of regions*/

3) DE-ALLOCATION OF VM_POOL REGIONS

When an address is requested to be released, its corresponding region is found to get its base_address and number of pages. For each page , free_page(page_num) function in the Page_table is called. This function calculates the corresponding start frame number allotted for this page and calls Release_frame_pool function from the ContFramePool Manager, which releases the set of frames allotted to this page. This way the physical memory allocated for the regions is released. Here, the corresponding array element is found. The number of pages is calculated and for each page the free_page function is called. Then the corresponding page_entry is marked invalid and the TLB is flushed, ie CR_3 reloaded.

FUNTIONS USED:

1) **VMPool::release(unsigned long _start_address)**

   This finds the corresponding region and calculates the number of pages to be released and for each page calls the release_page function.

   PSEUDO_CODE:

   VMPool::release(unsigned long _start_address)
   For from 0 to region count

   Find index of array storing the base_address.

   Num_pages = region[index].length/PAGE_SIZE

   While Num_pages >0

      Page_table-> free_page(_start_Address)

      Num_pages = Num_Pages -1

      _start_address = _start_address + PAGE_SIZE

2) **PageTable::free_page(unsigned long _page_no)**

   For each page, based on the address, finds the corresponding frame number and calls the corresponding release_frame function of the Process pool. The page is marked invalid and the TLB is reflushed.

   PSEUDO CODE:

   PageTable::free_page(unsigned long _page_no)

   Calculate corresponding start_fram_number using bit manipulation

```
process_mem_pool->release_frames(frame_no);
page_table_entry[obtained_page_table_index] |= 2; // MARK INVALID

load() //  Flushes TLBS
```

TESTCASES:

1) In the Kernel.C using the implementation of the operator new, call the VM_Pool's allocate function to allocate memory of _size < remaining size
2) In the Kernel.C using the implementation of the operator new, call the VM_Pool's allocate function to allocate memory of _size > remaining size, the pool returns saying no space to allocate
3) Use delete in Kernel. C test function, to release the memory allotted. The corresponding Pages, frames are released and the test case is successful.

LIMITATION OF ARRAY IMPLEMENTATION

If the regions are allocated in the form of an array, it suffers a limitation that if any region is freed , i.e. released in between, then the array element or that particular region is un-used. Since the memory space allotted in the current machine problem, is ample enough the fragmentation created due to the use of the array for allocation of Virtual Memory regions, does not hinder the allocation of new VM_Pool regions.

DESIGN IMPROVEMENT

This fragmentation can be overcome by using a LinkedList where two lists, one for Free_Memory regions and other for Allocated Memory_regions can be used, thus making it robust.