CSCE611: MP3: Design Document

Pranav Anand Taukari

UIN: 433003781

PROBLEM STATEMENT:

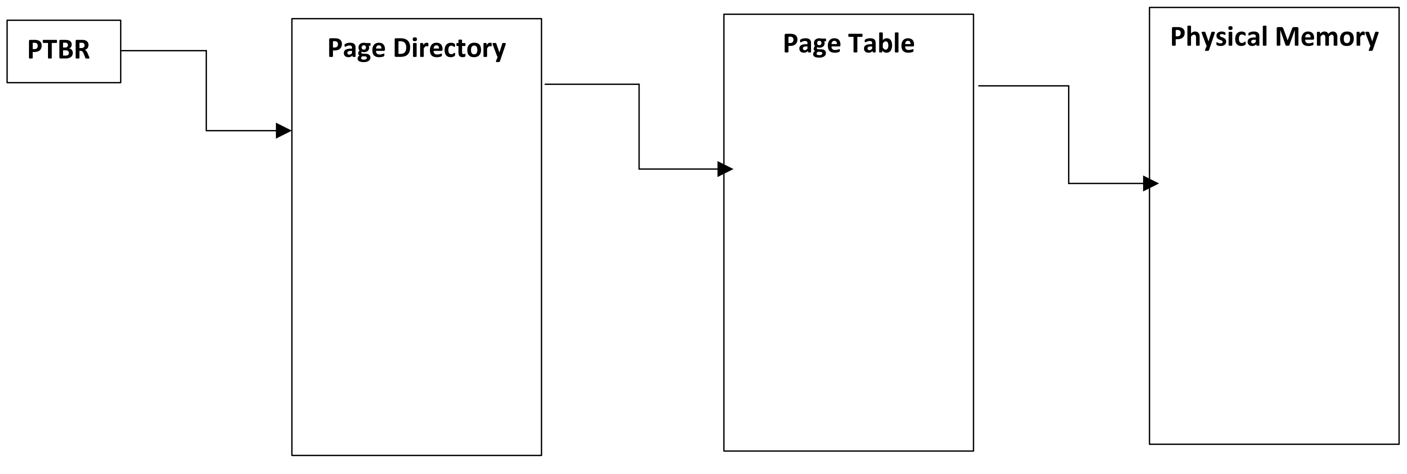
To set up and initialize the paging system and page table infrastructure.

Constraints:

1. Total Memory in the machine = 32 MB
2. The first 4MB are reserved for Kernel
3. Memory within the first 4MB directly mapped to physical memory
4. The next 28MB is used by user processes

Design:

The overall schematic involves Multilevel paging in the x86 system. Here we incorporate two-level paging. Level: 1 has a page directory, Level: 2 has a Page table



PDE (10 bits)	Page Number (10 Bits)	Page Offset (12 bits)

PAGE FAULT: A page fault occurs when the physical memory cannot be mapped in either the page directory (level 1) or the page table (level 2). In this case, a frame is allocated to the corresponding address bits thus creating a corresponding logical memory for the address in the Page table system

IMPLEMENTATION:

PART 1: DIRECT MAPPING OF SHARED MEMORY

The PageTable constructor sets up entries in the page directory and the page table. The page table entries are directly mapped to the address and its attributes are set to supervisor level, read/write and present. The remaining PageDirectory entries apart from the first one are marked as not present. To mark an entry as present the last bit is set to 1, and to mark it as not present the last bit is set to 0. Once this is done, the page_table is loaded, and its address is written in the CR3 REGISTER. This stores the starting address of the Page Table directory. It's also called as the Page Table Base Register. Once the Page table is setup for the shared memory the paging is enabled.

1. Init_paging – we initialize the private variables of the PageTable class.
2. Constructor- we first create the page table for directly mapped memory. This is determined by the shared memory size. We mark all the pages as present in this memory.
3. Load: this API load the current object of PageTable on the current_page_table variable and writes the address of page directory on cr3 register by using write_cr3
4. Enable_paging: this sets the private variable enabled_paging as true and marks the cr0 register by the specified value.

PART 2: MEMORY BEYOND 4MB , PAGE FAULT AND HANDLING:

Memory pages above 4MB have no physical memory associated with them. So, if tried to be accessed, it triggers an EXCEPTION 14, i.e. a Page Fault occurs and the Page Fault Handler needs to handle it. When a Page fault occurs the register CR2 contains the faulty address. In order to handle the fault, it is necessary to read the address for which a Page fault has occurred, get the corresponding Page_Directory and Page_Table indexes from the address bits and see if it is mapped to the Page Directory and Page Table. If not mapped, request for a frame and map it in the Page Directory and Page Table respectively. So once this mapping is done, next time when the address is accessed, it won't create a Page fault since its corresponding Logical address entries are mapped to the two-level Paging system

Testing

This problem was tested by the already written test suite in kernel.C.

I modified the number of pages being accessed to more than 27 MB and the program was able to throw an error. Similarly, if I request any memory less than 27 MB, which is the total available memory, it successfully allocates that.

The continuous frame pool implemented in MP2 was utilized in MP3 and was found to function properly with no errors.