825008857 Chaiwei Chang

# MP3

## Design for page table

We follow the implementation of page table management on x86 which is two layer paging system with level one: page directory entry and page two: page table entry. The page table maps virtual memory into physical memory by utilizing previously created contiguous frame pools of MP2. We are asked to have a page directory which is a pointer to 1024 pages and a page which is a pointer to 1024 places in memory. For any page table the initialization of page directory's first entry is a page table that points to the first 1024 pages in memory which translates to the first 4 MBs of our kernel. While we initialize the variable in the constructor, we have to make sure that regardless whether paging is on or not, the first 4MB has to map exactly the same frame number which is our phys address here. This is how we implement the direct mapping and sharing of the kernel space. The rest of the pages are demand paged for user level other than previous supervisor level. Additionally, virtual addresses are mapped to physical addresses only when we need them to, this uses the process frame pool get frame function to do so.

To the more detail, the Constructor initializes the page directory with a page table that points to the first 4 MBs as mentioned previously. It marks all the other page tables with the not in memory flag 010 I learnd this from tutorial. The page table load function sets the current page table to this and then writes the page directory pointer into the Cr3 register which will help us easily access page directory when needed as a page directory base register. Also, the page table enable_paging function sets paging enabled to 1 and then sets the 32$^{nd}$ bit of the Cr0 register to 1, this enables the MMU on the hardware to do paging.

Finally, all of the handle_fault function is not documented in the tutorial which is the main issue of this task MP3. With all the page initializing variable set up, whenever we encounter page fault exception we can get these information by setting the Cr0 set bit and loading cr3. First the fault handler reads the page directory from the Cr3 register and the desired access address from the Cr2 register. And then secondly, it checks the error code of the handler to determine if we indeed have a not in memory fault. I have record all the 0~7 possibility fault inside my code. If this page fault happed in memory we simply create a new page entry in the page table

by calling the process mem pool get additional frame and loading it into the page table index of our address. If it is NOT in memory we first create a new page table entry in our page directory by calling the kernel mem pool get frame and storing it in the page directory, and then subsequently loading in a new page as previously mentioned. The way the address read from the Cr2 register is interpreted, is the first 10 bits is the index of the page table in the page directory, while the second 10 bits is the index of the page in the page table to be accessed and lastly the last 12 bits is the offset of the pages (4KB) and the flags. Finally, I wrote some command on my code, but if grader of TA have any question about my logic feel free to contact me at hunkywei@tamu.edu