# Machine Problem 3 : Page Table Management

**Implementing PageTable class to enable paging system**

**Operating Systems : CSCE 611**

**Garuda Suma Pranavi : UIN 926009146**

Design

- Total amount of machine memory is 32MB.
- Kernel code and data : first 4MB which is directly mapped
- Process pool : 4MB – 32MB with an inaccessible region between 15-16MB
- Frames and pages are of the same size. Fixed size blocks of physical memory are frames and fixed size blocks of virtual memory are pages.

Since, we only access/get one frame at a time, I used the simple frame pool function given to us in MP2

In the exercise, I have implemented a 2 level page table:

- The first or outer page table is the page directory and contains pointers to page tables. It is of 4KB and hence consists 1024 entries.
- The page table page has 1024 page tables and each page table is able to map to 1024 virtual pages and correspondingly to frames.

Frame address structure:

Each page table entry is a frame address i.e. 32 bit address. Since each frame must be of 4KB, the last 12 bits will always be 0. Hence, we use the first 20 bits to distinguish between frames in memory – the first 10 bits indicate the page table no and the next 10 bits indicate the page number in the page table.

**Steps:**

1. **Initialize** the paging system : Passing the parameters to the private variables.
2. We then deal with direct mapping in **PageTable**() :
    - First, give page directory a frame (from kernel memory pool) to store entries.
    - We then assign a frame to the first page table as it maps from 0-4MB and this is just our kernel memory address.

- Then, we initialize the page table by filling it with the real physical address starting from 0KB.
- We also add the flags for each entry as 3 – 011 – marked as present.
- Then, assign the first page table address to the first page table directory entry.

3. **PageTable::load()** :
    - We put the address of the page directory in CR3.
4. **PageTable::enable_paging():**
    - To enable paging, we just have to set the CR0 to 1.
5. **PageTable::handle_fault():**
    - All the page faults happen when our page table is missing entries that are supposed to tell the CPU where the physical address is.

      There are certain statements in kernel.C that assign addresses that have no record in our paging system (addresses larger than 4MB).
    - We can access the fault address through CR2.
    - On AND, if we get 1 – this indicated protection fault and we have a switch statement later to indicate what went wrong as console output statements.
    - If the last bit is 0, it indicates that the page is not present in memory. In this case, we first allocate a frame form the process pool to the page directory table and set it’s flags.
    - The entries of this page directory point to the page tables in the page table page. We initialize all the entries of the page table page and set it’s flag to 010 – write.
    - We then load the page into the page table page.
    - If we have a page in memory, we just get and load it. We have now handled page fault.

Although the logic seems to be correct (according to me), the code faces some problems in handling faults. – exc 14. However, it does enable paging successfully.