CSCE - 611 (Operating Systems)
MACHINE PROBLEM -3

Nitin Chakravarthy Gummidela
(427006656)

The objective of the machine problem was to initialize a 2 level paging system and implement a page fault handler that is called when a page fault exception is raised.
The total memory in the machine is 32MB and the first 4MB(kernel memory) is directly mapped.
We implement the page table and the page fault handler in the **page_table.C** file and the definitions of the static variables and the methods were defined in the **page_table.H** file.
To allocate memory for frames we use the **get_frames()** method **cont_frame_pool.C** file which was implemented in the previous machine problem.

The following section explains each method in the **page_table.c** file.

init_paging(ContFramePool * _kernel_mem_pool,
ContFramePool * _process_mem_pool,
const unsigned long _shared_size) :
In this method we set the global variables for the PageTable class.
We se the _kernel_mem_pool, _process_mem_pool, _shared_size to the vaiables inside the class.

PageTable() : (Constructor):

In this method we initialize the page table directory and page table. Then we direct map the first 4 MB memory to the first page in the page table and then insert this page in the first entry of the page.

- First we get a frame using the get_frames() method. We assign the address to the page_directoy variable.
- Now, similarly we get another frame for the first page_table.
- To populate the entries of the page table starting from 0x0 to till the 4MB finishes. We turn on the last bit(present bit) and the last before bit (read/write bit).
- Then we assign the address of the page table in the first entry of the page directory and turn on the present and the read/write bit.
- In the remaining entries of the page directory we turn on only the read/write bit.

load():
In this method we load the page table directory into cr3 register by calling the **write_cr3** method defined in the paggind_low.asm file.

enable_paging():
In enable paging we turn on the cr2 register which represents if paging is turned on or off.

handle_fault(REGS * _r):
This is the method that gets called when a page fault exception is raised. We can see the handler being defined and being registered to the page fault exception code (14) in the Kernel.C file.

First we obtain the error code using -r -_ err_code. We check if the last bit is 0 or 1. If it is zero, that implies that the exception is due to a page not being present and we handle it. If it is one, the exception is raised due to lack of privileges to access the frame. We do not handle this at this point.

Once we checked the last bit is zero, Now there can be two cases where the error could occur. The error could occur in the page directory or in the page table. That is, there could not be a page table in the page directory entry or a page table could be present but the page table entry does not contain a page.
We handle both the cases by first checking the page directory entry is present or not. If it is present we access the page table. Now since the exception has occurred and we have only 2 levels, the page table entry should be empty. So we get a new frame from the process_pool using the get_frames() method of the process pool and assign it to the page table entry. Further we turn on the present bit and the read/write bit also.
In the second case where the page directory entry is not present, we first get a frame from the kernel_pool and assign it to the directoy_entry. Again we turn on the present and the read/write bit. Now to access the page table from the page directory we need only the first 20 bits in the address (rest are zeros). So we use a mask **0xFFFFF000** to get the address. Now we index into the page table and put a new frame from the process_memory_pool and assign it to the page table entry and turn on the present and read/write bit.

This way the reference can now be accessed and the next time the system reissues the reference, it can find the frame and the page fault would not occur.