# Design Reprot for Machine Problem 3

## Introduction

In the page table class, there are five static member variables, `current_page_table, kernel_mem_pool, process_mem_pool, paging_enabled, shared_size` and one non-static variable `page_directory`. Each page table instance has its own page directory and page table, and all page table instances share these static variables, which are used to allocate resources for different page table instance. During the implementation, I followed the tutorial K.J.'s tutorial on "Implementing Basic Paging" to implement the page table.

## Page Table Initialization

The page table initialization is quite strainght forward, we can pass the kernel frame pool object and process frame object to the static variable `kernel_mem_pool` and `process_mem_pool`. These two variables are the static variables and can be used to allocate memory for page table instances.

## Page Table Constructor

In order to construct the page table class, I followed the above mentioned tutorial. Instead of assigning some random address to the page directory, I used the `kernel_mem_pool->get_frames(1)*PAGE_SIZE)` to get frame memory from the kernel area because page directory and page table are management information and needs to be saved in the kernel area. Similariy, page table can be assigned by the same kernel memory pool. Now we need to map the first 4MB of memory to the page table address. Since the logical address of the first 4 MB is the same as physical address, so we can directly map it to the page table. Each frame size is 4 KB, so 4 MB/4KB = 1024 entries is required, we can divide the 4MB area into 1024 continuous area and assigned the address to this 1024 entries. All addresses are in the supervisor level, read/write, and present, so the last 3 bits are assigned to 011 through `page_table[i] = address | 3`. Also, we need to mark the first entry of the project directory to be occupied because the first page table is created and assigned values to it. So the `page_directory[0] |= 3` is used to mark the presence of first page table. However, the rest of other page tables are not ready to use, so all the presence bit is set to zero by `page_directory[0] |= 2`.

```
for(i=1; i<1024; i++)
{
page_directory[i] = 0 | 2; // attribute set to: supervisor level, read/write, not pre
sent(010 in binary)
};
```

## Page Table Constructor

As mentioned in the instructions, the loading step will assign the current page table to the static variable current_page_table pointer in the page table class. Also, the page directory address needs to be written to the register 3 through `write_cr3((unsigned long)page_directory)`

## Enabling Paging

In this step, the page table is enabled, so we have to set the page table enabling flag to 1 `paging_enabled = 1` and set the paging bit in CR0 to 1 through `write_cr0(read_cr0() | 0x80000000)`

## Fault Handling

The essence of this MP is to first set up the page table correctly and then to implement the method PageTable::handle fault(), which will handle the page-fault exception of the CPU. This method will look up the appropriate entry in the page table and decide how to handle the page fault. If there is no physical memory (frame) associated with the page, an available frame is brought in and the page-table entry is updated. If a new page-table page has to be initialized, a frame is brought in for that, and the new page table page and the directory are updated accordingly.

In the implementation of handling fault, we need to load the page directory address first from register 3 and then find the page address from register 2 through

```
unsigned long *cur_page_directory = (unsigned long*) read_cr3();
unsigned long page_addr = read_cr2();
```

It is already known that the first 10 bits are the address for page table in page directory, and the middle 10 bits are for the page in page table and the last 12 bits are the offset to locate page in physical memory. We can retrieve these information first through

```
unsigned long page_table_index = page_addr>>22;
unsigned long page_frame_index = ((page_addr>>12) & 0x3FF);
```

Now, we need to check the presence of page table through checking the last bit of the corresponding page directory entry

```
if ((cur_page_directory[page_table_index] & 1) != 1) {
```

If the last bit is 1, it means the page table is present, but if it is not 1, we need to initialize the page table first in the kernel memory area and mark the page table bits as present

```
cur_page_directory[page_table_index] = (unsigned long)(kernel_mem_pool->get_frames(1)
*PAGE_SIZE);
cur_page_directory[page_table_index] |= 3
```

Now we can use this address as the page table starting address, but here we need to ensure the address starts every 4 Kb through `page_table &= 0xFFFFF000`. After the initialization of the new page table, we will mark all the page table entries as not being present through

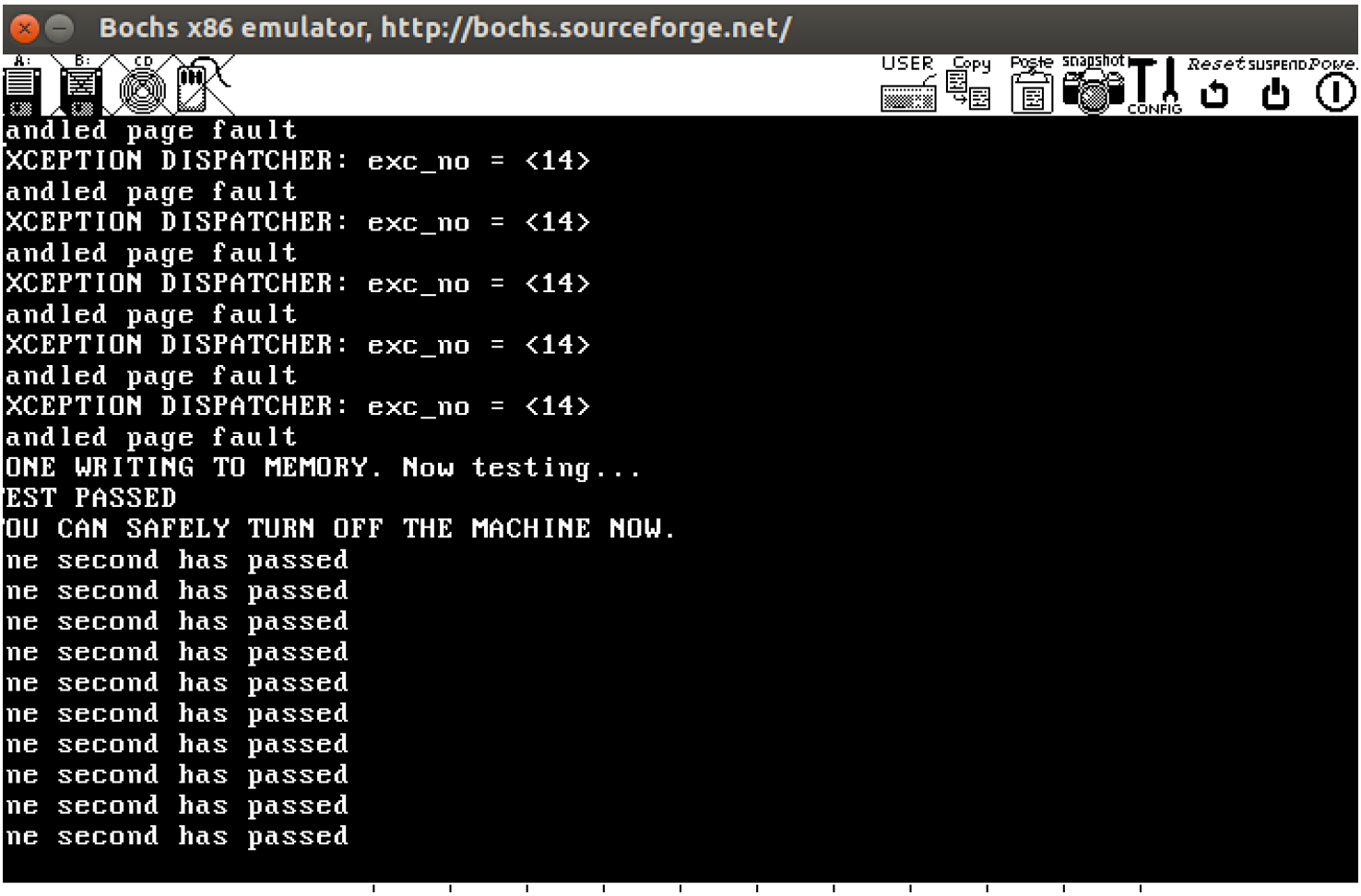
```
for (int i = 0; i < 1024; i++)
       page_table[i] = 0|2;
```

Now, the page sought by the page handler does not exist, so we need to initialize that page again through the process_mem_pool object and mark that page as present. Here, we did not use kernel_mem_pool object because the page we are seeking now is not related with management information and we need to put it in the process area.

```
page_table[page_frame_index] = (unsigned long)(process_mem_pool->get_frames(1)*PAGE_S
IZE) | 3;
```

## Result

After implementation of above code, I got the result


Bochs x86 emulator, http://bochs.sourceforge.net/
A:
B:
CD
USER
Copy
Paste
snapshot
CONFIG
Reset
suspend
Powe.
andled page fault
XCEPTION DISPATCHER: exc_no = <14>
andled page fault
XCEPTION DISPATCHER: exc_no = <14>
andled page fault
XCEPTION DISPATCHER: exc_no = <14>
andled page fault
XCEPTION DISPATCHER: exc_no = <14>
andled page fault
XCEPTION DISPATCHER: exc_no = <14>
andled page fault
ONE WRITING TO MEMORY. Now testing...
EST PASSED
OU CAN SAFELY TURN OFF THE MACHINE NOW.
ne second has passed
ne second has passed
ne second has passed
ne second has passed
ne second has passed
ne second has passed
ne second has passed
ne second has passed
ne second has passed
ne second has passed