Machine Problem 3: Page Table Management

Design :

In this programming assignment, we implement the Paging system similar to that on x86. We have a two layer paging system with level one – page directory entry and level two – page table entry. Our system has a total amount of 32 MB memory. We have kernel space from 0 to 4 MB and the process space from 4MB to 32MB. There also is a 1MB hole of inaccessible memory starting at 15 MB. The kernel space will be direct mapped from virtual memory to physical and shared between process page tables. User space memory will be managed. Both the frames and the pages are 4 KB in size.
Here, we 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 physical 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.

The control registers that we are modifying here :
CR0 : The CR0 register is 32 bits long. CR0 has various control flags that modify the basic operation of the processor. 32nd bit is for PG - Paging -> If 1, enable paging and use the CR3 register, else disable paging.
CR2 : Contains a value called Page Fault Linear Address (PFLA). When a page fault occurs, the address the program attempted to access is stored in the CR2 register.
CR3 : Used when virtual addressing is enabled, hence when the PG bit is set in CR0. CR3 enables the processor to translate linear addresses into physical addresses by locating the page directory and page tables for the current task. Typically, the upper 20 bits of CR3 become the page directory base register (PDBR), which stores the physical address of the first page directory entry.

Functions used in page_table.C :

- **init_paging** - we initialize the variables of the PageTable class.

```
PageTable::kernel_mem_pool  = _kernel_mem_pool;
PageTable::process_mem_pool = _process_mem_pool;
PageTable::shared_size      = _shared_size;
```

- **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. We mark rest of the frames as not present.

```
page_directory = (unsigned long*) (kernel_mem_pool->get_frames(1)*PAGE_SIZE);
unsigned long * page_table = (unsigned long*) (kernel_mem_pool->get_frames(1)*PAGE_SIZE);
unsigned long addr = 0;
page_table[i] = addr | 3;  // 3 -> 011 -> kernel + write + present
page_directory[i] =    2;  // 2 -> 010 -> kernel + write + not present
```

- **load -** Used to 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

```
write_cr3((unsigned long) page_directory);
```

- **enable_paging** - this sets the private variable enabled_paging as true and marks the cr0 register by the specified value.

```
paging_enabled = 1;
write_cr0( read_cr0() | 0x80000000); // update 32nd bit to 1
```

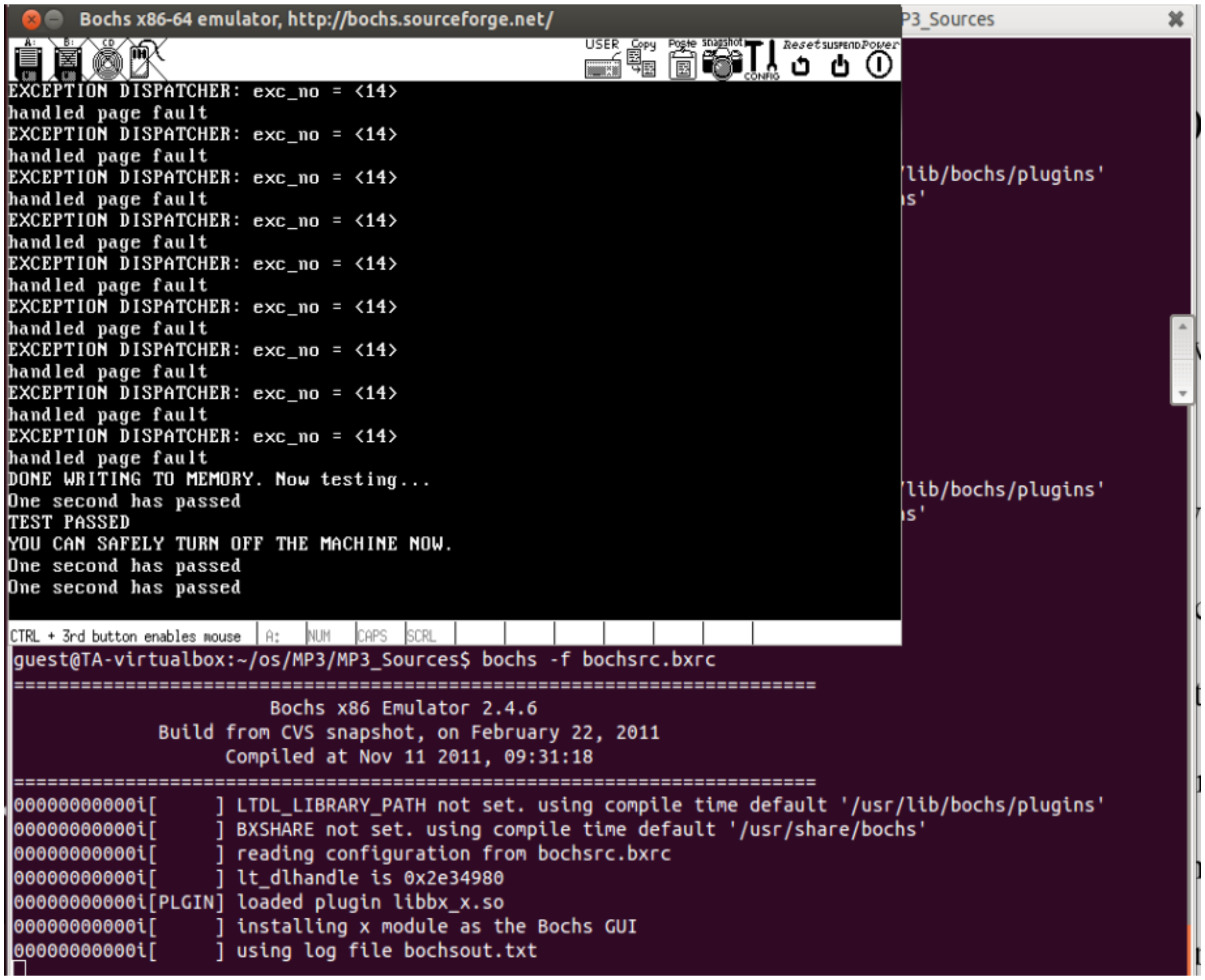
- **Handle_fault -** To handle a page fault, we first check whether the fault is in page directory or page table. This is done by check the entry from page directory whose index is received via the CR2 register. If the fault is in page directory itself, we first allocate a frame for the page table from kernel memory and then the actual page from the process memory pool. If the fault is in the page table, then we allocate a frame for the actual page from the process memory pool.

```
unsigned long * page_dir   = (unsigned long *) read_cr3();
unsigned long   addr       = read_cr2();
unsigned long   error_code = _r->err_code;  // to get the error code
unsigned long * page_tb;
```

```
// 10 bits -> PD, 10 bits -> PT, 12 bits -> offset
unsigned long   page_directory_addr = addr>>22; // first 10 bits
unsigned long   page_table_addr     = addr>>12; // first 20 bits
(error_code & 1) == 0                          ➔ page not present
(page_dir[page_directory_addr] & 1) == 1       ➔ fault in page table
(page_table_addr & 0x03FF)                     ➔ gives the last 10 bits
```

NOTE : No changes have been made to cont frame pool.H and cont frame pool.C from MP2.

OUTPUT :



```
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Now testing...
One second has passed
TEST PASSED
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
```

```
guest@TA-virtualbox:~/os/MP3/MP3_Sources$ bochs -f bochsrc.bxrc
========================================================================
                       Bochs x86 Emulator 2.4.6
              Build from CVS snapshot, on February 22, 2011
                    Compiled at Nov 11 2011, 09:31:18
========================================================================
00000000000i[     ] LTDL_LIBRARY_PATH not set. using compile time default '/usr/lib/bochs/plugins'
00000000000i[     ] BXSHARE not set. using compile time default '/usr/share/bochs'
00000000000i[     ] reading configuration from bochsrc.bxrc
00000000000i[     ] lt_dlhandle is 0x2e34980
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
```

Bochs x86-64 emulator, http://bochs.sourceforge.net/
P3_Sources
USER Copy Paste Snapshot CONFIG Reset SUSPEND Power
CTRL + 3rd button enables mouse
A:
NUM
CAPS
SCRL