

# MP3: Page Manager I

Ashutosh Punyani  
UIN: 834006613  
CSCE611: Operating System

## Assigned Tasks

**Main:** Completed.

## System Design

The main objective of machine problem 3 is the implementation of a demand-paging based virtual memory system where the goal is to initialize the page table for a single process, and therefore, a single address space. This manages kernel frames from 2MB to 4MB and process frames between 4MB and 32MB.

## Code Description

During the implementation of this machine problem, I made changes to the following three files:

1. **cont\_frame\_pool.H**
2. **cont\_frame\_pool.C**
3. **page\_table.C**

I used the same code that I implemented for MP2. In this machine problem, I implemented the following function definitions in **page\_table.C**:

1. **init\_paging** : This function is used to initialize static private data members in a class, including the kernel frame pools, memory frame pools, and the shared size for the page table.

```
void PageTable::init_paging(ContFramePool *_kernel_mem_pool,
                             ContFramePool *_process_mem_pool,
                             const unsigned long _shared_size)
{
    // initializing basic data structure for paging
    Console::puts("Initialized Paging System Start\n");
    kernel_mem_pool = _kernel_mem_pool;
    process_mem_pool = _process_mem_pool;
    shared_size = _shared_size;
    Console::puts("Initialized Paging System End\n");
}
```

Figure 1: **init\_paging**

2. **PageTable (Class Constructor)** : The constructor is used to construct the page table object. It initializes the first page directory by assigning the free frame from the kernel frame pool and marking it as valid (present). It also initializes the first page table by assigning the free frame from the kernel frame pool. The first page table is directly mapped to physical memory. The first directory entry holds the first page table, while all the remaining directories are marked as invalid (not present).

```

PageTable::PageTable()
{
    Console::puts("Constructed Page Table object Start\n");
    unsigned long page_directory_frame_number = kernel_mem_pool->get_frames(1);
    page_directory = (unsigned long *) (page_directory_frame_number * PAGE_SIZE);

    unsigned long page_table_frame_number = kernel_mem_pool->get_frames(1);
    unsigned long *page_table = (unsigned long *) (page_table_frame_number * PAGE_SIZE);

    // mapping the first 4M of memory
    for (unsigned int i = 0; i < ENTRIES_PER_PAGE; i++, physical_address += PAGE_SIZE)
    {
        // attribute set to: supervisor level,
        // read/write, present(011 in binary)
        page_table[i] = physical_address | 0x3;
    }

    // attribute set to: supervisor level,
    // read/write, present(011 in binary)
    page_directory[0] = (unsigned long)page_table | 0x3;

    for (unsigned int i = 1; i < ENTRIES_PER_PAGE; i++)
    {
        // attribute set to: supervisor level,
        // read/write, not present(100 in binary)
        page_directory[i] = 0 | 0x2;
    }

    current_page_table = this;
    Console::puts("Constructed Page Table object End\n");
}

```

Figure 2: **PageTable (Class Constructor)**

3. **load** : This function loads the current page directory into register CR3 using `write_cr3()`. The page table is loaded.

```

void PageTable::load()
{
    Console::puts("Loaded page table Start\n");
    write_cr3((unsigned long)page_directory);
    Console::puts("Loaded page table End\n");
}

```

Figure 3: **load**

4. **enable\_paging** : This function is used to enable paging by setting the paging bit (bit 31) of CR0 to 1 using `read_cr0` to read the contents of CR0 and `write_cr0` to write the contents to CR0. Also, we set the boolean `paging_enabled` to true. Before enabling the paging, the page directory and page table should be set up and loaded correctly.

```

void PageTable::enable_paging()
{
    Console::puts("Enabled paging Start\n");
    unsigned long cr0_reg = (unsigned long)(read_cr0() | 0x80000000);
    paging_enabled = 1;
    write_cr0(cr0_reg);
    Console::puts("Enabled paging End\n");
}

```

Figure 4: **enable\_paging**

5. **handle\_fault** : This function is utilized to handle raised faults. It examines the error code received from the register through bit manipulation. Subsequently, it retrieves the faulty address from CR2 register using the `read_cr0` function. Additionally, it reads the current page directory from CR3 using the `read_cr3` function. If the current page directory is an invalid page directory entry, the function first obtains a free frame from the kernel frame pool, assigns it to the current page directory, and marks it as 'present.' Following this, it obtains another free frame from the kernel frame pool and assigns it to the new page table, initializing all its entries as invalid. Simultaneously, the address pointing to that page location is assigned a free frame from the process frame pool and marked as 'valid' (present). If the page directory is valid, the function employs an existing page table and assigns a free frame from the process frame pool, marking it as 'valid' (present). In the event of a different scenario than the aforementioned two, the function halts execution with a message stating that 'something went wrong'.

```

void PageTable::handle_fault(REGS *_r)
{
    Console::puts("handle_fault Start\n");
    unsigned long err_code = _r->err_code;
    if ((err_code & 0x1) == 0x0)
    {
        Console::puts("handle_fault err_occured\n");
        unsigned long faulty_address = (unsigned long)(read_cr2());
        unsigned long *page_directory_list = (unsigned long *) (read_cr3());
        unsigned long directory_location = (faulty_address & 0xFFC00000) >> 22;
        unsigned long page_location = (faulty_address & 0x003FF000) >> 12;
        bool page_table_fault = false;

        if ((page_directory_list[directory_location] & 0x1) == 0x0)
        {
            Console::puts("directory issue and new page table");
            Console::puts("\n");
            unsigned long new_page_table_frame_number = kernel_mem_pool->get_frames(1);
            unsigned long *new_page_table = (unsigned long *) (new_page_table_frame_number * PAGE_SIZE);

            // attribute set to: supervisor level,
            // read/write, not present(010 in binary)
            page_directory_list[directory_location] = (unsigned long) new_page_table | 0x3;

            // Initializing the page table entries
            for (unsigned int i = 0; i < ENTRIES_PER_PAGE; i++)
            {
                // attribute set to: supervisor level,
                // read/write, not present(010 in binary)
                new_page_table[i] = 0 | 0x2;
            }

            page_directory_list[directory_location] = (unsigned long) new_page_table | 0x3;

            // Initializing the page table entries
            for (unsigned int i = 0; i < ENTRIES_PER_PAGE; i++)
            {
                // attribute set to: supervisor level,
                // read/write, not present(010 in binary)
                new_page_table[i] = 0 | 0x2;
            }

            unsigned long physical_frame_number = process_mem_pool->get_frames(1);
            new_page_table[page_location] = (unsigned long) (physical_frame_number * PAGE_SIZE) | 0x3;
        }
        else
        {
            Console::puts("existing page table issue");
            Console::puts("\n");
            unsigned long *existing_page_table = (unsigned long *) (page_directory_list[directory_location] & 0xFFFFF000);
            unsigned long physical_frame_number = process_mem_pool->get_frames(1);
            existing_page_table[page_location] = (unsigned long) (physical_frame_number * PAGE_SIZE) | 0x3;
        }

        Console::puts("resolved page fault\n");
    }
    else
    {
        Console::puts("Something went wrong\n");
        assert(false);
    }
    Console::puts("handle_fault End\n");
}

```

Figure 5: handle\_fault

## Testing

During the development of the code, I wrote several `Console::puts()` and `Console::putui()` statements to identify where my code was breaking and to understand if the logic was incorrect or not performing as expected. During testing, the execution was running infinitely while running the program. Also, during testing, test case 0 was failing. After making changes and understanding the console debug logs, the code ran successfully. I removed all Console statements and changed back the Kernel.C to the original one.

```
make: Nothing to be done for 'all'.
=====
Bochs x86 Emulator 2.7
Built from SVN snapshot on August 1, 2021
Timestamp: Sun Aug 1 10:07:00 CEST 2021
=====
0000000000i[ ] BXSHARE not set. using compile time default '/usr/local/share/bochs'
0000000000i[ ] reading configuration from bochsrc.bxrc
0000000000i[ ] installing x module as the Bochs GUI
0000000000i[ ] using log file bochsout.txt
Installing handler in IDT position 0
Installing handler in IDT position 1
Installing handler in IDT position 2
Installing handler in IDT position 3
Installing handler in IDT position 4
Installing handler in IDT position 5
Installing handler in IDT position 6
Installing handler in IDT position 7
Installing handler in IDT position 8
Installing handler in IDT position 9
Installing handler in IDT position 10
Installing handler in IDT position 11
Installing handler in IDT position 12
Installing handler in IDT position 13
Installing handler in IDT position 14
Installing handler in IDT position 15
Installing handler in IDT position 16
Installing handler in IDT position 17
Installing handler in IDT position 18

Installing handler in IDT position 44
Installing handler in IDT position 45
Installing handler in IDT position 46
Installing handler in IDT position 47
Installed exception handler at ISR <0>
Installed interrupt handler at IRQ <0>
Installed interrupt handler at IRQ <1>
Installed exception handler at ISR <14>
Initialized Paging System Start
Initialized Paging System End
Constructed Page Table object Start
Constructed Page Table object End
Loaded page table Start
Loaded page table End
Enabled paging Start
Enabled paging End
WE TURNED ON PAGING!
If we see this message, the page tables have been
set up mostly correctly.
Hello World!
EXCEPTION DISPATCHER: exc_no = <14>
handle_fault Start
handle_fault err_occured
directory issue and new page table
resolved page fault
handle_fault End
EXCEPTION DISPATCHER: exc_no = <14>
handle_fault Start
handle_fault err_occured

EXCEPTION DISPATCHER: exc_no = <14>
handle_fault Start
handle_fault err_occured
existing page table issue
resolved page fault
handle_fault End
EXCEPTION DISPATCHER: exc_no = <14>
handle_fault Start
handle_fault err_occured
existing page table issue
resolved page fault
handle_fault End
EXCEPTION DISPATCHER: exc_no = <14>
handle_fault Start
handle_fault err_occured
existing page table issue
resolved page fault
handle_fault End
DONE WRITING TO MEMORY. Press keyboard to continue testing...
One second has passed
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed

Bochs is exiting with the following message:
[XGUI ] POWER button turned off.
=====
csce410@csce410-VirtualBox: ~/Desktop/AshutoshPunyanl_CSCE611/MP3$
```

Figure 6: Testing