CSCE611 Machine Problem 3 Page Table Management

Gehao Yu 629008717

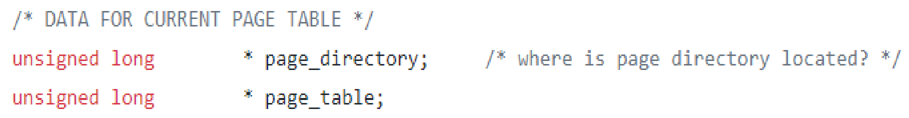
Mar 3 2021

1 Introduction

We have to design a page table management based on the requirement of the machine problem 3.

2 Modification in page_table.H

Only add a variable called page_table to store the information of page table.

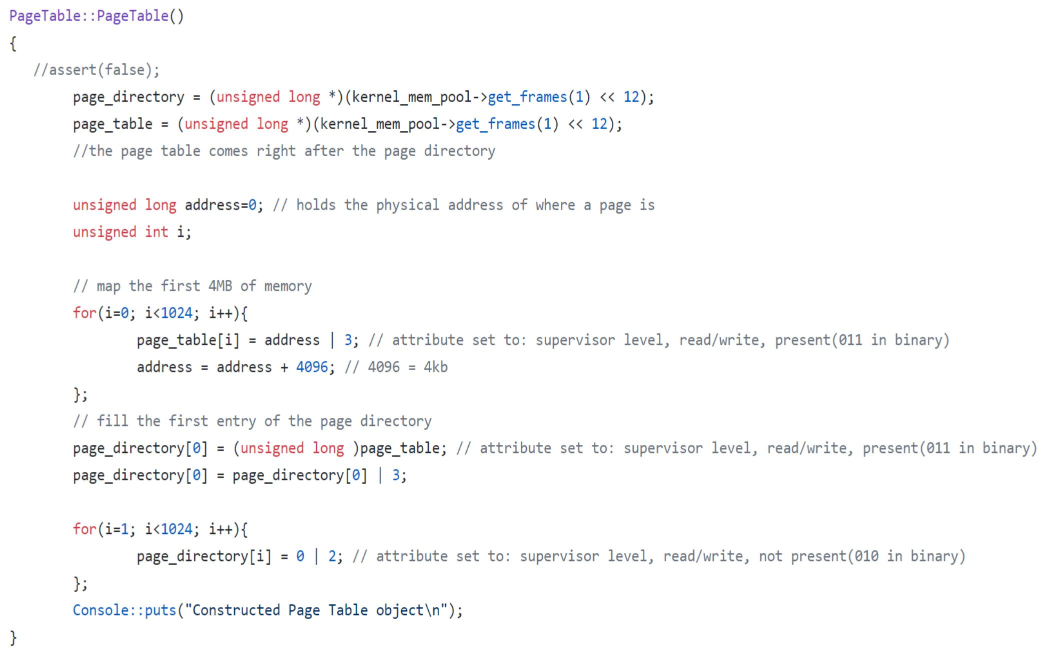


```
/* DATA FOR CURRENT PAGE TABLE */
unsigned long      * page_directory;    /* where is page directory located? */
unsigned long      * page_table;
```

3 Initialization Process

Once we need to initialize the paging process, we would assign all the input into variables of Page Table management system. Considering the requirement of Machine Problem 3, we need to create a page directory and several page tables. All these data structures are in kernel pool framework, including the management information of process frame pool and kernel.
We would initialize the page directory and then page tables. the page directory would map the first 4MB in the kernel(1024 pages). By using the get_frame function, we could map the virtual address into the physical address.
Then we initialize the page tables, and let each entry in the page directory pointing to one page. For the first page table, the directory would label it as "Present"(011 in last 3 digits) and rest page tables, the directory would label

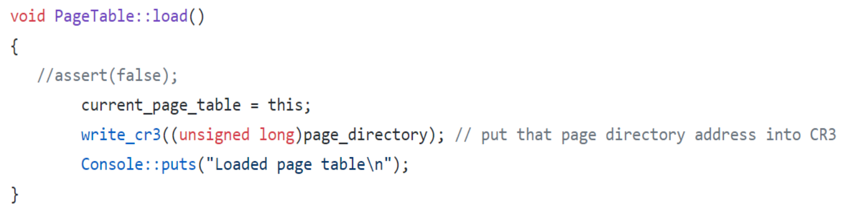
as "Not Present"(010 in last 3 digits).



```
PageTable::PageTable()
{
   //assert(false);
        page_directory = (unsigned long *)(kernel_mem_pool->get_frames(1) << 12);
        page_table = (unsigned long *)(kernel_mem_pool->get_frames(1) << 12);
        //the page table comes right after the page directory

        unsigned long address=0; // holds the physical address of where a page is
        unsigned int i;

        // map the first 4MB of memory
        for(i=0; i<1024; i++){
                page_table[i] = address | 3; // attribute set to: supervisor level, read/write, present(011 in binary)
                address = address + 4096; // 4096 = 4kb
        };
        // fill the first entry of the page directory
        page_directory[0] = (unsigned long )page_table; // attribute set to: supervisor level, read/write, present(011 in binary)
        page_directory[0] = page_directory[0] | 3;

        for(i=1; i<1024; i++){
                page_directory[i] = 0 | 2; // attribute set to: supervisor level, read/write, not present(010 in binary)
        };
        Console::puts("Constructed Page Table object\n");
}
```

4 Function load()

We will take the given page table as the current page table and write the page directory into CR3(register 3).



```
void PageTable::load()
{
   //assert(false);
        current_page_table = this;
        write_cr3((unsigned long)page_directory); // put that page directory address into CR3
        Console::puts("Loaded page table\n");
}
```

5 Function enable_paging()

To enable paging, we could set the 31st digit on CR0(register 0) to 1 by using read_cr0() function and re-write it back.

```
void PageTable::enable_paging()
{
   //assert(false);
        paging_enabled = 1;
        write_cr0( read_cr0() | 0x80000000); // set the paging bit in CR0 to 1
        Console::puts("Enabled paging\n");
}
```

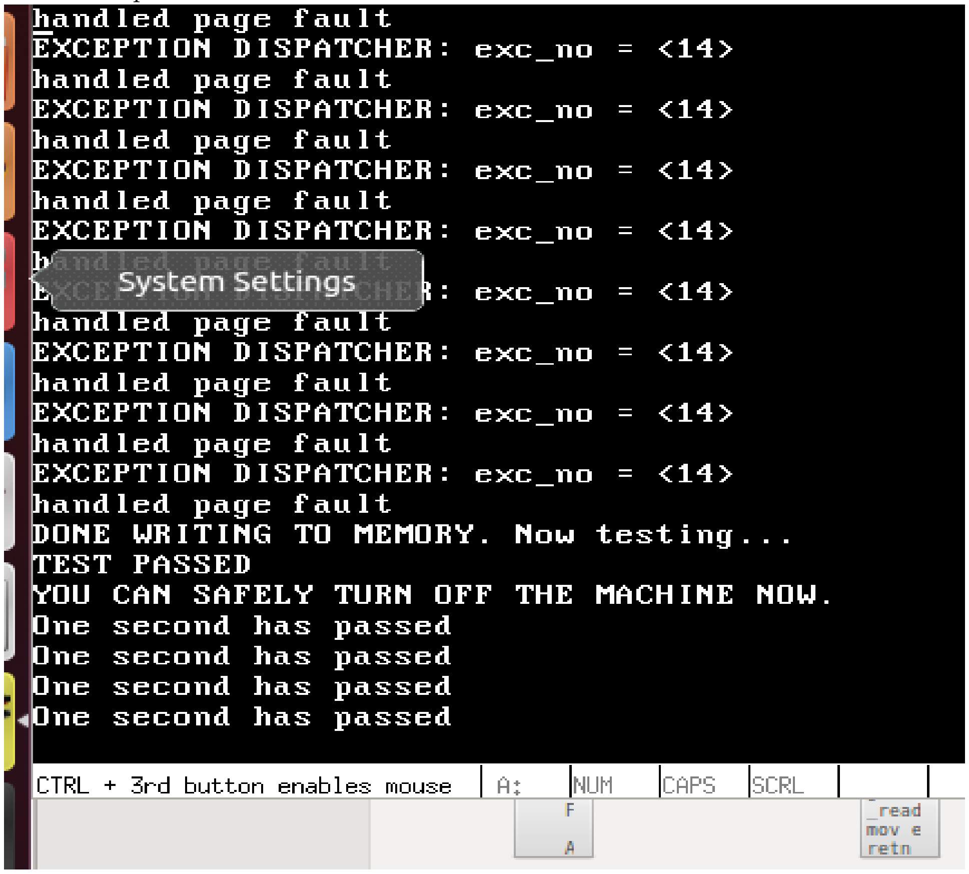
6 Function handle_fault()

When a fault happened, we should try to handle it. First, we should get the current page directory and error virtual address from CR3 and CR2 by read_cr3() and read_cr2() functions.
There are in total 32 digits of the virtual address we got, the first 10 digits are the offset of page directory, the second 10 digits are the offset of the page table and the last 12 digits are the offset of page table entry.
If the offset of page directory for the address told us that the corresponding page table is "Not Present", which means the last 3 digits are "010", we should create a new table by finding a new frame in the frame pool and repeat what we have done in the initialization process of page table. After these, we updated the corresponding current page directory entry, setting the last 3 digits to "011".
If the offset of page directory for the address told us that the corresponding page table is "Present", which means the last 3 digits are "011", we don't need create a new page table, all what we have to do is find out the corresponding page table, get the frame and update the page directory entry.

```
void PageTable::handle_fault(REGS * _r)
{
  //assert(false);
        unsigned long *cur_page_dir = (unsigned long*) read_cr3();
        unsigned long *page_table;
        unsigned long addr = read_cr2();
        unsigned long dir_offset = addr>>22;
        unsigned long err_info = _r->err_code;
        if((err_info & 1) == 1){
                Console::puts("Protection Fault.\n");
        }
        else{
                // If the current page directory's offset corresponding page table is "non present"
                if((cur_page_dir[dir_offset] & 1) != 1){
                        Console::puts("not present\n");
                        //get a frame from free frame pool and initialization process
                        cur_page_dir[dir_offset] = (unsigned long)((kernel_mem_pool->get_frames(1)<<12) | 3);
                        page_table = (unsigned long*)(cur_page_dir[dir_offset] & 0xFFFFF000);
                        //empty all the entries in the new page table.
                        for(int i = 0; i< 1024; i++){
                                page_table[i] = 0|2;
                        }
                }
                // Add the page table into the page directory entry.
                unsigned long table_offset = ((addr>>12) & 0x3FF);
                page_table = (unsigned long*)(cur_page_dir[dir_offset] & 0xFFFFF000);
                page_table[table_offset] = (process_mem_pool->get_frames(1) << 12) | 3;
                Console::puts("handled page fault\n");
        }

}
```

7 Result

The result performance of the code:



```
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...
TEST PASSED
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
```