Machine Problem 3: Page Table Management

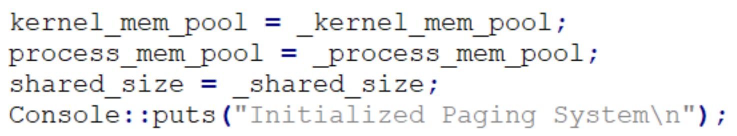
This machine problem is required to implement paging mechanism on x86, which will require us to initialize the paging system and the page table infrastructures. In a simple page management in kernel, like the previous machine problem, memory with the first 4 MB will be directly mapped ti physical memory, other rest memory will be freely mapped.

This page table is constructed by two level hierarchy paging, which one would get the page directory, and another is the page table itself. For dealing with page fault, we need to check whether it triggers the page fault or not. We need to check the CR2 register to find the page fault in page directory or page table. If the page fault in the page directory, we can directly get this page table, otherwise, we need to allocate memory first and set up the page table. Finally, set up the page number of the page table to use.

I copied the cont_frame_pool.H and cont_frame_pool.C from the previous machine problem. Both of the codes work correctly.

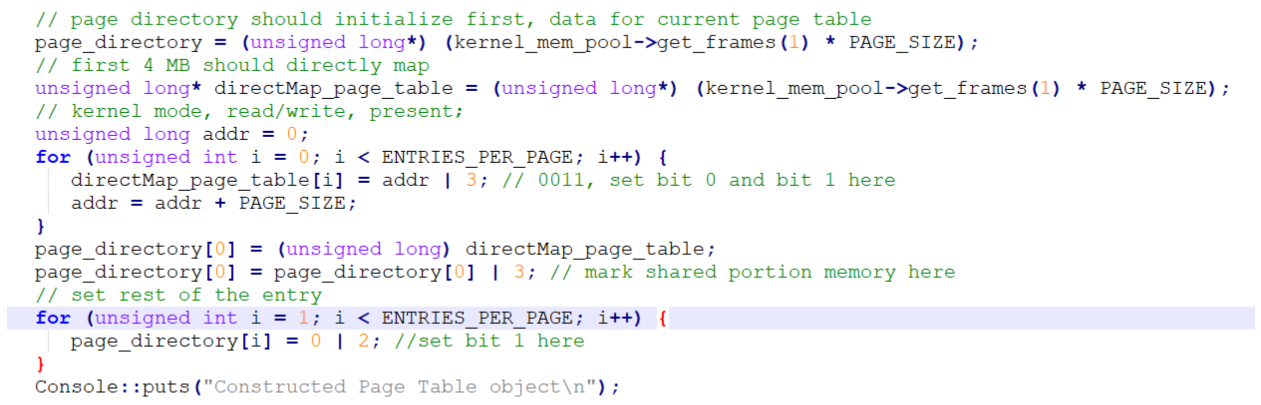
The functions I modified are all in page_table.C:

Init_page: I just followed the machine problem instruction from hand out and notes from page_table.H to initialize the private parameters.



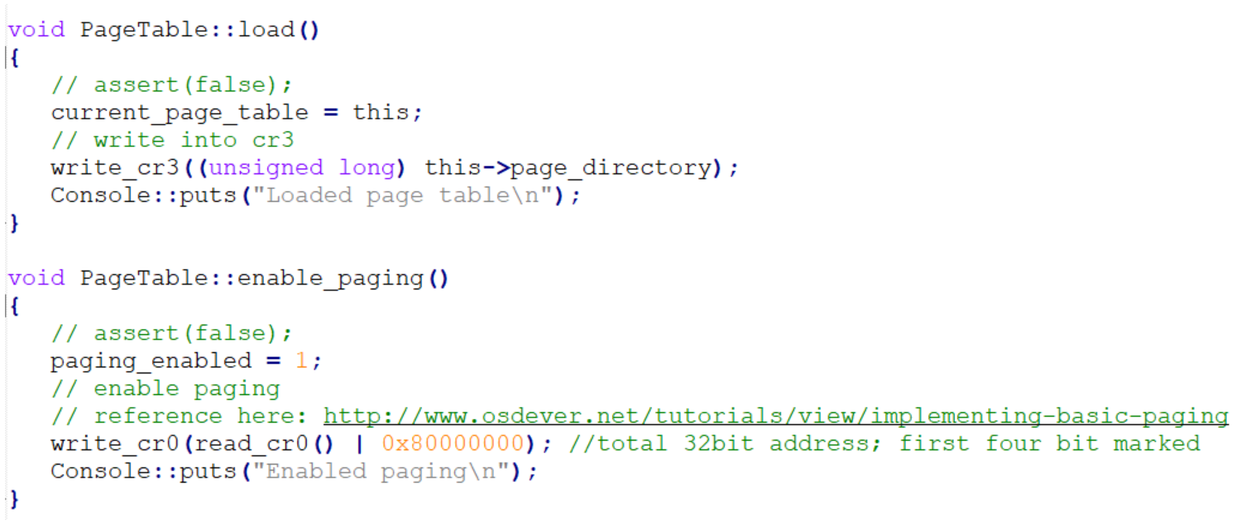
```
kernel_mem_pool = _kernel_mem_pool;
process_mem_pool = _process_mem_pool;
shared_size = _shared_size;
Console::puts("Initialized Paging System\n");
```

Page_Table(): I just initialize the page table with given location for the directory and page table correspondently. And then here need to determine the shared memory properties.



```
// page directory should initialize first, data for current page table
page_directory = (unsigned long*) (kernel_mem_pool->get_frames(1) * PAGE_SIZE);
// first 4 MB should directly map
unsigned long* directMap_page_table = (unsigned long*) (kernel_mem_pool->get_frames(1) * PAGE_SIZE);
// kernel mode, read/write, present;
unsigned long addr = 0;
for (unsigned int i = 0; i < ENTRIES_PER_PAGE; i++) {
    directMap_page_table[i] = addr | 3; // 0011, set bit 0 and bit 1 here
    addr = addr + PAGE_SIZE;
}
page_directory[0] = (unsigned long) directMap_page_table;
page_directory[0] = page_directory[0] | 3; // mark shared portion memory here
// set rest of the entry
for (unsigned int i = 1; i < ENTRIES_PER_PAGE; i++) {
    page_directory[i] = 0 | 2; //set bit 1 here
}
Console::puts("Constructed Page Table object\n");
```

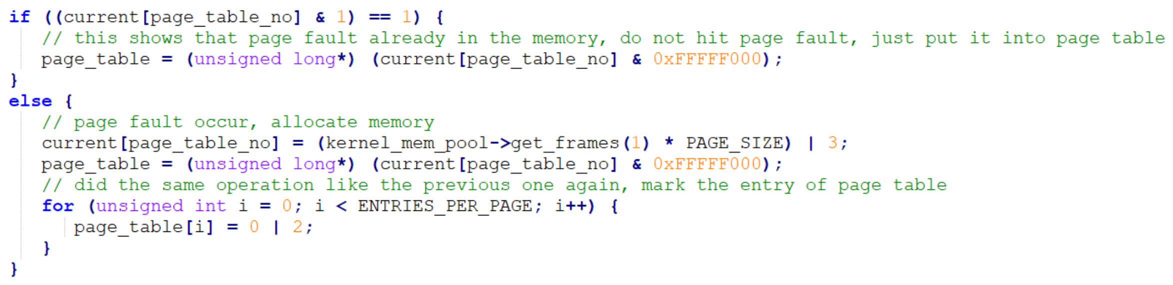
load() and enable_paging(): these two are kind of easier parts, for load() just write to CR3 via current page table directory, and for enable paging, just mark the paging_enabled to 1 and follow the instruction to mark CR0.



```
void PageTable::load()
{
    // assert(false);
    current_page_table = this;
    // write into cr3
    write_cr3((unsigned long) this->page_directory);
    Console::puts("Loaded page table\n");
}

void PageTable::enable_paging()
{
    // assert(false);
    paging_enabled = 1;
    // enable paging
    // reference here: http://www.osdever.net/tutorials/view/implementing-basic-paging
    write_cr0(read_cr0() | 0x80000000); //total 32bit address; first four bit marked
    Console::puts("Enabled paging\n");
}
```

handle_fault(REGS * _r): For dealing with page fault, we need to check whether it triggers the page fault or not. We need to check the CR2 register to find the page fault in page directory or page table. If the page fault in the page directory, we can directly get this page table, otherwise, we need to allocate memory first and set up the page table. Finally, set up the page number of the page table to use.



```
if ((current[page_table_no] & 1) == 1) {
    // this shows that page fault already in the memory, do not hit page fault, just put it into page table
    page_table = (unsigned long*) (current[page_table_no] & 0xFFFFF000);
}
else {
    // page fault occur, allocate memory
    current[page_table_no] = (kernel_mem_pool->get_frames(1) * PAGE_SIZE) | 3;
    page_table = (unsigned long*) (current[page_table_no] & 0xFFFFF000);
    // did the same operation like the previous one again, mark the entry of page table
    for (unsigned int i = 0; i < ENTRIES_PER_PAGE; i++) {
        page_table[i] = 0 | 2;
    }
}
```