MP3: Page Table Management

Po-Chieh Wang
UIN: 933004975
CSCE 611: Operating System

Assigned Tasks
Main: Completed

System Design
The goal of this machine problem is to build a page table management system based on machine problem 2. I achieved this goal by following the references in the document. First, the page table class will be constructed, the kernel will find a place in the kernel pool for the page directory. It will also find a space in the kernel pool for the first page table, which is the direct-mapped space. Then the page table will be enabled by register cr0, and will start paging by using register cr3. When page fault happens, exception 14 will raise and the handle_fault function will find a place to put the swapped-in page and return.

Code Description
To finish this machine problems, I have changed page_table.c.

PageTable::PageTable: In this constructor, it will first find a frame in the kernel frame pool to store the page directory, and a frame to store the first page table, which is the direct-mapped space. It will set every page table entry's last three bits to 011, which means they are in supervisor level, read/ write, and presented. Also, it will set the 1st page directory entry in the page directory to the address of the page table, the last three bits are also 011. The rest of the page directory entries' three bits will be set as 010, which means they are not presented. Lastly, it will set paging_enabled to 0.

```
PageTable::PageTable()
{
   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 address = 0;
   unsigned int i;
   for (i = 0; i < 1024; i++)
   {
      page_table[i] = address | 3;
      address += PAGE_SIZE;
   }
   page_directory[0] = (unsigned long) page_table;
   page_directory[0] = page_directory[0] | 3;

   for(i = 1; i < 1024; i++)
   {
      page_directory[i] = 0 | 2;
   }

   paging_enabled = 0;
```

PageTable::init_paging: This function will set the static pointers to the frame pools we constructed and set the shared size.

```
void PageTable::init_paging(ContFramePool * _kernel_mem_pool,
                            ContFramePool * _process_mem_pool,
                            const unsigned long _shared_size)
{
   kernel_mem_pool = _kernel_mem_pool;
   process_mem_pool = _process_mem_pool;
   shared_size = _shared_size;
```

PageTable::load(): This function sets the static pointer to itself, and put the page directory address into register cr3.

```
void PageTable::load()
{
   current_page_table = this;
   write_cr3((unsigned long) current_page_table->page_directory);
```

PageTable::enable_paging: This function sets the paging bit in cr0 to 1 and sets the parameter paging_enabled to 1.

```
void PageTable::enable_paging()
{
   write_cr0(read_cr0() | 0x80000000);
   paging_enabled = 1;
```

PageTable::handle_fault: This function handles page fault. First, it gets the page fault address by looking at the register cr2. Then it does some bit manipulation to get the frame number in the page directory and the frame number in the page table. Then it checks if the page fault happens

in the page directory or the page table by looking at the last bit in the page directory entry. If the page directory entry is not presented, the page fault happens in the page directory. It will find a frame in the kernel pool for the page table and update the ner page directory entry. On the other hand, if a page fault happens in the page table, it will find a frame in the process pool and update the page table entry.

```
void PageTable::handle_fault(REGS * _r)
{
   unsigned long faulty_address = read_cr2();
   unsigned long faulty_address_dir = faulty_address >> 22;
   unsigned long faulty_address_pte = (faulty_address >> 12) & 1023;

   if ((current_page_table->page_directory[faulty_address_dir] & 1) == 0)
   {
      unsigned long * new_page_table = (unsigned long *) (kernel_mem_pool->get_frames(1)*PAGE_SIZE);
      current_page_table->page_directory[faulty_address_dir] = (unsigned long) new_page_table;
      current_page_table->page_directory[faulty_address_dir] |= 3;
      for (int i = 0; i < 1024; i++)
      {
         new_page_table[i] = 2;
      }
   } else
   {
      unsigned long * page_table = (unsigned long *) (current_page_table->page_directory[faulty_address_dir] >> 12 << 12);
      page_table[faulty_address_pte] = (process_mem_pool->get_frames(1)*PAGE_SIZE);
      page_table[faulty_address_pte] |= 3;
   }
```

Testing

I think the tests provided in Kernel.C are enough for this machine problem, nothing needs to be added. The test results show that the page table management system I implemented is correct.