MP3: Page Table Management

Cheng-Yun Cheng
UIN: 633002216
CSCE611: Operating System

Assigned Tasks

Main: Completed.

System Design

The goal of the machine problem is to implement a page system for a single address space. The memory within the first 4MB is direct-mapped, and the memory beyond 4 MB is freely mapped.

Code Description

I changed page_table.h and page_table.c to complete the PageTable class for this machine problem.

page_table.c: init_paging : Set the global parameters, kernel_men_pool, process_men_pool, and shared_size, for the paging system.

```
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;
}
```

page_table.c: PageTable constructor : Initialize a page directory and a page table for the first 4 MB. First, get a frame from the kernel pool to store the page directory. It is important that all the page directory entries should be set as invalid. Secondly, get another frame from the kernel pool to store a page table. This page table is used for the first 4 MB memory. Because these memories are direct-mapped, put physical address into page table entries and set them as valid. In the end, update the first PDE and set it as valid.

```
PageTable::PageTable()
{
   // get a frame from kernel pool to store page directory
   unsigned long directory_frame = kernel_mem_pool->get_frames(1);
   page_directory = (unsigned long*) (directory_frame * PAGE_SIZE);

   // set all PDE as invalid, read/write, and supervisor level
   for(int i = 0; i < ENTRIES_PER_PAGE; i++){
      page_directory[i] = 0x2;
   }

   // get a frame from kernel pool to store page table page for the first 4MB
   unsigned long pt_frame = kernel_mem_pool->get_frames(1);
   unsigned long* page_table = (unsigned long*) (pt_frame * PAGE_SIZE);

   // the first 4MB is direct-mapped
   // set PTE as valid, read/write, and supervisor level
   unsigned long address = 0x0;
   for(int i = 0; i < ENTRIES_PER_PAGE; i++){
      page_table[i] = address | 0x3;
      address += PAGE_SIZE;
   }

   // update the first PDE and set it as valid
   page_directory[0] = ((unsigned long) page_table) | 0x3;
}
```

page_table.c: load : Load the page table into the processor context. There are two steps. The first is storing the address of page directory into the CR3 register, and the second is set the page directory as the current table.

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

page_table.c: enable_paging : Enable the paging to switch the kernel from physical addressing to logical addressing. First, set a particular bit in the CR0 register as 1. Secondly, set paging_enabled as 1.

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

page_table.c: handle_fault : The page fault handler. Read the address that caused the page fault from the CR2 register, and then call the page_fault function.

```
void PageTable::handle_fault(REGS * _r)
{
  unsigned long logic_address = read_cr2();
  current_page_table->page_fault(logic_address);
}
```

page_table.c: page_fault : Deal with page fault. First, check whether the PDE is valid or not. If the PDE is invlid, get a frame from the kernel pool to create a new page table. Update the PDE and set all PTE as invalid. Secondly, for invalid PTE, get a frame from the process pool. Store the frame address into PTE and set it as valid.

```
void PageTable::page_fault(unsigned long logic_address)
{
    int pt_no = logic_address >> 22;
    // if PDE is invalid, create a new page table
    if((page_directory[pt_no] & 0x1) == 0){
        // get a frame from kernel pool and store it in page directory
        unsigned long pt_frame = kernel_mem_pool->get_frames(1);
        page_directory[pt_no] = (pt_frame * PAGE_SIZE) | 0x3;

        // initial all PTE as invalid
        unsigned long* new_page_table = (unsigned long*) (pt_frame * PAGE_SIZE);
        for(int i = 0; i < ENTRIES_PER_PAGE; i++){
            new_page_table[i] = 0x0 | 0x2;
        }
    }

    // get a frame from process pool
    // store it in page table and set it as valid
    int p_no = (logic_address >> 12) & 0x3ff;
    unsigned long* page_table = (unsigned long*) (page_directory[pt_no] & 0xfffff000);
    if((page_table[p_no] & 0x1) == 0){
        unsigned long p_frame = process_mem_pool->get_frames(1);
        page_table[p_no] = (p_frame * PAGE_SIZE) | 0x3;
    }
}
```

Testing

For the machine problem, I only use the provided test, because it covers all functions which I implemented.