MP3: Page Manager I

Kai-Chih Huang
UIN: 333000021
CSCE611: Operating System

Assigned Tasks

Main: Completed.

System Design

The goal of this machine problem is to design a page manager that can run single process. I implemented functions in PageTable class to setup parameters of paging, to enable paging system, and to handle page faults.

Before we instantiate a PageTable instance, we first need to call PageTable::init_paging() to initialize some parameters. After initializing, we can instantiate PageTable, which will by default create a page directory table and a page table page. Then, we need to call enable_paging to officially start the paging system. When there is page fault, the exception 14 will be raised, and our handle_fault() function will be called to deal with it. The details will be explained in the following section.

Code Description

I changed page_table.C for this machine problem. To compile code, simply run following command lines under MP2_Sources directory:

```
$ make clean    // clean the old compile files before we compile

$ make          // compile files

$ ./copykernel.sh (if permission denied, try chmod u+x ./copykernel.sh then do it again)    //copy kernel

$ bochs -f bochsrc.bxrc     // run bochs
```

I will walk through the functions/methods defined in page_table.C. as follows.

init_paging (ContFramePool * _kernel_mem_pool, ContFramePool * _process_mem_pool, const unsigned long _shared_size)
This function set variables of the size of shared address space, kernel and process memory pool. These variables are static, which is common across different page tables for different processes.

PageTable()
This function construct a PageTable object. We require 2 frames from kernel memory pool by calling get_frames method of ContFramePool object, 1 for page directory and 1 for page table page. We only create 1 page table page at this time because the entries in this page table page is directly mapped to the first 4MB of physical memory, which the address is already known. As for other pages table pages, we will create it when it is needed via handle_fault function. Then, we initialize the page table entries and set as supervisor, read/write, present mode. After that, we set the first page directory entry to point at this table, and set the

remaining entries to supervisor, read/write, not present mode, since they are not pointing to any page table page yet.

void load()

This function set the PageTable object that calls this function as the current table. I use write_cr3() function to store the address of the page_directory to CR3 register, so that when CPU knows where to start to walk the page table.

static void enable_paging()

This function turns on the paging system by setting the paging bit in CR0 to 1, so that CPU knows the paging system is enabled. Then, we also set the paging_enabled flag to 1.

static void handle_fault(REGS * _r)

This function will be called by hardware when page fault happens. First it will find out the fault address and extract the target page directory entry and corresponding page table entry. If the page directory entry is not present, then require a new frame as page table page, initialize its entries as "not present" state, then point the faulty PDE to this new table and set its present bit to 1 since now this entry points to a valid page table page.

Testing

I did not add additional test data in this MP3, since I believe the provided test set is sufficient to check if the page table system works as expected. The test result is shown as follows.