Hung-Hsueh Shih (UIN: 132006330) CSCE611 Operating System
Machine Problem 3 : Page Table Management

Introduction
The objective of this machine problem is to set up and initialize the paging system and the page table infrastructure for a single address space.

Design:
In order to use two-level paging system in this mp, we need to create an object PageTable which contains page directory by using the first 10 bits of logical address and page table page by using the middle 10 bits of logical address and the offset which is the last 12 bits in logical address. Since we do not have memory manager yet, we need to store page directory and page table page in frames. When we turn on paging enable logical address and we need to have page fault handler to address the issue of page fault.

Implementation:

1) page_table.C

init_paging: set up the kernel frame pool and process frame pool and the share memory region to initialize the page table object.

Constructor: in the constructor I use one frame to define page directory and one frame to define page table page. In order use frame, I call the get_frame() function from kernel frame pool which is an object of cont_frame_pool. Since the first 4MB are direct-mapped I use one page table page to reference this memory region and initialize each entry in the page table page to present and write. After initializing the page table page fore the shared memory region I use first entry of page directory to refer this page table page. In the end, I initialize the rest of entry in page directory to invalid.

load(): load the page directory address into page table register by using write_cr3() to store page directory into cr3 register.

enable_pagining() :Based on the reference resource : http://www.osdever.net/tutorials/view/implementing-basic-paging to starting paging I need to set the paging bit(bit 31)of CR0 to 1, and paging will be turn on.

handle_fault(): this is page fault handler which will invoke hardware to interrupt and exception routine. Page fault will happen in two situation. First, cpu reference a page and it's page directory is 0, thus, we need to get a frame

from kernel pool to create a page table page and point the entry of page directory to new page table page. The second situation is that page directory is valid but the page is not in the page table page. To solve this problem, we need to get a frame from process frame pool and reference the frame toe the page table page entry. When an error occurs, I use the exception number to know whether it is a page fault. Then, I read the address of current head of page directory as a pointer from cr3 register and read the address which cause page fault from cr2 register. After, I get the logical address which cause page fault, I use the first 10 bits as index to find out the value of PDE. If the present bit of PDE is 0 which means the page directory is invalid, then I create a page fault page for this entry and reset the entry to valid. On the other hand, I check use the middle of 10 bits logical address to check whether the page fault occurs because the page table page is invalid. If it is invalid, get a frame for the PTE and reset the value of PTE to valid.

2) cont_frame_pool.H

Header file for contiguous frame pool, I define the private variables in this header file included the basic information for the frame pool (such as bit map, total number of free frames in this pool, total size of the frame pool, and the pointer point to next frame.) Beside, I also define two method for cont_frame_pool class, one getter and one setter.

3) cont_frame _pool.C

Constructor : declare all the information for this frame pool, included the number of first frame in this frame pool, the size, in frames, in this frame pool, the total number of free frames, and the information frame number.
In addition, I initialize the bit map to 0x0 to initialize each frame in this frame pool is free. Then, based on the frame pool to set up the storing location for this frame pool (in kernel frame pool I store the bit map in the first frame, thus, I need to change the state of first frame to 01 which is head. Last, I link the linked list like structure for the frame pool.

get_state() : use the frame number as an input return the state of the frame from 00, 01, 10 to11. Since one byte can store state for four frames, I need to based on the frame number to find out the index of frame in bit map and the offset in this one byte. Eventually, I do the bit manipulation and return the state of frame.

set_state() the setter for the frame pool, take frame number as input, then change the state of frame number based on the input state we want to set up. Once again, do the bit manipulation to change the state of frame.

get_frames() input is how many frames we want to allocated. Iterating the Bit map first and using the getting to find out the state of this frame until we find out contiguous frames which have enough space to be allocated. Set the first frame in this sequence to 01 by using the setter and set the rest of frames in sequence to 11 as allocated.

mark_ inaccessiable() given the frame number and how many contiguous of frames, use setter to change the state of given input to inaccessible which is 10.

Release_frames() first checker whether the frame pool possess this frame number. Then use getter to check this frame number is the head of sequence. In the end, use setter to reset all frames to free.

needed_info_frames()
Return the number of info frames required to store the frames. In this case, we use one frame to 16KB.