

## MP4 Design

Tsao Yuan Chang 927000791

In MP4, we need to do some modification on `page_table.C`, `page_table.H` to make use of virtual memory. We also need to implement VM pool in `vm_pool.C` and `vm_pool.H`. I didn't do any modification on `cont_frame_pool.C` and `cont_frame_pool.H` since they're working great. This design document won't cover `cont_frame_pool.C` and `cont_frame_pool.H`, but I'll cover my works on other parts.

### `page_table.C`

#### 1. constructor

Unlike MP3, I save page directory and page table outside the first 4MB, so I need to get the frame from `process_mem_pool`. For PTEs, I make them writable and valid while PDEs are writable but not valid(not present). And for the last PDE, it points to the start of the page directory, so I can do the recursive page table lookup task. The registered VM pools are stored in the array `registered_vmpools` whose maximum size is 100. `registered_vmpools` is initialized to NULL in the constructor.

#### 2. `load()` and `enable_paging()`

There's no modification in these 2 functions comparing to MP3.

#### 3. `handle_fault()`

First, I read the address causing page fault from CR2. And then I compute the PD and PT virtual address of this address in the format (1023 1023 X) and (1023 X Y) respectively.

## Addressing a PDE using Recursive Table Lookup



## Addressing a PTE using Recursive Table Lookup



If the page fault is due to accessing invalid page, which is what we're trying to deal with, I'll handle this page fault. First I'll find which vm pool this page fault address is in. If I successfully find the vm pool, I'll check where did the page fault happen: PD or PT. If it happened in PT, I'll get a frame from `process_mem_pool` use this frame as a PTE. Otherwise, I need to get frames for both PDE and PTE.

#### 4. `register_pool()`

If the number of registered pool hasn't reached 100, I'll register the input `_pool` and update the `registered_num`.

#### 5. `free_page()`

In this function, we need to compute which frame contains `_page_no`, and then use `release_frame()` to free this page.

## page\_table.H

I added some private variables such as `registered_vmpools` and `registered_num`. And I also added some functions for this MP like `handle_fault`, `register_pool` and `free_page`. Other parts are the same as MP3.

## vm\_pool.C

### 1. constructor

In the constructor, first I save the inputs into private variables. And then I use the struct regions to save the region descriptor, including the information such as the region's base address and its size. Finally, I use the function `register_pool` in `page_table.C` to register this vm pool.

### 2. allocate()

In this function, I get the new region's base address by the previous one's descriptor. And then add the current one into the regions array.

### 3. release()

First I need to know which region does the `start_address` belong to. After that, I free the pages one by one in that region using the `free_page()` function. Finally I update the regions array and flush the TLB by reloading the page table.

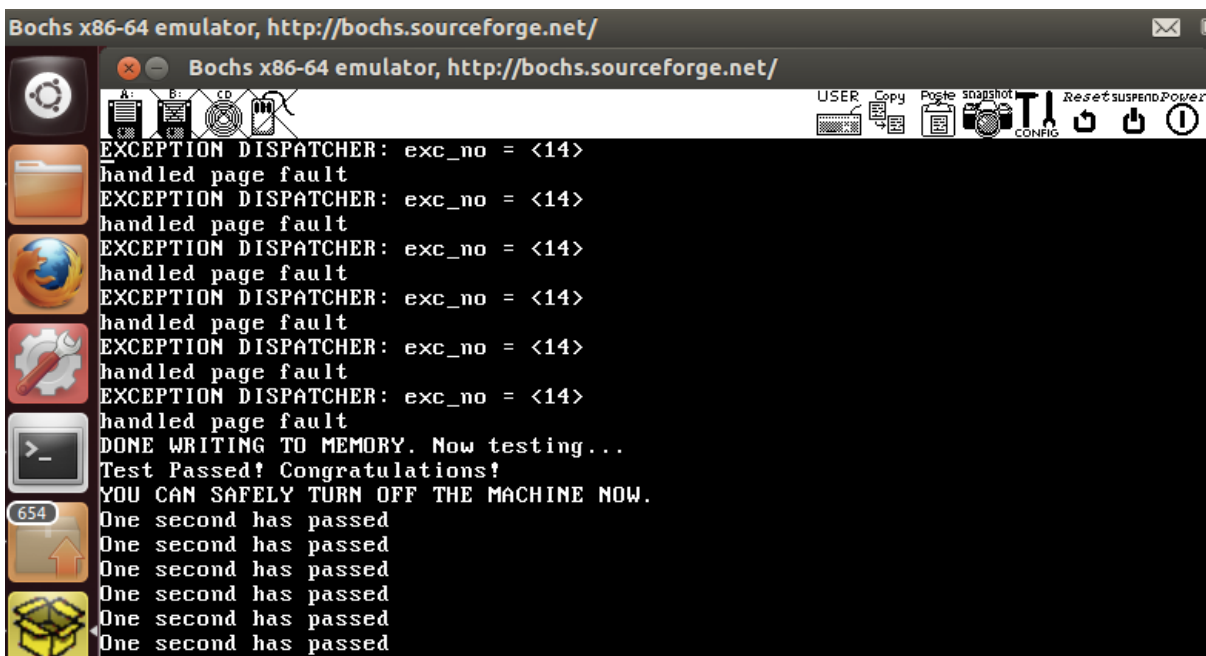
### 4. is\_legitimate()

I check whether the input `_address` is legitimate by going through the regions array. If for some region, the input `_address` is greater or equal than the region's `base_address` and smaller or equal than `base_address + size`, then the address is inside this region and we can return 1. Else if we cannot find any region containing this address, we return 0.

## vm\_pool.H

In this file, I declared a struct `region_descriptor` to record each region's base address and size. I also added some private variables such as `region_num` and `regions` (of `region_descriptor` type) for convenience.

Final testing result :

A screenshot of a Bochs x86-64 emulator window. The title bar reads "Bochs x86-64 emulator, http://bochs.sourceforge.net/". The window contains a terminal-like interface with a black background and white text. The text shows a series of "EXCEPTION DISPATCHER: exc\_no = <14>" messages, each followed by "handled page fault". After several of these, it says "DONE WRITING TO MEMORY. Now testing...", "Test Passed! Congratulations!", and "YOU CAN SAFELY TURN OFF THE MACHINE NOW.". This is followed by a series of "One second has passed" messages. On the left side of the window, there is a vertical toolbar with various icons. On the right side, there is a horizontal toolbar with icons for "USER", "Copy", "Paste", "Snapshot", "CONFIG", "Reset", "Suspend", and "Power".

```
Bochs x86-64 emulator, http://bochs.sourceforge.net/
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Now testing...
Test Passed! Congratulations!
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
```