Implementation Details

1. Frame Pool

1.1 Frame pool management data structure:

I use bit map way to manage the frame pool usage. I declare an array of ***unsigned char*** type pointer, since every char is one byte. And one byte is 8 bit. So we will need k=(*nframes/8*) byte to manage the usage information of our frame_pool.

Such pointer is called ***oneByte * free_frames***

1.2 Where to store the management data structure

The ***free_frames*** to manage both kernel_frame_pool and process_frame_pool is store in the kernel memory. They are placed in the first frame and the second frame of the kernel memory. Since each frame is 4KB, so that it is big enough to store the usage of (2 MB/4KB) frames of the kernel memory and (28 MB/4KB) frames of the process memory.

To find the memory address for the pointer is easy. We know the frame_no of the first kernel frame is 512. Then the address will be 512*4096. And the address for store the management data structure of the process_kernel will be 513*4096.

After this, we initialize all the value to 0 and mark the first and second frame in the kernel_frame_pool as used.

1.3 How to find a free frame

Just loop from 0 to nframe/8 to find the first value that is not equal to 255 (11111111), that means there is an free frame. We then get the first bit number equal to 0 in this value. For example if we find that current first value not equal to 255 is the 3rd one and the first bit equal to 0 in this value is 6. Then we can get the free frame number by 3*8+6=30, by add it to its corresponding base_frame_no we can get the position of this frame in memory.

1.4 Mark used and release frame

Use bit manipulation to set corresponding bit in our free_frames data structure to 1 or 0.

2. Page Table

2.1 Where to put page table

Page table is put in kernel memory. To find the right address to put page table just use kernel_mem_pool->get_frame() to times the page_size (4096). So one page table use one frame in the kernel memory.

Before make a page table, we first need to make an page_directory to manage our page tables. I also put page_directory in kernel memory by the same way.

Then I just map the first 4MB memory directly and put this page table in the first entry of the page_directory.

3 Page Fault Handler

After get the address that cause the page fault by ***address=read_cr2()*** . We can use this address to find where is the page_table in the page_directory and map which entry in page_table to the physical memory.

3.1 Find entry in page_directory

First right shift the address 22 bits, to get the highest 10 bits, that tell us which entry in the page_directory we should have our page_table.

Then if there is no page_table in that entry yet, we have to create one first. It is same as above how we create the page_table for mapping the first 4MB, except that now it is not directly mapping. So that we just use process_mem_pool->get_frame()*4096 to find a physical address and map this address to a entry in our page_table. In this case, we need to map the entry 1024 frames to our 1024 entries in the page_table. Not only a single address.

If there is already existing an page_table, then we can simply map the corresponding entry to its physical address.

3.2 Find entry in page_table

Use a mask to get the second highest 10 bits in the address, and right shift 12 bits, that tell us which entry in the page_table we should map it to the physical address in memory when a page fault happen.