Overall Design

In MP2 we implemented a frame pool to management frame creation and deletion. Our system was a 32 MB machine with the kernel space from 0 to 4 MB and the user space from 4MB to 32MB. There also is a 1MB hole of inaccessible memory starting at 15 MB. The kernel space will be direct mapped from virtual memory to physical and shared between process page tables. User space memory will be managed. Both the frames and the pages will be 4 KB in size.

Initial Frame Pool Design

In my design, I used the bitmap to record management information to manage the usage of frames. The page size is 4 KB. Here is some simple calculation, if one bit is used to monitor one frame, then the total number of frames can be monitored by this 4KB page is 4x1024x8 = 32 KB frames with each frame 4 KB size and 128MB total. Now if we used 2 bits to manage one frames, then the total number of frames can be managed is 4x1024x8/2 = 16 KB with each frame 4 KB and 64 MB total. Here our system has only 32 MB, so one page is enough to manage all the 32 MB frames. In general, if 2 bits used to mark one frame, if n frame is requested to allocate, then the number of management frame can be calculated as: **n/16KB + n%16KB.**

In this assignment, we need to assign continuous frames and free continuous frames, so we need to know the start of the continuous frames to be assigned as well as the frames to be used. More specifically, in this 2-bit information case, I define the management bits as *11* if this frame is available, 00 is used and as start of continuous available frames and 01 for the used but not the start of this continuous page demanding. Also, I used a static pointer to the Frame Pool class after each frame pool creation, it helps me to locate the exact frame pool during deletion of frames.

How to Get Frames

In this notation, I initialize the bitmap as *0xFF* for all the available frames and if the information frames occupy the first frame, so I marked the particular byte as *0x3F* (0011 1111), the first two bits is marked as 00. Now we go to the *get_frames(nframes)* function, below is the simple illustration of bitmap assignment, bitmap[i] controls First, Second, Third, Forth, in total four frames, namely, every two bits control one frame.

Table 1 Bitmap Allocation Illustration

i |bits	First		Second		Third		Fourth	
1	0	0	0	0	0	0	0	1
2	0	1	0	1	0	1	0	1
3	0	0	0	0	0	0	0	0
…..	0	0	0	1	0	1	0	1
n	0	1	1	1	1	1	1	1

In order to get the free frames, we need to first locate which frames is available, so we can transverse all the information frame from 1, 2, to n. Here, I used two bits to mark each frame, so

the two-bit number might either be 00 or 01 (00 for the start sign, and 01 for the used sign). I just need to check the first bit of every two-bit combination and if the first one is 0 then occupied. So I mask the bitmap[i] with *0xAA*(1010 1010) and then the bitmap[n] with nonzero is identified as the one with free frames. So, I mask the bitmap[n] & *0x80 (1000 0000)* with shift 2 to know the exact location of free frames, in the bitmap[n][j].

Now I have to check if there are continuous n frames available in the memory. I used the subroutine

```
boolean checkContinuous(unsigned char* bitmap, int i, unsigned char mask, unsigned int
_n_frames)
```

to check if there is continuous space for this assignment. Basically, this subroutine enumerates for 0 to n frames, and keep checking the bitmap[i] afterwards to know if the current frame is assigned or not. So, if the frame is already being assigned, it means it cannot find continuous frames and there might be a hole.

Now if the above check is successful, so we can flip the first two bits available to *0x00* with a mask *0xC0* and the rest of available bits to *0x01* with mask *0x80* to mark the n frames are all used.

How to Mark Inaccessible

If we want to mark a frame as inaccessible, so we first locate bitmap[i] as the index and get bitmap[i][j] as the exact place. More specifically,

i = n_frame / 4, j = n_frame % 4

We want to make the corresponding bits as *0x00* as inaccessible, *so we first shift 0xC0(1100 0000)* with 2*j places, then use the xor operation as bitmap[i] ^ (0xC0>>2j).

How to Free Frames

To delete use frames, we can reverse the process to get frames. But here to need to which memory pool we are currently dealing with because there might be many memory pool assigned at the same time. Here, I used a static variable `ContFramePool::pool_list` as the linked list to record the creation of memory pool. Whenever the Class ‘`cont_frame_pool` is created, I append the newly created class to the head of the linked list.

Now, if we want to delete the memory pool area starting with first_pool, we need to know which memory pool created this first_pool. So, we transverse the linked list from head, and each memory pool, it has to be between head->base_frame and head->base_frame + head->nframes. We keep checking this condition and if it is not satisfied, we move to the next pointer.

Next, after identifying the memory pool as current memory pool, we can now reverse the creation process through identifying the index i = n_frame /4, j = n_frame %4. This place should have the bits 0x00 because it is the start of the continuous sequence, and then we flip these two bits to 0x11 through xor it with 0x11.

For the rest of the non-start-of-sequence frames, I simply iterate the bitmap bits to make sure the current bits are *0x01.* Then I flip these bits to 11 as marks for free. Otherwise, if I encountered bits *0x00* or *0x11*, it means I reach the end of this continuous memory assignment block, it is time to stop.