Machine Problem 2: Frame Manager

This machine problem is to implement frame manager to allocate different frames, and it will track which frame is allocated and which frame should be released. Like the handout mentioned, the total memory of this machine is 32MB, the first 4MB are used for the kernel and shared by all processes. The first 1MB is for all global data. Thus, the 4MB to 32MB is for process, and 1MB hole at 15MB is marked as inaccessible.

In the cont_frame_pool.H

I added some private parameters and functions to help initialize and help to release frames.

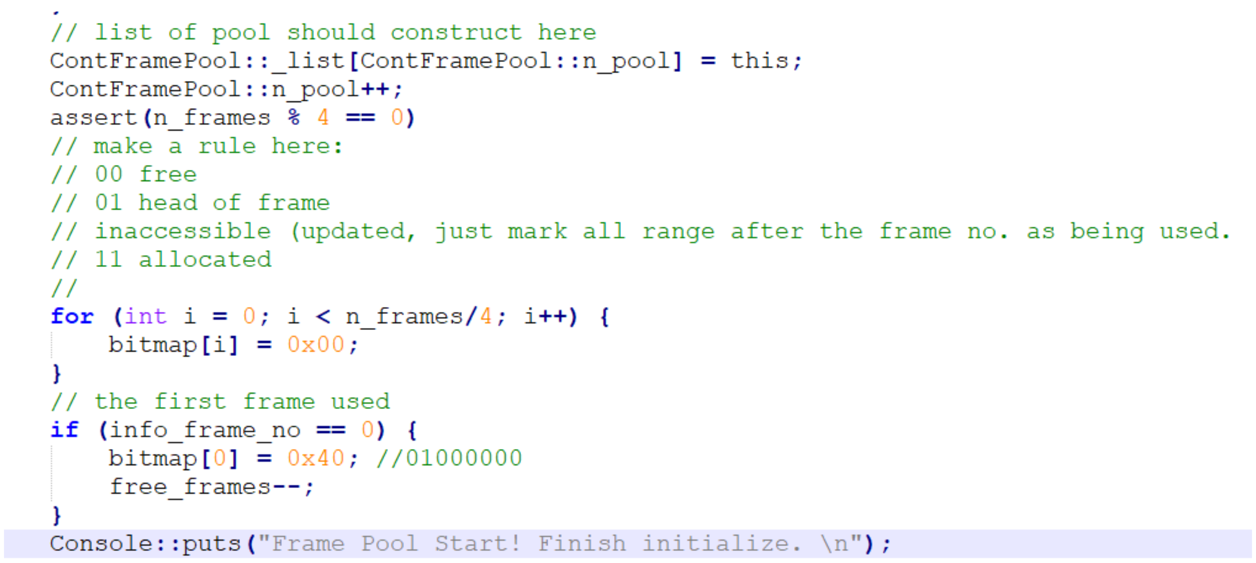
```
private:
    /* -- DEFINE YOUR CONT FRAME POOL DATA STRUCTURE(s) HERE. */
    // based on the hand out instruction, implement bit map approach
    unsigned char* bitmap;
    //initialize the same things in the public
    unsigned long base_frame_no; // frame pool start
    unsigned long n_frames; // frame pool size
    unsigned info_frame_no; // manage information store
    unsigned long n_info_frames; //size of storing in the manage information
    unsigned long free_frames; //free frames remain

    // helper function to release frames
    void helper(unsigned long first_frame_no);

    // pointer of pointer to implement to release frames
    static ContFramePool** _list;
    static unsigned int n_pool;
```

These can help construct the ContFramePool class, and the pointer of pointer _list and n_pool can help find the frame no that should be released.

In the cont_frame_pool.C

There are several implementations I modified.

In the constructor part,



```
// list of pool should construct here
ContFramePool::_list[ContFramePool::n_pool] = this;
ContFramePool::n_pool++;
assert(n_frames % 4 == 0)
// make a rule here:
// 00 free
// 01 head of frame
// inaccessible (updated, just mark all range after the frame no. as being used.
// 11 allocated
//
for (int i = 0; i < n_frames/4; i++) {
    bitmap[i] = 0x00;
}
// the first frame used
if (info_frame_no == 0) {
    bitmap[0] = 0x40; //01000000
    free_frames--;
}
Console::puts("Frame Pool Start! Finish initialize. \n");
```

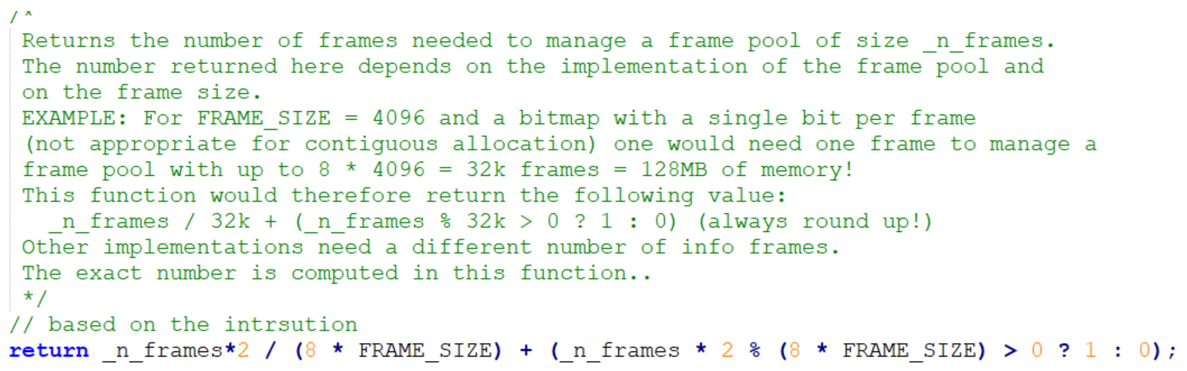
I marked two-bit as the status, 00 as free, 01 as the head, 10 as inaccessible and 11 as allocated. Then I initialize the frame pool to set the head frame.

In the get_frames function, it is one of the most difficult part in this machine problem, the function required to allocate a number of contiguous frames from the frame pool. _n_frames: Size of contiguous physical memory to allocate in number of frames. If successful, returns the frame number of the first frame. If fails, returns 0. I used the method to traverse the whole length of the frame pool. Since the bitmap will store 4 frames, after find the byte position and head position, I set the head of the frame as 01, and rest of the _n_frames as 11.

In the mark_inaccessible function, just find the byte position and head position of the head of the frame, then mark all the rest of _n_frames as being used (allocated), which may be easier than just change the head of frame from 01 to other bit number. And update the number of free frames. It is similar implementations from simple_frame_pool.C.

In the release_frames function, I wrote a helper function, which when the pointer of pointer _list find the frame no. that need to be released, I implemented XOR operation to change the value to 00 (free).

In the needed_info_frames function, I just followed the note instruction from the header file.



```
/*
 Returns the number of frames needed to manage a frame pool of size _n_frames.
 The number returned here depends on the implementation of the frame pool and
 on the frame size.
 EXAMPLE: For FRAME_SIZE = 4096 and a bitmap with a single bit per frame
 (not appropriate for contiguous allocation) one would need one frame to manage a
 frame pool with up to 8 * 4096 = 32k frames = 128MB of memory!
 This function would therefore return the following value:
   _n_frames / 32k + (_n_frames % 32k > 0 ? 1 : 0) (always round up!)
 Other implementations need a different number of info frames.
 The exact number is computed in this function..
 */
// based on the intrsution
return _n_frames*2 / (8 * FRAME_SIZE) + (_n_frames * 2 % (8 * FRAME_SIZE) > 0 ? 1 : 0);
```

In the kernel.C just uncommented the process pool and test it.