Design for MP2

This design document describes the implementation of my frame pool and gives reasoning why the implementation is correct.

1. **Helper functions**-
   I created multiple helper functions that were thoroughly tested in the (MP2/testing) folder in my github. These helper functions allowed me to do bitwise manipulations in the bitmap accurately and with robustness. These functions were used multiple times throughout the actual implementation
   a. **bool frameCompare(_frame, val)**
      Checks if frame number in bitmap is equal to the binary value.
      ex. frameCompared(3, 0b00) , will return true if frame 3 in the bitmap has a mask equal to 0b00, or false otherwise.
   b. **void setFrameBitMask(_frame, val)**
      Sets the frame's bit mask inside the bitmap to val.
      ex. setFrameBitMask(4, 0b11), sets the bit mask of frame 4 to 0b11
   c. **void markContFrameMasks(_base, _n_frame, val)**
      Sets _n_frames bitmasks to val starting from _base in the bitmap
2. **Implementation of required functions-**
   The main implementation I used was a bitmask for each frame pool to keep track of 4 different states of a frame:
      00- allocated
      01- inaccessible
      10- head
      11- free
   This means that we need to keep track of 2 bits per frame as overhead. So I adjusted calculations accordingly to see how many number of info frame are required (n_frames/(4 * frame_size)).
   I also use a singly linked list and have a static variable head.
   a. Constructor
      In the constructor there are a few size checks that are done, then the bitmap is initialized as free (with exception to info frames if they are internal).
      The "this" pointer is then inserted into the singly linked list after "head" or becomes the head if it is null.
   b. Get_frames
      This is a simple for loop that searches for a "First Fit" segment of frames that match the desired frame count. Once this is found, we set the first frame bit mask to "10" and all allocated frames after the first frame to "00". This will result in something like this (frames = 5).
      (10)(00)(00)(00)(00)(11)(11)....
      We then return the first frame number.
   c. Mark_inaccessible
      Simply iterate through the bitmap and set the bitmasks of all the frames to "01" which is inaccessible

d. **Release_frame**
   First we iterate through the singly linked list to find the proper frame pool by checking the frame pool boundaries and the frame number requested. Then using the frame pool's bitmap, we mark frames as "11" or free until we reach a frame mask that is not "00" or we reached the end of the frame pool frame count.
e. **Needed_info_frames**
   This is simply just an equation based on frame size, frames per byte, and number of frames we want to give to the frame pool. The resulting equation is:
   N_frames/(frame_size * 4) + (n_frames % (frame_size * 4) > 0 ? 1 : 0)

3. **Testing**
   The program was tested using the Kernel.C file which had the default test. Other tests were done one utility function separately. Many edge cases were considered and bit mask operations on the bitmap were tested in the "MP2/testing" folder. This provides strong evidence for correctness in the overall program.
4. **Modifications**
   The only files modified were cont_frame_pool.H and cont_frame_pool.C.
   The cont_frame_pool.H was modified in order to add helper function prototypes which were motivated earlier, and to add a static variable pointer to the head of a singly linked list which was also motivated above. The linked list allows for proper release frames function that only requires a frame number and is static.