Implementation of the Frame Pool

Vishruth Raghuraj Gollahalli

Assumptions:

I have used the following initial bits to determine various kind of frames:
11 -> Free frames
00 -> Allocated frames and also head of sequence
01 -> Allocated frames but not head of sequence

Implementation:

1. get_frames : For this function, first, I find out if there are enough free frames to allocate frames. If there aren't, break out of the function and return 0. If there are free frames available, then do the following:
   Keep searching for a frame that is not allocated (in my case, a frame whose first two bits are 11). Once such a frame is found, I store it in the startFrameNo variable, and, as long as the frames are within the bounds, I look further to see if enough continuous frames are yet marked as free.
   Once outside the loop, I can be sure that enough free frames are available and I continue to mark the first frame as allocated as head of sequence (mask with 0xC0) and the remaining as allocated (mask with 0x80). Finally I reduce the number of available free frames by the number of allocated frames within the loop and return the startFrameNo.

2. mark_inaccessible : This function was implemented much in the same way as given in the simple frame allocator example. The only major difference was that, in this function, the mark_inaccessible function was called for a range of "_n_frames" frames. This was done by running a loop within which for each frame, the mark_inaccessible function was called. Within the function, the masked the bitmap with 0xC0.

3. release_frames : This was the trickiest. The first step was to traverse through the linked list to find the frame pool within which the frames needed to be released. Once the frame pool was identified, I called the release_frame function (followed the example in the simple frame allocator code). Within the release_frame function, first I performed a sanity check to ensure that the frame was indeed a head of sequence. If it wasn't I would break out of the function. If it is, I systematically reset the head of the sequence frame to 0xFF (free) and then ran a loop to do the same to the allocated contiguous frames. Finally, I increased the available free frames count after each reset.

4. needed_info : I used 8 bits (1 character) to represent the state of the frame. Thus I calculated one info frame per each FRAME_SIZE frame.

All changes made were in the cont_frame_pool.c and cont_frame_pool.h files. The only change I made in the kernel.c file was uncommenting the process pool for completeness. This did not affect the functioning of the frame allocator.

References:

[1] A. Silberschatz, P. Galvin, and G. Gagne, *Applied Operating System Concepts,* John Wiley & Sons, Inc., New York, NY, 2000