Design Document

(describing implementation of frame pool manager)

Suma Garuda – UIN: 926009146

1. Overview

I have used a char to represent a frame in bitmap (just started to work with masks to implement a 2 bit approach for efficiency, but could not complete it). Inside a bitmap, we use the following representations:

1st bit – to determine if a given frame is head of sequence
2nd bit – to determine if given frame is free or allocated
Thus, 00 – free, 01 – allocated, 11 – head of sequence and allocated

2. Constructor: ContFramePool

We first initialize all disk blocks/frames as available (i.e. fill bitmap with 0's). Then, we check the _info_frame_no, to see where we can store management information. If it is 0, we store management information in the base frame, mark it allocated and reduce the no of free frames by 1. If it has any other value, we call the needed_info_frame_function. We then use a linked list to link the nodes of the frame pool, with the static pointer head, non-static pointer next and the this pointer.

3. Get_frames()

To start getting the requested no of frames, we find the first available frame and mark it as head of sequence. Then we can continue checking if we have the requested number of contiguous frames available by comparing bitmaps (first fit approach), if we find them we mark them as allocated and return the head of sequence frame number to test call.

4. Mark_inaccessible()

There is a certain region of memory – between 15 and 16MB that cannot be accessed, thus we use this function to mark the frames in this region as allocated.

5. Release_frames()

This function contains 2 implementations, one is really easy to understand. This function used the kernel.C function to access the start of KERNEL_FRAME_POOL and PROCESS_FRAME_POOL and their respective sizes. Here, we check if the requested frame/frame_pool lies in 2MB-4MB (i.e. kernel frame pool) or 4MB-32MB(i.e. process frame pool), then find the head_of_sequences in the frame pools requested and mark all the contiguous frames as free. Although this was my first implementation approach, I could not get the function to access the bitmap due to the function being static.

Thus, I had to change my approach completely and use a linked list to traverse all frames (find the frame pool of the requested frame segment) and free the ones requested (by marking them as 00) in the non-static release_frame function (just so that we have access to the bitmap).

6. Needed_info_frames()

   Just made the required changes to the needed_info_frames specified in simple_frame_pool.H (as we are using a char to represent the frame in the bitmap).

7. Testing and Debugging:

I just added one more line to kernel.C to test if get_frames() and release_frames() works with both process and kernel pool. Other than that, I just used print statements throughout the code to check variable values or identify where the code would get stuck.