MP2 Design Document
Alex Huddleston
CSCE 410 – 700

Introduction

For purposes of simplicity and conciseness, I will mostly be talking about how I implemented the frame pool using cont_frame_pool.C as included. I'll go in order from top to bottom of the member functions needed and address the data structures separately.

Implementation

Constructor

The constructor starts off in a similar way to simple_frame_pool, as most of the same data structures are still in tact for this class. However, things change once we get into allocating multiple info_frames per pool. I made a slight adjustment to the bitmap size depending on whether a number of info frames was specified, then made further adjustments when considering that the bitmap I used for this project was the proposed "2 bits per frame" bitmap, as I felt it was the most convenient and efficient to completing this project. The only real modifications here are in ensuring that the number of info frames needed for the pull is correctly marked as reserved when specifying more than one info frame, and ensuring that the pools data structure is initialized correctly, which I will go over more thoroughly in the data structures section of this deisgn document.

get_frames()

get_frames() by far took the longest to complete. At first, I figured it would be easy enough to just modify the code provided in simple_frame_pool, but I quickly discovered that simple_frame_pool's implementation was not at all a way that I could wrap my head around easily. After a while, my own implementation ended up being very similar, but much more drawn out than the one provided.

It was difficult for me to simply adjust what was given for two reasons:

1. I had to account for each frame now accompanying 2 bits in my bitmap rather than just one.
2. I had to account for allocating multiple frames in a contiguous sequence rather than just picking the first one available.

The final adjustments I made were to updating the bitmap, as I needed to ensure that the first frame in the sequence only cleared one bit for its information frame and all sequential frames following it cleared both of their information bits.

mark_inaccessible()

I pretty much directly copied this from the simple implementation. Only needed to account for checking for 2 bits and adjusting the bitmap_index and mask math according to how my bitmap was laid out. Again, this will be addressed in more detail in the data structures section.

release_frames()

While get_frames() took the longest, this function was the trickiest to wrap my head around. Once I understood the concept of what I was trying to accomplish, I got to work on making a data structure for the class that could hold the different pools I would need in order to maintain release_frames() as a static function, then I made a member function specific to the object that would be able to release the frames needed once that object was identified.

To identify the needed pool, I simply performed a check to see if the frame I needed to release was within the bounds of the pool I was currently checking, going through each in my list. Once the correct pool was found, I called that object's release_frames_here() function with the frame needing to be released. I followed the instructions pretty straightforwardly from there, you will see that little code is changed from what was provided.

needed_info_frames()
This is literally just the formula you provided us, but adjusted from my number of bitmap information frames that could fit into one frame.

Data Structures

ContFramePool * pools
For the most part, things remained similar to simple_frame_pool, with one major distinction: the inclusion of a ContFramePool pointer called pools. This static member was what allowed me to keep track of the separate ContFramePool objects that I had available to me. Initially, I thought I could implement this using a vector, but quickly realized my foolish mistake in seeing that the kernel itself had limited library options when being compiled. Regardless, since most of the code had been working with pointer arrays already, I thought having my own object pointer array would be a fitting addition to the class, and initialized it to the size of the object times the size of a frame, since I thought there could never be more than FRAME_SIZE number of ContFramePools. I believe in the end that was a bit generous, but it served it purpose.

Bitmap structure
I decided on implementing this project using the "extended bitmap" structure suggested to us in the project handout. Seeing as I was now accounting for 2 bits per frame instead of one, I made sure to adjust the math accordingly throughout the program, as I knew that now I would only be able to hold 64MB worth of frames in a single frame if the bitmap were to take one single frame. I also needed to make sure that wherever I checked or cleared one bit in the simple pool program, this program needed to clear and shift 2 bits at a time (unless I was setting a single bit out of 2 for the sequence header).

Additional Information
It should be noted that I am aware that I may be able to check for 2 bits at a time with the mask variable, rather than checking one bit, then shifting over to check the other. I kept the checks the way they were to avoid from oversimplifying and messing up somewhere, just a safety measure for myself.

Also, I have a bit of a habit for choosing short variable names over descriptive ones. Wherever I use "i" it is usually a bitmap_index, and "c" is used as a counter variable for checking contiguous frames. I occasionally use "temp" as a temporary bitmap_index, since I need "i" to stay what it was to add it to the frame_no if checking for contiguous frames succeeds.