Design notes

Mian Qin UIN:725006574

1 Overview

Fundamentally, I use bitmap (2 bits per frame) to manage the continuous frames allocation and deallocation. Each frame metadata (2 bits) has four statuses as follows: The allocation and deallocation algorithm are conduct according to the frame status.
b11 - the frame is free
b10 - the frame is marked as inaccessible
b01 - head frame of the allocated continuous frames
b00 - allocated frames following the head frame
Noted, here we use a special status b10 rather than b00 to indicate inaccessible frames, which is necessary. Because if we use b00 to indicate inaccessible, when the last allocated frame of a allocated frame sequence happens to be the previous frames of the inaccessible frames, when releasing this sequence, it will also release all the inaccessible frames which is logically wrong. So here I use another status b10 to indicate inaccessible frames.

In case of the algorithms for allocating frames. I use **first fit** algorithm which is simple to implement also prove to have good enough performance. The detailed algorithm implementation will be described in the next section.

For the release_frames() api. As the handout note mentioned, the release_frames() api is a static function, we need to determine which frame pool entity owns the released frame sequence. I add a simple signal linked list in to the frame pool class to do the work. A static head pointer member for all the frame pool entity as the first frame pool, each frame pool entity has a next pointer point to the next frame pool entity, When release_frames() is called, in the static function first traverse the linked list to find which frame pool entity owns the sequence. Then call the private member function _release_frames() of that entity to actual perform the release algorithm.

2 Implementation

2.1 constructor

The constructor will first initiate the private members which maintains the information of how many frames in the pool, the start frame of the pool etc.
Then it will calculate how many info_frames need is the bitmap is allocated internally. Then initial the bitmap.
Besides, the constructor also need to link all the existing entities of the frame pool. This is done using the static pointer head and non-static pointer next and the this pointer.

2.2 get_frames

get_frames() use the “first fit” algorithm to allocate continuous _nframes of frames. First, it will traverse the bitmap (2 bits by 2 bits) from the n_info_frames to find _nframes of continuous frames that is in b11 status. Then it will revisit the found _nframes of free frames metadata

and update the bitmap status as b10 followed with _nframes – 1 b00.
Finally, it returns the start frame of the allocated frame sequence.

2.3 mark_inaccessible

mark_inaccessible use a reload member function to mark each frame's matadata as b10.

2.4 release_frames

release_frames() public api is a static function. It will traverse the single linked list that linked all the frame pool entities and find where the release frames sequence belongs to. Then, it will call the private function _release_frames() of that entity to actually perform the release.
The private function will first check the start frame of the release sequence whether it's metadata status is b01. Then it will update the start frame's metadata as b11. Then check the following frame status, if is b00 also update to b11, until hit a non b00 frame and stop.

2.5 needed_info_frames

Calculate the number of info frames, each frame use 2 bits as metadata.

3 Debug and verification consideration

For debugging, I hook up with gdb to enable debug feature. This is already done in MP1.
For verification consideration, since we only have a simple test case in the kernel.C, i.e. a recursive mem_test function. It's not guaranteed that the code is 100% correct even passing the test case. What I did is go through each crucial function get_frames() and release_frames() and watching the bitmap data using gdb to make sure the code is performed correctly.
Also, I add more test code into kernel.C (see github) to test the mark_inaccessible() api to further verify the code.