MACHINE PROBLEM 2

OBJECTIVE - To implement a frame manager, which manages the allocation of frames (physical pages). The frame manager is responsible for the allocation and the release of frames, i.e, it needs to keep track of which pages are being used and which ones are free.

OVERVIEW -
We need to implement the below functions in cont_frame_pool.C to complete the implementation of frame manager -

1) ContFramePool -> class constructor, It initialises the frame pool
2) get_frames -> allocates a number of contiguous frames
3) mark_inaccessible -> Marks a contiguous sequence of frames as inaccessible .
4) release_frames -> Releases a previously allocated contiguous sequence of frames back to its frame pool .
5) needed_info_frames -> Returns the number of frames needed to manage a frame pool of size _n_frames

Additionally, I have added two functions to abstract the usage of bitmap

6) get_state -> to get the state of a frame in bitmap
7) set_state -> to set the state of a frame in bitmap

IMPLEMENTATION -
2 Files changed -
1) cont_frame_pool.C
2) cont_frame_pool.H

I am using a bitmap to store the state of each frame, the possible states are Free, Used and HoS(head of sequence) . As there are 3 possible states we need two bits to store state of each frame and we can store state of upto 4 frames in each unsigned char.

An additional list of pointers to class objects is required to maintain the list of frame pools because release_frames is a static function, to find out the frame pool to which the frames to be removed belong to and to access other member variables/pointers like bitmap etc.
We need to declare these three pointers in the class-

```
static ContFramePool* head;
static ContFramePool* tail;
ContFramePool* next;
```

The head and tail point to the first and last allocated frame pools respectively and the next pointer points to the next member of the list.
Detailed implementation of the functions-
ContFramePool -
1) initialises all the variables/pointers.

2) if info_frame_no is zero we keep the management info in the first frame, otherwise at the provided frame location and we assume that it is valid.
3) all the frame states are set to free except the first one(set to used) which stores management info if info_frame_no is zero.
4) update the pointers in the frame pool list i.e. head, tail, next.
get_frames-
1) Traverses the frame pool and searches for the first available spot to fit in the required no. of frames. This can run in O(n) time, n is the length of the frame pool.
2) marks the first frame state as HoS and the rest as Used.
3) returns the first frame's frame number.
4) 0 is returned if space is not found.
mark_inaccessible-
1) Marks the first frame as HoS.
2) Marks the remaining frames as Used.
release_frames-
1) traverses the frame pool list from head to tail to find the frame pool which contains the given first frame and it also checks whether all the frames to be removed are within the range of framepool.
2) checks if the first frame is HoS, if yes changes the state of the frames to Free.
needed_info_frames -
Simply returns the number of frames needed to manage a frame pool of size _n_frames.

Note:appropriate range checks on inputs like base frame no and frame number are done wherever required.

TESTING:

1) Testing is done using the test memory function in kernel.C for both kernel and process memory pool.
2) tested all the functions by calling functions in kernel.C locally(not uploaded to github) in a random fashion. Identified bugs and fixed them.