CSCE 611
MP2: Design Document

Shweta Sharma
UIN: 433003780

Problem Statement:

The objective of machine problem 2 is to implement a simple frame pool manager which manages allocation and release of frames.
The following constraints are considered while implementing the frame pool manager:
1) Total Memory available in the machine = 32 MB
2) Kernel Memory Pool Size = 4 MB ; Location : 512B -1023B
3) Process Memory Pool Size = 28MB ; Location : 1024B – 8191B
4) Frame Size : 4KB = 4 *1024 * 8 = 32K bits

Other implementations taken during implementation:

1) Mapping the frame pool to a bit-map.
2) A frame takes 8 bits of memory, the first bit representing if the frame is allocated or not and the second bit representing if it is head of a sequence and remaining 6 bits are free.
To use only 2 bits per frame, we use this pattern:
1. 11 -> Frame is free
2. 00 -> Frame is allocated
3. 10 -> Frame is the first frame of contiguous frames (head frame) and it is allocated

Methods to be implemented:
1. **Contiguous Frame Pool Constructor:** This initializes the data structure in our case a linked list node for contiguous frame pool from the given starting base frame number till base frame number plus the number of frames. For this range of frames, it marks the corresponding bitmap indexes to be free initially. We also consider the info frame number which is the frame number used to store management information for frames. If this number is set to zero, we by default assume the first base frame number to store the management information.
2. **get_frames**: This function takes in the number of frames requested as a parameter and returns the head frame number of the newly allocated block of frames.
   The detailed steps are as follows:
   a. FIrst, it checks if there are enough frames free to allocate. If there is, the control continues throughout the functions else it causes an error.
   b. Next, we loop in to check the first available free frame in the memory using our bit patterns mentioned above.
   c. After we have found the first frame, we check if there are enough contiguous frames from the first frame i.e. there are free frames from first frame to first frame+number of frames.
   d. If there is , we set this entire frame as used in the bitmap, and return the head frame.

e. If there isn't a contiguous free space for all the frames, we return to step b. To find the next free available frame and continue from there. If during this process, we try to access an address higher than the available memory, our function throws an exception.

3. **mark_inaccessible**: This function takes the base frame number and number of frames function parameters and marks this region of frames as inaccessible. The detailed steps are:

a. We iterate through the range of frame numbers given and check if this is available in the given memory constraints. If not, the function throws an error.
b. If this range of frames is valid, we mark them as used using bit masking.
c. If some values are already marked as used in this region, we throw an error.

4. **release_frames**: This function takes in the head frame number as one of the function parameters and releases contiguous blocks of frames that starts with this particular frame passed in the function. The detailed steps are:

a. We iterate through the list of frame pools to check which frame pool our frame belongs to, namely process pool or kernel pool.
b. Once we have found the frame pool, we check if the frame mentioned is a head frame or not. If it is not a head frame, we throw an error.
c. Else, we start deallocating frames that start with the head frame till we find the next head frame in the memory.
d. In this way, we released an entire block of frame from the memory.

5. **needed_info_frames**: This function takes in the total number of frames as argument and returns the number of info frames required to store the total number of frames. We have already assumed our frame size to be 32K, assuming that we need 8 bits to store a frame, our function implements a general formula to calculate the needed info frames.

The formula is:
n_info_frames = [(_n_frames * _n_info_bits)/ Frame size in bits] + [(_n_frames % n_info_bits)/Frame Size) > 0 ? 1 :0]

In our case _n_info_bits is 8,
So, the formula implemented by our function is:
n_info_frames = [(_n_frames *8)/ Frame size in bits] + [(_n_frames %8)/Frame Size) > 0 ? 1 :0]

We also put the remainder of the total frames in a new info frame due to which we use the mod operator.