MP2: Frame Manager

Dhanraj Murali UIN: 734003894

CSCE611: Operating System

# Assigned Tasks:

**Main:** Completed.

# System Design:

1. Total memory size = 32MB
2. Memory reserved for kernel = 4MB
3. Memory partition to be used by kernel = 2MB to 4MB
4. Memory partition to be used by process = Above 4MB

# Code Description:

1. Bitmap is maintained to effectively maintain the state of the frames in the given memory pool. Bitmap is char array with each element comprising of 8 bits. We use 2 bits to maintain the state of each frame, which means status of 4 frames in each byte.
2. We also maintain a linked list of memory pools allocated for each requesting process.

## Files modified:

1. cont\_frame\_pool.H
2. cont\_frame\_pool.C
3. kernel.C – modified to test different sizes of frames. Reverted to original file.

All changes made as part of MP1 bonus to enable GDB integration in the following files are also included in this repository as well.

1. Makefile
2. linker.ld

## Function Description:

## **get\_state(unsigned long \_frame\_no)**

**Logic Used:**

T`he input \_frame\_no is converted to the index that can be used to access the bitmap array by dividing it by 4. The index is then used to access the byte where the state of the frame is stored. The position of the frame information is calculated by taking the reminder of the above calculation. We use a mask of value 0x3 which is equivalent to 11 in binary. The mask is then shifted to match the position of the state information of the frame in the bitmap byte. The mask is then multiplied with the bitmap to remove the frame state information of the frames that are stored in the byte so we just have the state of the frame in interest. The manipulated byte with only the frame state information of one frame is then right shifted to move the frame state information to the LSB of the byte. It is then compared with the values of the frame states defined earlier – Free, Used and HoS, and the frame state is returned.

1. **set\_state(unsigned long \_frame\_no, FrameState \_state)**

**Logic Used:**

The input \_frame\_no is converted to the index that can be used to access the bitmap array by dividing it by 4. The index is then used to access the byte where the state of the frame is stored. We use the mask similar to get\_state function and shift it to match the location of the frame of interest. The input \_state is then checked to see the state that is to be set in the frame state information bit. Then the bitmap byte is multiplied with the complement of the mask to change the state information of the frame in interest to 00. Then it is added with the desired frame state value as per the input \_state.

1. **get\_frames(unsigned int \_n\_frames)**

**Logic Used:**

We use 2 loops to iterate through the bitmap of the frame pool in interest to check if we have contiguous free frames to allocate. The first loop starts from frame 0 and starts to check if it is free, if a free frame is located in the frame pool then control enters the 2nd loop to start checking contiguous free frames of interested size. If no such contiguous locations are available, we just oprint that we don’t have enough frames. If such contiguous free frames are present, then we set the new states of the frames, with HoS for the first frame and Used for the rest of the frames. Then we return the frame number of the first free frame of the sequence in its physical memory mapping by adding it to the base frame number of the pool.

1. **mark\_inaccessible(unsigned long \_base\_frame\_no, unsigned long \_n\_frames)**

**Logic used:**

We first check if the provided input \_base\_frame\_no belongs to the frame pool of the current instantiation of the class. If it is part of the current frame pool then we mark the input \_base\_frame\_no as HoS and rest of the frames in the mentioned limit \_n\_frames as Used so that they are inaccessible.

1. **release\_frames(unsigned long \_first\_frame\_no)**

**Logic Used:**

The function is static so we cant call the function using an instantiation i.e a specific memory pool of the class type. So we have to find the memory pool of interest meaning the memory pool in which the \_first\_frame\_no is located. This is done by checking if the frame belongs to the memory pool of the first node in the linked list that is used to maintain the allocated pools. Once the memory pool of interest is found, then we start setting the state of the particular pool until we encounter a frame with a frame state or a frame with HoS state.

1. **needed\_info\_frames(unsigned long \_n\_frames)**

**Logic Used:**

Here, the info frames are nothing but the number of frames required to store the bitmap of frame states. The frame size is the same as the machine’s page size and we also calculated earlier that for each frame we need 2 bytes of information. For example, if we take a frame size of 4 kilobytes which is equal to 4096bytes, we can store the frame state information of 4\*4096 frames in 1 frame. So, for any process, the required no of frames is calculated by dividing the memory size by frame size. To find out the no of frames required to store the frame state information, simply divide the number of frames by 4\*frame size. Suppose if there any remainder then add one more frame to the required info frames value to accommodate the frame state of the remainder frames.

# 

# Testing:

