Machine Problem 2
FRAME MANAGER

CSCE 410/611: Operating Systems
Fall 2022

by
Ritika Nigam
UIN: 633002197

Objective:

The main objective of this machine problem is to build a frame manager for allocation and management of continuous frame pool in kernel memory. The total system memory available is 32 MB, of which 4 MB is used by kernel and a 1MB hole at 15MB which is inaccessible. The frame size used is 4KB.

Implementation:

The implementation of the frame pool manager is divided into several modules as API's in the frame manager. The different API's are as follows:

- Class Constructor – ContFramePool()
- get_frames
- mark_inaccessible
- release_frames
- needed_info_frames

From these, the release_frames() api is a static method and can be called without any object of the class. As we have 4 different mark to set on a frame, we will use 2 bits to represent one frame. Following represents how they are assigned in this implementation:

- 01 - Head of frame
- 11 - ALLOCATED
- 00 - FREE
- 10 – Inaccessible

Next we define the private variables to be used in the class ContFramePool. Apart from the regular variable needed for frame counts, I added a singly linked list to track the number of framepools being instantiated. These are static lists and can be used by the release_frames api to determine which frame pool manages the particular frame and then act accordingly. I have used just one bitmap to track the allocated frames and their management information using the values mentioned above.

Let us now describe the implementation with each of the above API:

a. **Constructor**: This API takes all the arguments and assign them to private variables. It sets the bitmap pointer based on info_frame_no. It will then update the list of objects that have been created from this class. So, we basically update the list and next pointer for same.
b. **get_frames:** This is the core API of this whole implementation. This will determine if we have the number of continuous frames available in the frame pool. The search algorithm is optimized to run in O(n) time, i.e. the length of frame pools. It does a linear search on the frames and determines if we have a continuous length of those frame available. If found,

it will update the first frame as head of the sequence and then mark rest of them as allocated. It then returns the frame number at which we found the empty sequence.

c. **mark_inaccessible:** This API basically marks the set of frames as inaccessible (10). It also checks whether the range of frames given is within the number frames managed by this frame pool.
d. **release_frames**: This API frees the allocated frames and marks them as free(00) to be used by other process. The first part of this API is to determine the frame pool from which this frame will be released. To do that, we access the static list of frame pools to check their range and then use its bitmap pointer. The marking of free frames continues till we have allocated frames.
e. **needed_info_frames**: This API simply returns the number of management frame needed for a particular memory size.