Design Document
Continuous Frame Pool Manager

CSCE611
Operating Systems
MP2- Fall 2022

by

Tanu Shree

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

TEXAS A&M UNIVERSITY, COLLEGE STATION

This machine problem requires a frames manager which is responsible for the allocation and release of frames. Since different parts of the memory is used by different parts of operation system, the frame manager will manage a collection of frame pools which supports get and release operations of frames.

The total system memory available is 32 MB, out of which first 4MB are reserved for kernel (The actual code starts at 1MB). There is one 1 MB hole at 15MB which is inaccessible. The frame size used is 4KB.

We have four different statuses for the frames; Free, Used, HoS and Inaccessible. Therefore, we will use 2 bits to represent one frame. I have used the statuses in following format:

1. 0x0: Free
2. 0x1: Used
3. 0x2: HoS
4. 0x3: Inaccessible

In this Machine problem we are implementing following API:

1. Class Constructor – ContFramePool()
2. get_frames
3. mark_inaccessible
4. release_frames
5. needed_info_frames

As a part of this MP, I have made changes in following files:

- **cont_frame_pool.H**

    1. Defined the frame pool data structure.
    2. Added HoS and Inaccessible as frame state to the enum.
    3. Defined a single linked to store the list of pools.

- **cont_frame_pool.C**

    1. **set_status** and **get_state**: Defined to set and get the status of the frames as required. For that I used switch statement to set and get the different status (Free, Used, HoS and Inaccessible). These functions are used to make the code DRY, so that the repletion is avoided and the changes can be done easily if required.

    2. **Constructor**: It takes all the arguments and assign them to the private variables. It first sets all the frames free and then sets 1st frame to 'Used' (based one info_frame_no, here 0). It also updates the list of pools and its next pointer. And hence frame pool is initialized here.

3. **get_frames**: This is the most important API of the whole implementation. It does the linear search to get the available frames and once the required number of free continuous frames are found, the first frame is set to 'HoS' and the rest are set to 'Used'. Along with that the number of free frames is also updated. It return the first frame (marked HoS) of requested allocation frames.

4. **mark_inaccessible**: This is like get_frames. In this the allocator knows, which frame exactly is to be marked HoS and how many after that are to be marked 'Used'.

5. **release_frames**: This API releases the allocated frames and mark them 'free', so that they can be used by the other process. Since this function is static, we first need to know, in which pool the requested release frame is located. For that we use the static single linked list of pools. Once we know the pool, we first check whether the first frame is marked 'HoS'. If yes, we traverse the subsequent frames until we find the next free frame, or the frame marked 'HoS'. Till then, we mark the frames 'free'.

6. **Needed_info_frames**: This API returns the number of management frames needed for a memory size.

Testing:

For testing and debugging the code I created a temporary void function called print_frames(). This function prints the state of all the frames ranging from 0 to 3 (free-0, used-1, HoS-2, Inaccessible-3). This helps to check the algorithm for accessing 2bits at a time and performing the bitwise operation on them to set/get the status of the frames.

I tested each API during their individual implementation. Test_memory function was already present in kernel.C file to test the memory allocations. During the test, it failed several times giving the incorrect allocation error. With several debugs using print function, I could figure out the problem (the problem was in the get_frames function) and resolved the issue.

Screenshot of the successful run: