CSCE - 611

MACHINE PROBLEM 2: FRAME MANAGER

DESIGN DOCUMENT

Submitted by: RAJENDRA SAHU (731008796)
github-repo : CSCE-410-611-fall-2021/rpsahu_CSCE611

INTRODUCTION:

This frame manager is responsible for the allocation and release of physical memory frames. The context & code layout had already been established by the *handout_mp2.pdf* & *MP2_Source.zip*. This document is intended to give the reader an idea/logic of the implementation behind the frame manager without having to deduce it from the code which generally should be avoided. The source codes submitted with this guide also has self-explanatory comments aiming for a good code documentation which will turn helpful in future as we reuse this frame manager for upcoming machine problems.

SOURCE CODES MODIFIED:

My design took some idea from the source codes *simple_frame_pool.H/C* to figure out the frame & bitmap index standards. The gist of single frame allocation & release in this source codes certainly helped me to code the actual continuous frame allocator. However, the actual design implementations only deal with the following 3 source codes:

1. *kernel.C : uncommenting process_pool creation & testing*
2. *cont_frame_pool.H : core implementation*
3. *cont_frame_pool.C : core implemenrtation*

ABSTRACTIONS INTRODUCED:

1. **STATE LOGIC**:
   Each allocation or release request would happen with a call providing frame_no & the no_of_frames. Hence, we need to maintain a head state that would denote that a particular frame is the head frame of a sequence of frames. This idea was borrowed from the explanation in *cont_frame_pool.C.*
   So each frame should have 3 states i.e., FREE, ALLOCATED_BUT_NOT_HEAD, ALLOCATED_AND_HEAD. 2 bits help us achieve these 3 states.
   Zeroth bit -> Free or not
   First bit -> Head of Sequence
   Take note both bits can't be set which would indicate head of a free sequence of frames. It doesn't make a logical sense; hence a head of a sequence frame will have its FREE bit cleared.
   These states have been denoted in #defines in *cont_frame_pool.C.*

2. **TYPEDEF STRUCTURE**: We have 2 bits of state information that we need to manage per frame. We choose an *unsigned char* structure to pack 4 of this 2-bit states by using the bitfield token ":". So, 1 pointer of this structure would be pointing to information of 4 frames.
   This has been implemented in DATASTRUCTURES section in *cont_frame_pool.C.* under the name of *bitmop_char_s.* We create a pointer of this structure in *ContFramePool* class to store this management information specific to each frame pool.

3. **HELPER FUNCTIONS**: A set of helper functions were introduced on top of the main implementation functions to keep the code dry & promote reusability. The following are the helper functions.

a. ***unsigned char getState(unsigned long _frame_index_4_multiple, unsigned short _bitmap_index):*** Getter function to retrieve the 2 bit state of a particular frame which can be referenced by the given parameters.
b. ***void setState(unsigned long _frame_index_4_multiple, unsigned short _bitmap_index, unsigned char val); :*** Setter function to set a particular state in the 2 bit container. Since it's just a 2 bit implementation, logical AND OR operations were not really required concerning to preserve the state of other bits.
c. ***Parameters in above 2 functions:***
_frame_index_4_multiple -> index to refer the char that contains frame state out of the char array
_bitmap_index -> bitmap index to refer the actual state from the 4 states present in 1 char
Since the state is 2 bits & char is 8 bits the factor of 4 comes into picture.
d. ***bool isFree(unsigned long _frame_index);*** : To check if a singular frame is free or not.
e. ***void allocate(unsigned long _frame_index);*** : To allocate a singular frame also has support for head of frame sequence.
f. **void release(unsigned long _frame_index);** : To release a single frame
g. The above 3 helpers internally call the getter/setter pair. One thing that is observation worthy is that we pass appropriate calculation of divide & modulus of *_frame_index* by 4 in the parameters to the getter/setter pair. This is because we are packing 4 2-bit state info in 1 8-bit char.

4. **FRAME POOL LIST:**
We implement a list to store the frame pool objects. This list is implemented by using a static head pointer & non static next pointer in *ContFramePool* class. We traverse the list using the next pointer. This list will be useful when we want to release a sequence of frames without having to know whether the frame pool is the owner of the head frame.

CORE IMPLEMENTATION:

This includes the actual implementation of the *ContFramePool* class (members & functions).

1. ***ContFramePool()* constructor** : The constructor does the usual initialization of the data members through the passed parameters. On top of this it does the following:
    a. Update the bitmap pointer to point to the appropriate address based on the condition *info_frame_no* == 0 or not.
    b. Set all the frame states to FREE by using setState setter. This is the starting reference state of the memory we intend to create.
    c. setState() to ALLOCATE to the frames for management info based on the condition *info_frame_no* == 0 or not.
    d. Updating the frame pool list by appending the current new frame pool at the tail.
2. ***ContFramePool::get_frames():*** This works on the logic of traversing the entire frame pool to find the first free frame, then checking if it has required no of additional free frames contiguously next to it. If the condition is satisfied it picks up frame that frame index and calls *allocate()* helper to allocate one frame by one.
3. ***ContFramePool::mark_inaccessible():*** First do a range check if the provided frame range lies in the frame pool or not , then allocate that range one by one .
4. ***ContFramePool::release_frames() :*** This is the static function common to every frame pool. The function is just provided a first frame number of a previously allocated contiguous sequence of frames. It first traverses through the frame pool list being maintained and does a range check to find out to which frame pool does this first frame number belongs to. Once that is found out the frame pool specific *release_frames_from_pool()* is called.
5. ***ContFramePool::release_frames_from_pool :*** This is the frame pool specific non static function that gets called from the above static function. It works on the logic of calling *release()* helper starting from the first frame number (provided in the static release_frames() call) one by one until it reaches a free frame, releasing the desired sequence of frames.

6. ***ContFramePool:neede_info_frames:*** Returns the number of frames required to store the management info of *_n_frames.* This calculation is based on the FRAME_SIZE and the number of frame state info packed in a 8 bit char bitmap. Refer the comments in the actual function of exact logic.
7. ***kernel.c :*** Uncomment the process pool creation code & increase the allocs_to_go = 128 for a greater exhaustive testing on both the pools.

TESTING:

Since the tester logic recursively allocates, sets 1KB in a frame with a preset expected value, verifies the same segments with the expected value and then releases the frames. I believe this logic should be enough to stress test the frame pools on their frame management functionality.

CODE MAINTENANCE:

The entire source code base has been pushed to the student's github repository. This have been done progressively over the course of development. This should help the TA in grading the assignment. Apart from the code base provided in the zip file submitted on canvas, the code can also be compiled from the github repo code base. Making the code remote should be a good step in maintaining the frame manager for future assignments.

GENERIC INSTRUCTIONS:

Enter the code directory provided in this zip file. This directory has all the needed source files & the makefile for a successful compilation. Type make, copy the kernel to bochs img file & run the bochs simulation.
The console output should reach the message:

"`Testing is DONE. We will do nothing forever.`
`Feel free to turn off the machine now.`"