MP2: Frame Manager

Ashutosh Punyani
UIN: 834006613
CSCE611: Operating System

Assigned Tasks

Main: Completed.

System Design

The main objective of machine problem 2 is to implement continuous frame pool manager which is responsible for the allocation and release of frames which are used by kernel frame pool and process frame pool.

This manages kernel frames 2MB - 4MB and process frames between 4MB - 32MB. It uses bitmap where two bits represents the states of the frame. As two bits are used to store the state, so one byte of the bitmap can store the states of 4 frames. We have 3 different states for each

Value	**State**	**Description**
0X0	Free	Represents a free frame
0X1	Used	Represents an allocated non-head frame
0X2	Head Of Sequence	Represents an allocated head of the frame

Code Description

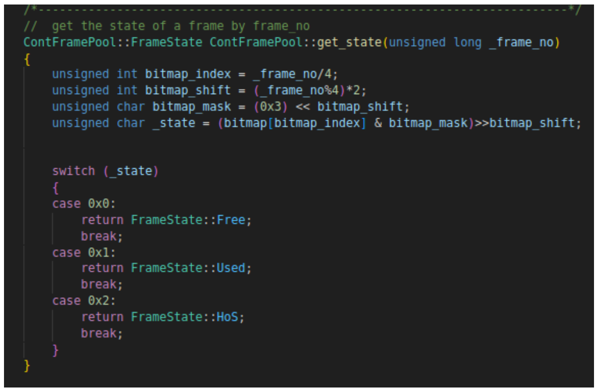
During the implementation of this machine problem I have made changes to the following two files :

1. **cont_frame_pool.H**
2. **cont_frame_pool.C**

In `cont_frame_pool.H` I have define the data structure to be used each contiguous frame pool. In addition to this I have pointer of the same class to hold the address of the next frame pool.

In this machine problem I have implemented the following functions definition in `cont_frame_pool.C`:

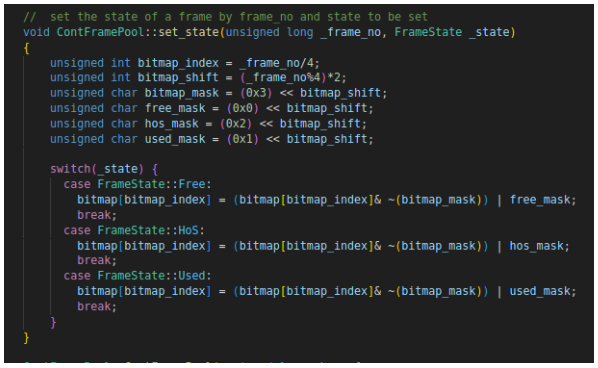
1. **get_state** : This function is used to get the state of a particular frame. This uses bit operations for getting the state of a frame. Resultant of the bit operations is then passed to switch statement.



```cpp
/*---------------------------------------------------------------------------*/
//  get the state of a frame by frame_no
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no)
{
    unsigned int bitmap_index = _frame_no/4;
    unsigned int bitmap_shift = (_frame_no%4)*2;
    unsigned char bitmap_mask = (0x3) << bitmap_shift;
    unsigned char _state = (bitmap[bitmap_index] & bitmap_mask)>>bitmap_shift;


    switch (_state)
    {
    case 0x0:
        return FrameState::Free;
        break;
    case 0x1:
        return FrameState::Used;
        break;
    case 0x2:
        return FrameState::HoS;
        break;
    }
}
```

Figure 1: **get_state**

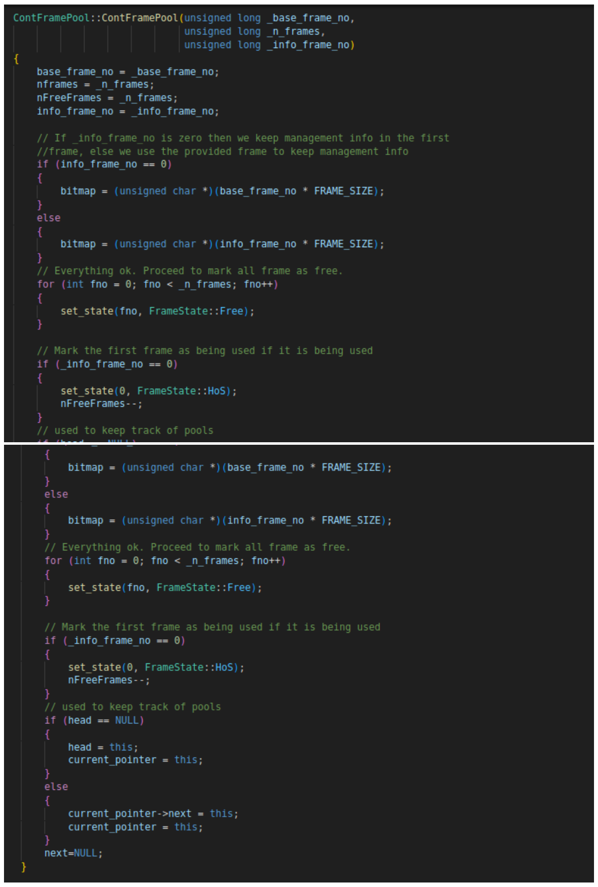
2. **set_state** : This function is used to set the state of a particular frame with a state passed as parameter . This uses bit operations for setting the state of a frame. Switch statement is used to set the state of a frame.



```cpp
//  set the state of a frame by frame_no and state to be set
void ContFramePool::set_state(unsigned long _frame_no, FrameState _state)
{
    unsigned int bitmap_index = _frame_no/4;
    unsigned int bitmap_shift = (_frame_no%4)*2;
    unsigned char bitmap_mask = (0x3) << bitmap_shift;
    unsigned char free_mask = (0x0) << bitmap_shift;
    unsigned char hos_mask = (0x2) << bitmap_shift;
    unsigned char used_mask = (0x1) << bitmap_shift;

    switch(_state) {
      case FrameState::Free:
        bitmap[bitmap_index] = (bitmap[bitmap_index]& ~(bitmap_mask)) | free_mask;
        break;
      case FrameState::HoS:
        bitmap[bitmap_index] = (bitmap[bitmap_index]& ~(bitmap_mask)) | hos_mask;
        break;
      case FrameState::Used:
        bitmap[bitmap_index] = (bitmap[bitmap_index]& ~(bitmap_mask)) | used_mask;
        break;
    }
}
```

Figure 2: **set_state**

3. **ContFramePool (Class Constructor)** : Constructor is used to set the data structure of a frame pool with all the frames in free state. Based on the parameter `_info_frame_no` value, management info is stored either in the first frame or use the provided frame in the parameter. In addition to this linked list is also created to keep the track of pools.



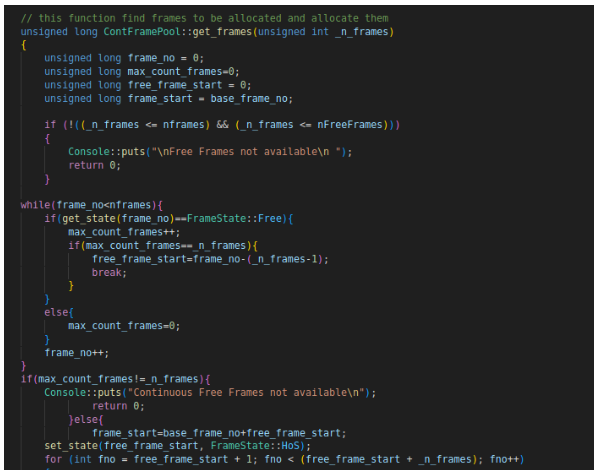
```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    base_frame_no = _base_frame_no;
    nframes = _n_frames;
    nFreeFrames = _n_frames;
    info_frame_no = _info_frame_no;

    // If _info_frame_no is zero then we keep management info in the first
    //frame, else we use the provided frame to keep management info
    if (info_frame_no == 0)
    {
        bitmap = (unsigned char *)(base_frame_no * FRAME_SIZE);
    }
    else
    {
        bitmap = (unsigned char *)(info_frame_no * FRAME_SIZE);
    }
    // Everything ok. Proceed to mark all frame as free.
    for (int fno = 0; fno < _n_frames; fno++)
    {
        set_state(fno, FrameState::Free);
    }

    // Mark the first frame as being used if it is being used
    if (_info_frame_no == 0)
    {
        set_state(0, FrameState::HoS);
        nFreeFrames--;
    }
    // used to keep track of pools
```

```
    {
        bitmap = (unsigned char *)(base_frame_no * FRAME_SIZE);
    }
    else
    {
        bitmap = (unsigned char *)(info_frame_no * FRAME_SIZE);
    }
    // Everything ok. Proceed to mark all frame as free.
    for (int fno = 0; fno < _n_frames; fno++)
    {
        set_state(fno, FrameState::Free);
    }

    // Mark the first frame as being used if it is being used
    if (_info_frame_no == 0)
    {
        set_state(0, FrameState::HoS);
        nFreeFrames--;
    }
    // used to keep track of pools
```

```
    if (head == NULL)
    {
        head = this;
        current_pointer = this;
    }
    else
    {
        current_pointer->next = this;
        current_pointer = this;
    }
    next=NULL;
}
```

Figure 3: **ContFramePool (Class Constructor)**

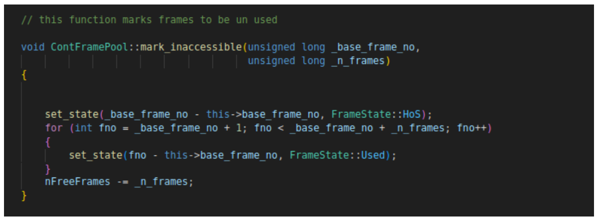
4. **get_frames** : This function is used to search the contiguous sequence of frames with a size of `_n_frame`. Based on whether the contiguous frames are available or not the function allocates the frames and set bitmap as Head of sequence(HoS) or Used(Used). First frame is set as HoS and remaining `_n_frame - 1` as Used



```
// this function find frames to be allocated and allocate them
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
    unsigned long frame_no = 0;
    unsigned long max_count_frames=0;
    unsigned long free_frame_start = 0;
    unsigned long frame_start = base_frame_no;

    if (!((_n_frames <= nframes) && (_n_frames <= nFreeFrames)))
    {
        Console::puts("\nFree Frames not available\n ");
        return 0;
    }

while(frame_no<nframes){
    if(get_state(frame_no)==FrameState::Free){
        max_count_frames++;
        if(max_count_frames==_n_frames){
            free_frame_start=frame_no-(_n_frames-1);
            break;
        }
    }
    else{
        max_count_frames=0;
    }
    frame_no++;
}
if(max_count_frames!=_n_frames){
    Console::puts("Continuous Free Frames not available\n");
            return 0;
        }else{
            frame_start=base_frame_no+free_frame_start;
    set_state(free_frame_start, FrameState::HoS);
    for (int fno = free_frame_start + 1; fno < (free_frame_start + _n_frames); fno++)
```

Figure 4: **get_frames**

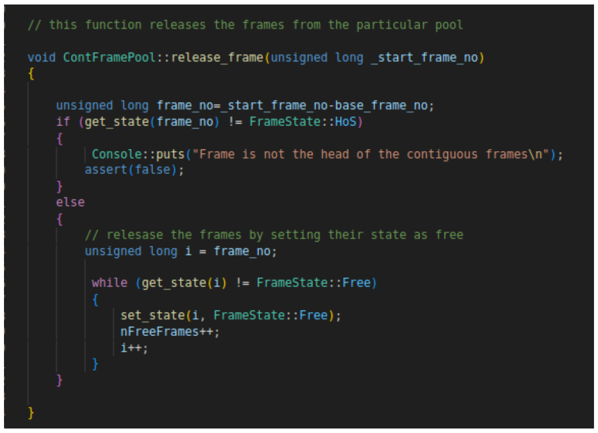
5. **mark_inaccessible** : This function is used to mark the particular range of contiguous sequence of frames as inaccessible passed as the parameters.



```
// this function marks frames to be un used

void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{


    set_state(_base_frame_no - this->base_frame_no, FrameState::HoS);
    for (int fno = _base_frame_no + 1; fno < _base_frame_no + _n_frames; fno++)
    {
        set_state(fno - this->base_frame_no, FrameState::Used);
    }
    nFreeFrames -= _n_frames;
}
```

Figure 5: **mark_inaccessible**

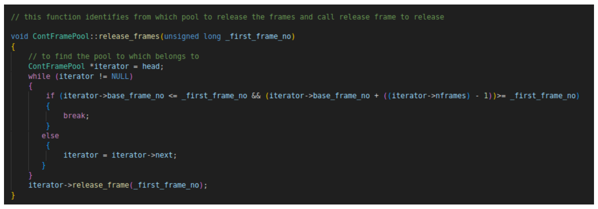
6. **release_frame** : This function is used to release a frame from the particular pool.



```
// this function releases the frames from the particular pool

void ContFramePool::release_frame(unsigned long _start_frame_no)
{

    unsigned long frame_no=_start_frame_no-base_frame_no;
    if (get_state(frame_no) != FrameState::HoS)
    {
         Console::puts("Frame is not the head of the contiguous frames\n");
        assert(false);
    }
    else
    {
        // relesase the frames by setting their state as free
        unsigned long i = frame_no;

         while (get_state(i) != FrameState::Free)
         {
             set_state(i, FrameState::Free);
             nFreeFrames++;
             i++;
         }
    }

}
```

Figure 6: **release_frame**

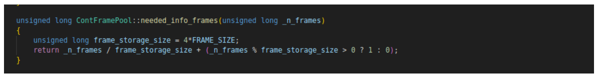
7. **release_frames** : This function is used to identify the particular frame in which the given frame number belongs to. For this, linked list is used to the search the frame pool to which this frame number belongs to.



```
// this function identifies from which pool to release the frames and call release frame to release

void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    // to find the pool to which belongs to
    ContFramePool *iterator = head;
    while (iterator != NULL)
    {
        if (iterator->base_frame_no <= _first_frame_no && (iterator->base_frame_no + ((iterator->nframes) - 1))>= _first_frame_no)
        {
            break;
        }
       else
        {
            iterator = iterator->next;
        }
    }
    iterator->release_frame(_first_frame_no);
}
```

Figure 7: **release_frames**

8. **needed_info_frames** : This function returns the number frames to manage a frame pool of size `_n_frames`.

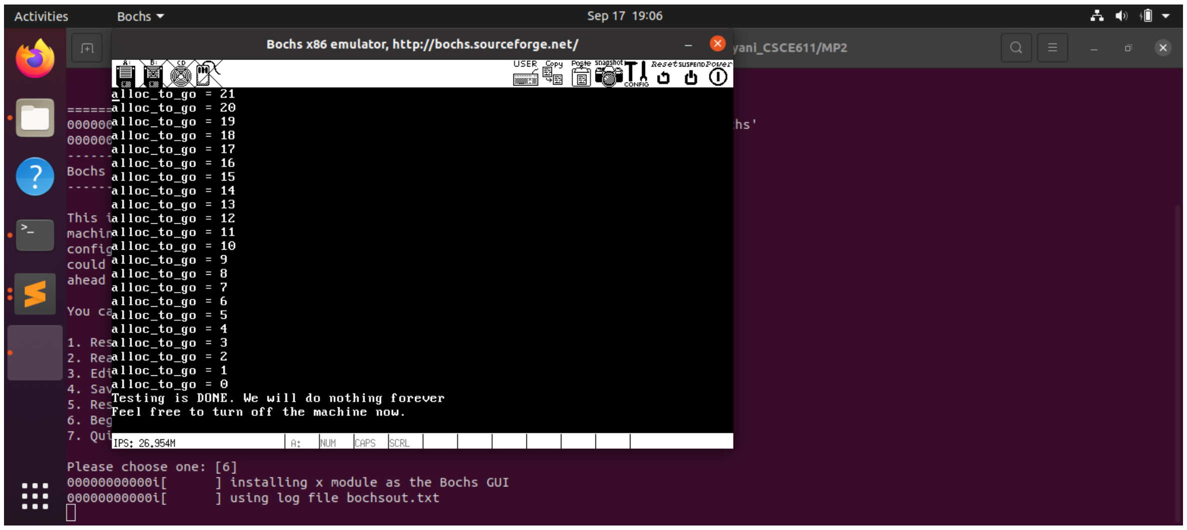


```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    unsigned long frame_storage_size = 4*FRAME_SIZE;
    return _n_frames / frame_storage_size + (_n_frames % frame_storage_size > 0 ? 1 : 0);
}
```

Figure 8: **needed_info_frames**

Testing

During the development of code I wrote several `Console:puts()` and `Console:putui()` to identify where my code was breaking and to understand where logic is wrong or not performing the way it should implement. During testing, Kernel.C has thrown the error `MEMORY TEST FAILED. ERROR IN FRAME POOL` several times I also used `test_memory` function written in Kernel.C for testing by changing the values `_allocs_to_go`. After the code ran successfully I removed all Console statements and changes back the Kernel.C to the original one.



```
alloc_to_go = 21
alloc_to_go = 20
alloc_to_go = 19
alloc_to_go = 18
alloc_to_go = 17
alloc_to_go = 16
alloc_to_go = 15
alloc_to_go = 14
alloc_to_go = 13
alloc_to_go = 12
alloc_to_go = 11
alloc_to_go = 10
alloc_to_go = 9
alloc_to_go = 8
alloc_to_go = 7
alloc_to_go = 6
alloc_to_go = 5
alloc_to_go = 4
alloc_to_go = 3
alloc_to_go = 2
alloc_to_go = 1
alloc_to_go = 0
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.
```

```
Please choose one: [6]
00000000000i[      ] installing x module as the Bochs GUI
00000000000i[      ] using log file bochsout.txt
```

Figure 9: **Testing**