MP2: Frame Manager

Nachiket Umesh Naganure
UIN: 532008698
CSCE611: Operating System

Assigned Tasks

Main: Completed.

System Design

The goal of the machine problem was to design a frame manager, which will manage allocation of contiguous frames in physical memory. It will keep track of which frames are being used and also the ones which are free. The frame manager will manages these frames in groups called frame pool (Kernel pool and process pools).

1. One needs to maintain the states of all the frames that the frame manager manages. We make use of a BITMAP for maintaining the states of the frame.
2. We use two bits to represent the state of the frame. We have four states to represent and use binary encoding for representation.
3. Each frame can one of the following states
    (a) Free (11) : This frame is free and can be allocated upon requested.
    (b) Used (10): This frame is allocated
    (c) HoS (01): This frame is allocated and is the first of the contiguously allocated frames
    (d) Inaccessible (00): Frame is forbidden from being allocated for normal usage.
4. We make use of bit manipulation to access the two bits representing the frame in the character bitmap. Each character space has 8 bits which can support 4 frames.
5. **Allocation :**
    (a) Upon a request to allocate say n frames traverse the character bitmap to find contiguous n frames which are free to allocate and flip the bits that represent the frames to set them to "Used" state
6. **Deallocation :**
    (a) For deallocation, we need the frame number belonging to the first frame of the contiguous frames which would have HoS (Head of Sequence) state. We set it to free.
    (b) We then start freeing the frames which which follow the HoS frame till we encounter another new HoS frame.

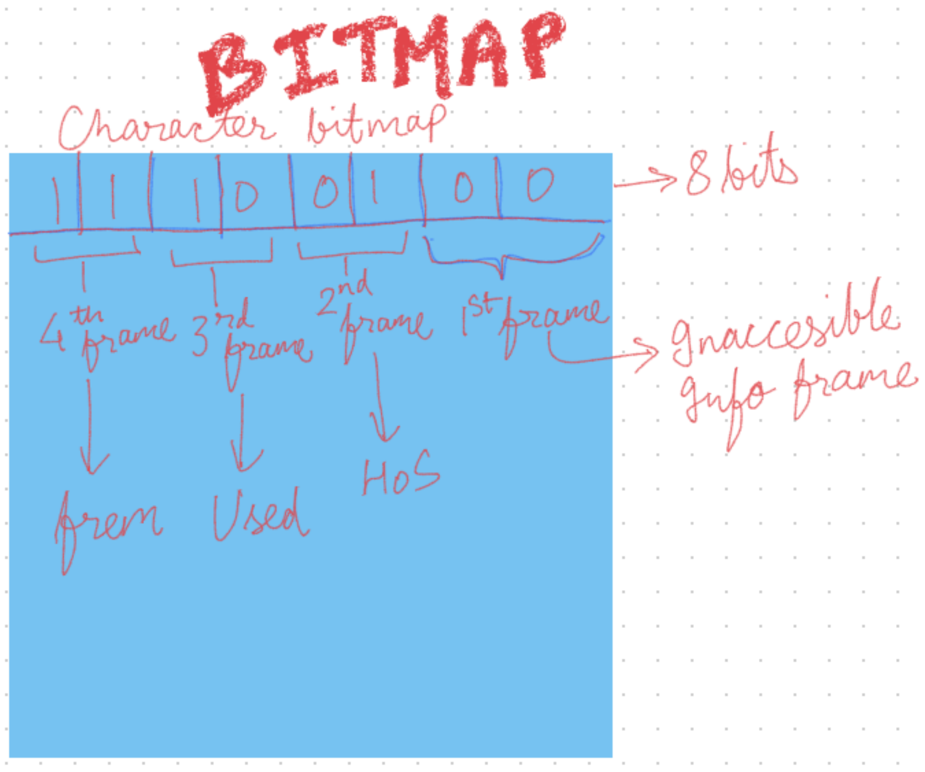
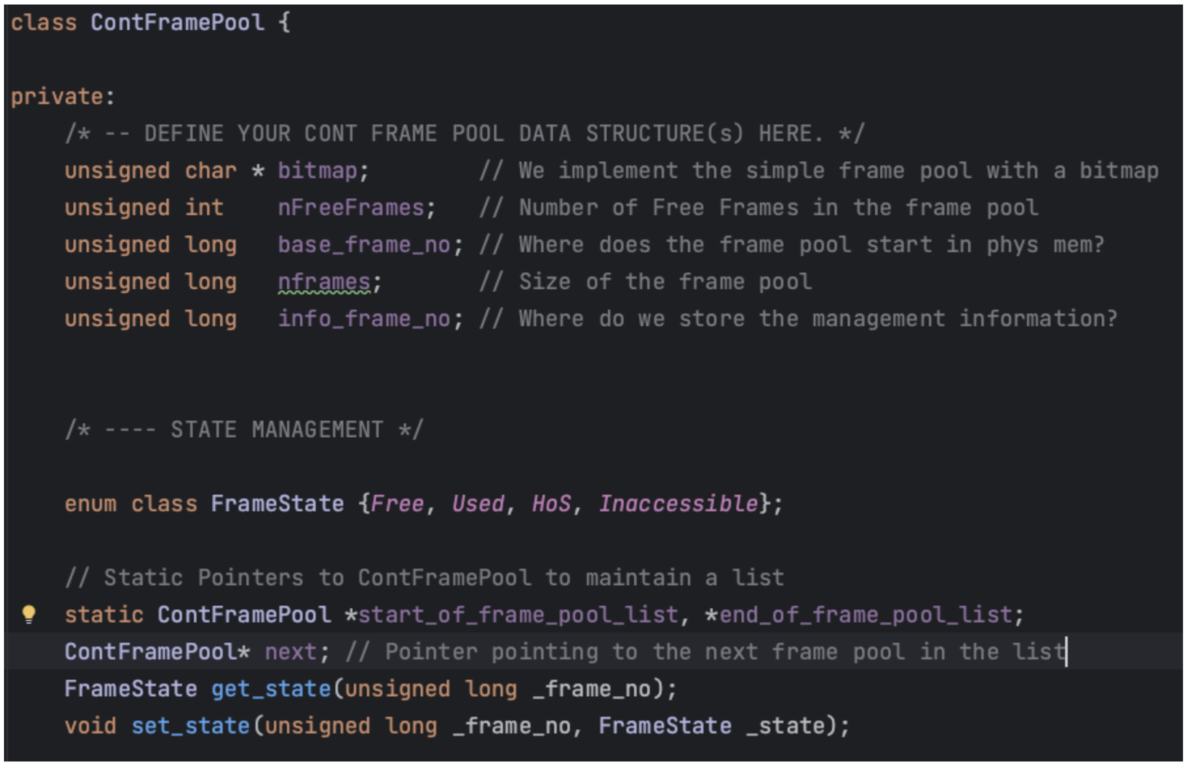


Figure 1: Character bitmap

Code Description

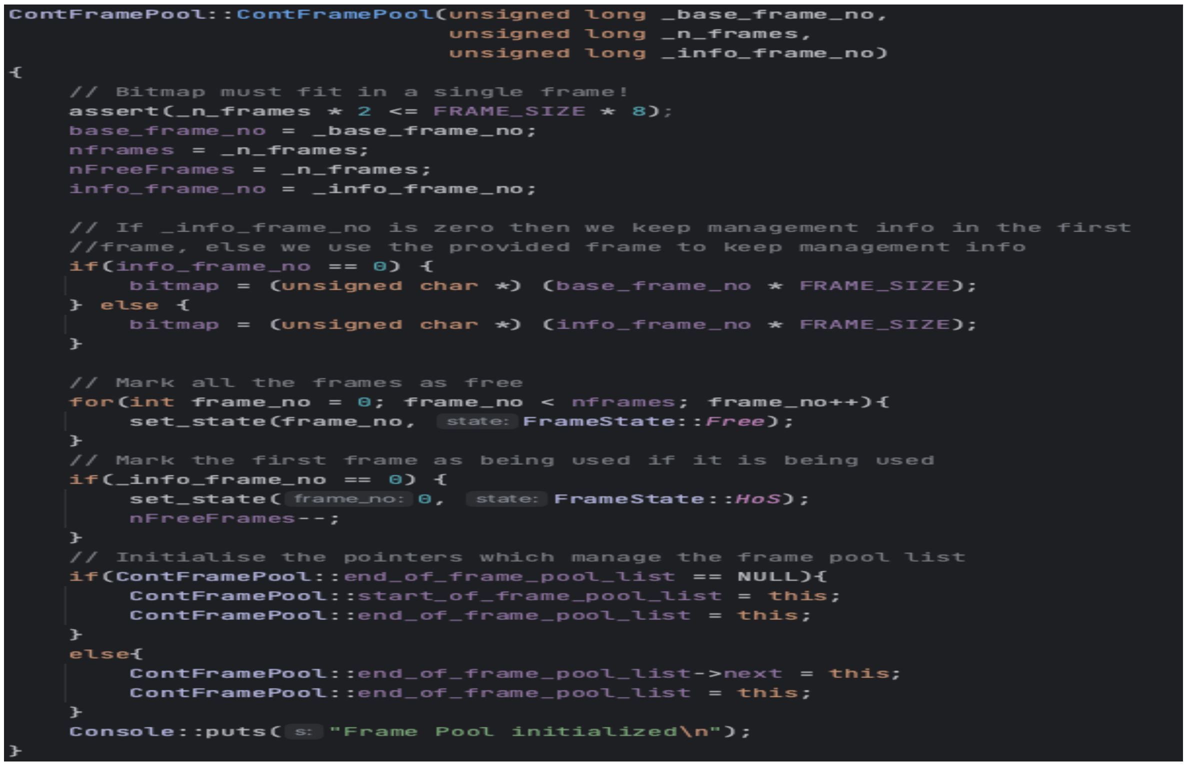
I mainly changed two files cont_frame_pool.C and cont_frame_pool.H for this machine problem. To compile the code, you need to run the make file. To run the simulator, use copykernel.sh script and run the command ”bochs -f bochsrc.bxrc” to start the bochs simulator.

cont_frame_pool.H: Private datamembers : We use the a character bitmap, enum class (to maintain frame state), and static pointers to maintain the framepool. This method is used to allocate the memory.



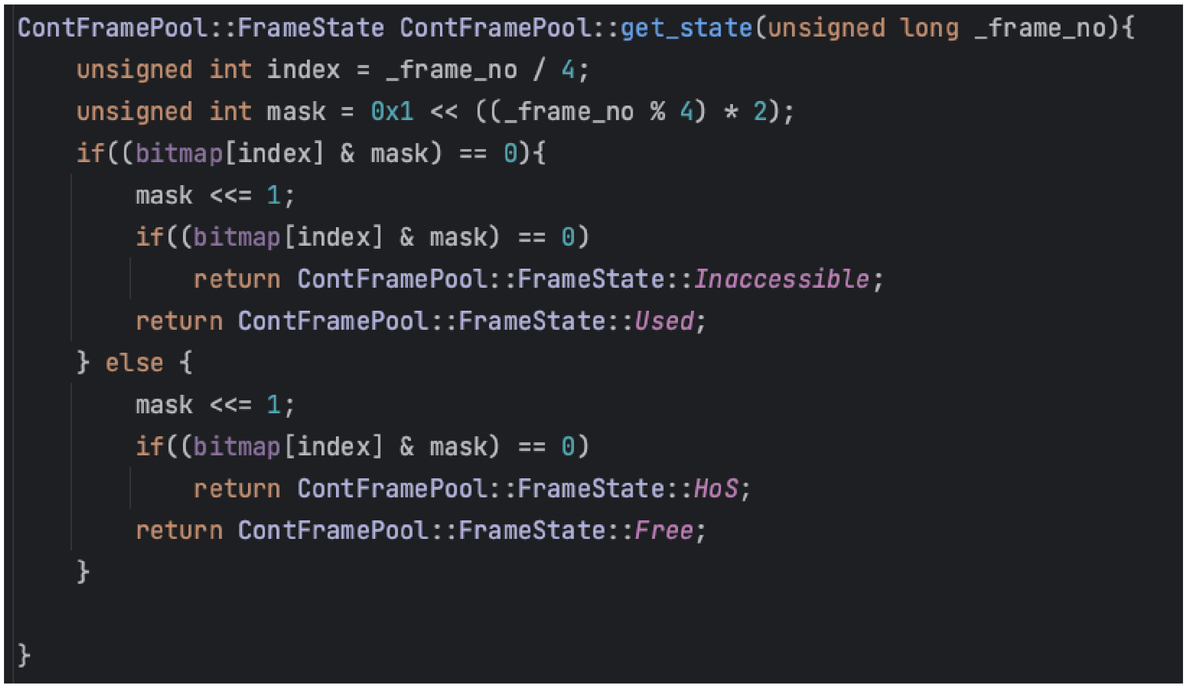
```
class ContFramePool {

private:
    /* -- DEFINE YOUR CONT FRAME POOL DATA STRUCTURE(s) HERE. */
    unsigned char * bitmap;        // We implement the simple frame pool with a bitmap
    unsigned int    nFreeFrames;   // Number of Free Frames in the frame pool
    unsigned long   base_frame_no; // Where does the frame pool start in phys mem?
    unsigned long   nframes;       // Size of the frame pool
    unsigned long   info_frame_no; // Where do we store the management information?


    /* ---- STATE MANAGEMENT */

    enum class FrameState {Free, Used, HoS, Inaccessible};

    // Static Pointers to ContFramePool to maintain a list
    static ContFramePool *start_of_frame_pool_list, *end_of_frame_pool_list;
    ContFramePool* next; // Pointer pointing to the next frame pool in the list
    FrameState get_state(unsigned long _frame_no);
    void set_state(unsigned long _frame_no, FrameState _state);
```

cont_frame_pool.C: ContFramePool Constructor : This constructor initialises the character bitmap, Framepool sequence pointers, and also marks all the frames as ”FREE”.



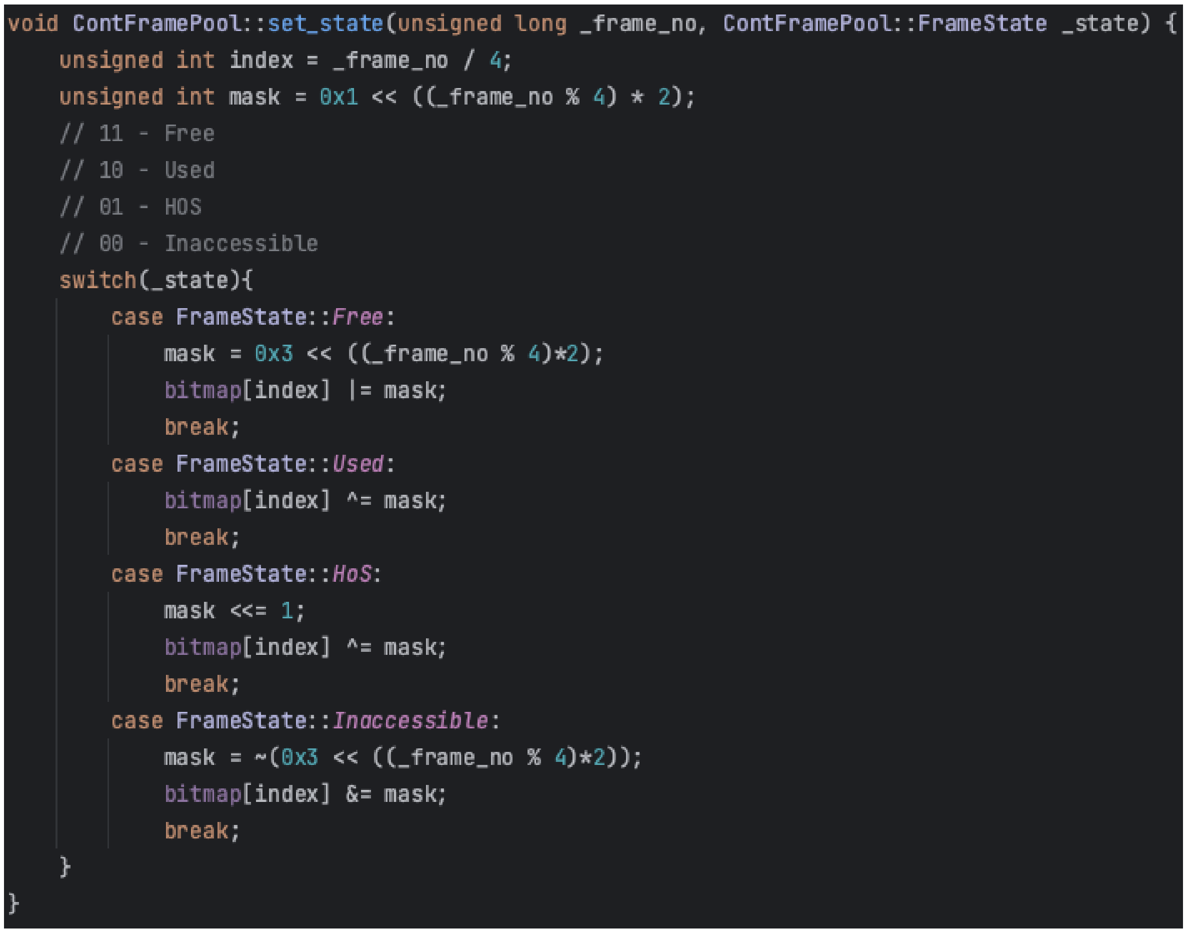
```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    // Bitmap must fit in a single frame!
    assert(_n_frames * 2 <= FRAME_SIZE * 8);
    base_frame_no = _base_frame_no;
    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);
    }

    // Mark all the frames as free
    for(int frame_no = 0; frame_no < nframes; frame_no++){
        set_state(frame_no, state: FrameState::Free);
    }
    // Mark the first frame as being used if it is being used
    if(_info_frame_no == 0) {
        set_state( frame_no: 0, state: FrameState::HoS);
        nFreeFrames--;
    }
    // Initialise the pointers which manage the frame pool list
    if(ContFramePool::end_of_frame_pool_list == NULL){
        ContFramePool::start_of_frame_pool_list = this;
        ContFramePool::end_of_frame_pool_list = this;
    }
    else{
        ContFramePool::end_of_frame_pool_list->next = this;
        ContFramePool::end_of_frame_pool_list = this;
    }
    Console::puts( s: "Frame Pool initialized\n");
}
```

cont_frame_pool.C: get_state : This method is used get the state of frame from the two bits corresponding to the frame. We get the row(index) by dividing the frame number by 4 and the column(mask) by doing modulus 4 operation. Then we do bit operations using the mask to find the state. It returns the FrameState Enum Object. This method is used to allocate the memory.



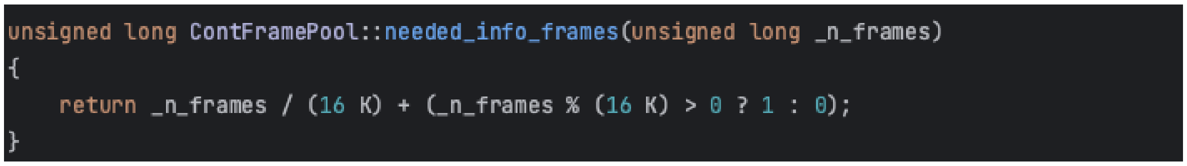
```
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no){
    unsigned int index = _frame_no / 4;
    unsigned int mask = 0x1 << ((_frame_no % 4) * 2);
    if((bitmap[index] & mask) == 0){
        mask <<= 1;
        if((bitmap[index] & mask) == 0)
            return ContFramePool::FrameState::Inaccessible;
        return ContFramePool::FrameState::Used;
    } else {
        mask <<= 1;
        if((bitmap[index] & mask) == 0)
            return ContFramePool::FrameState::HoS;
        return ContFramePool::FrameState::Free;
    }

}
```

cont_frame_pool.C: set_state : This method is used set the state of frame by manipulating the two bits corresponding to the frame. We get the row(index) by dividing the frame number by 4 and the column(mask) by doing modulus 4 operation. Then we do bit operations using the mask to set/clear the each of two bits. This method is used to allocate the memory.



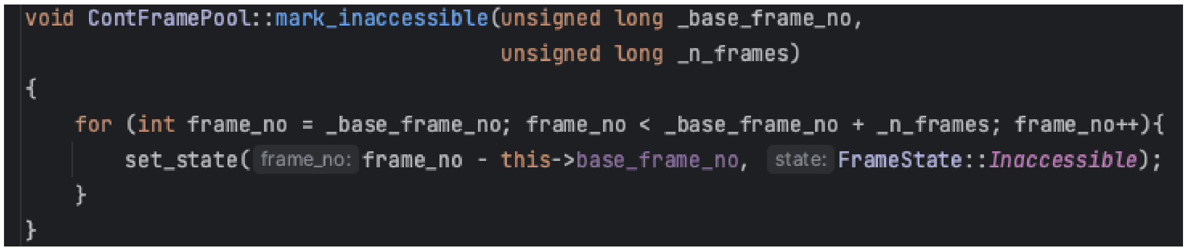
```
void ContFramePool::set_state(unsigned long _frame_no, ContFramePool::FrameState _state) {
    unsigned int index = _frame_no / 4;
    unsigned int mask = 0x1 << ((_frame_no % 4) * 2);
    // 11 - Free
    // 10 - Used
    // 01 - HOS
    // 00 - Inaccessible
    switch(_state){
        case FrameState::Free:
            mask = 0x3 << ((_frame_no % 4)*2);
            bitmap[index] |= mask;
            break;
        case FrameState::Used:
            bitmap[index] ^= mask;
            break;
        case FrameState::HoS:
            mask <<= 1;
            bitmap[index] ^= mask;
            break;
        case FrameState::Inaccessible:
            mask = ~(0x3 << ((_frame_no % 4)*2));
            bitmap[index] &= mask;
            break;
    }
}
```

cont_frame_pool.C: needed_info_frames : This method returns the number of frames needed to store our character bitmap. We would need one frame to manage a frame pool with up to 4 * 4096 = 16k frames, since we use two bits per frame.



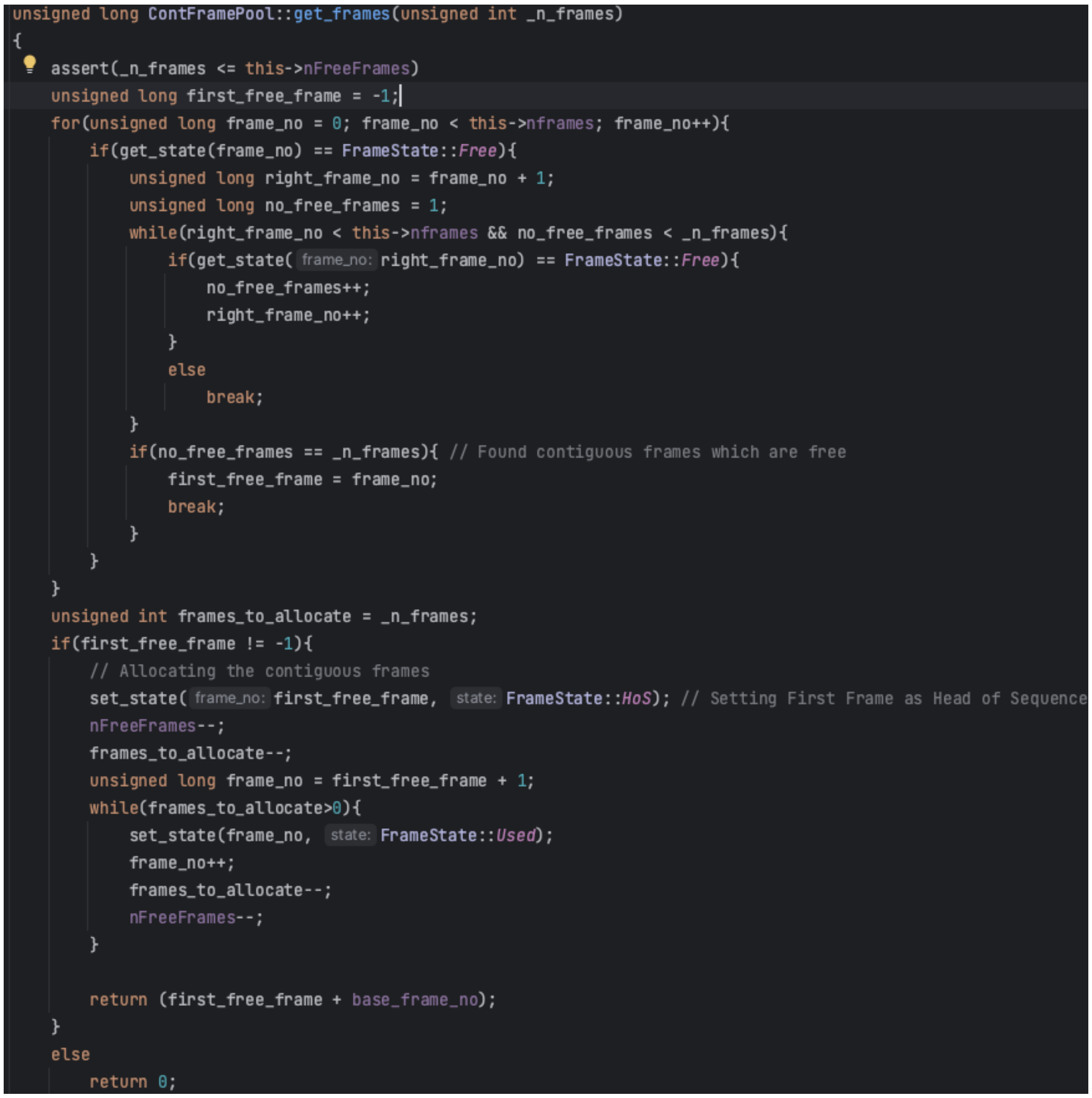
```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    return _n_frames / (16 K) + (_n_frames % (16 K) > 0 ? 1 : 0);
}
```

cont_frame_pool.C: mark_inaccessible : This method sets the state of a certain set of given frames as Inaccessible.



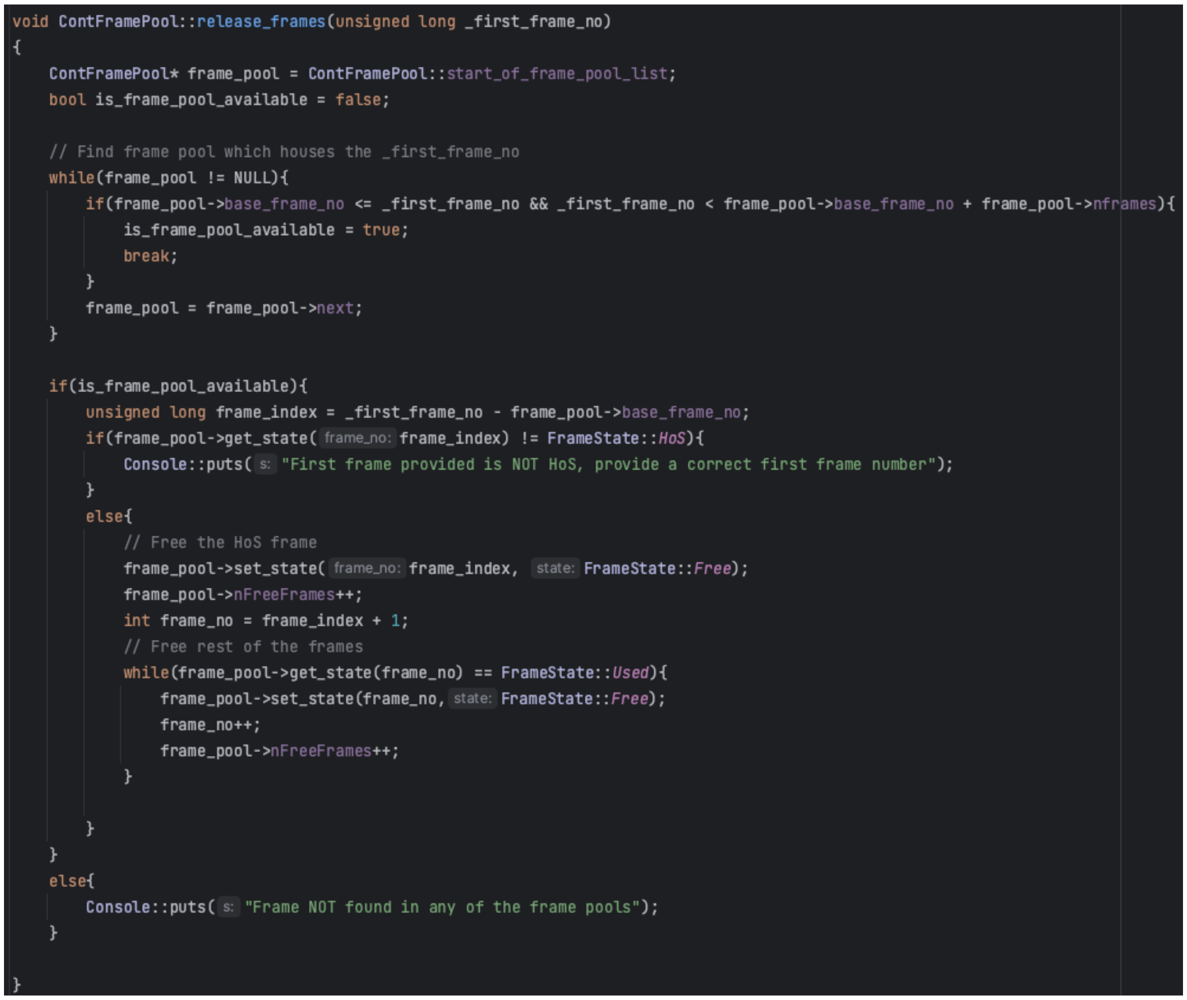
```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{
    for (int frame_no = _base_frame_no; frame_no < _base_frame_no + _n_frames; frame_no++){
        set_state( frame_no: frame_no - this->base_frame_no,  state: FrameState::Inaccessible);
    }
}
```

cont_frame_pool.C: get_frames : Given n as input, this method finds the first contiguous n "Free" frames, marks them as "Used" (the first frame of this set is marked as "HoS") and returns the frame number of HoS back. If it is not able to find n frames that are free, then it returns zero. It just iterates from start of the frame pool and using the get_state() function, it finds the first n free contiguous frames and using set_state() function it sets the frames as "USED".



```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
    assert(_n_frames <= this->nFreeFrames)
    unsigned long first_free_frame = -1;
    for(unsigned long frame_no = 0; frame_no < this->nframes; frame_no++){
        if(get_state(frame_no) == FrameState::Free){
            unsigned long right_frame_no = frame_no + 1;
            unsigned long no_free_frames = 1;
            while(right_frame_no < this->nframes && no_free_frames < _n_frames){
                if(get_state( frame_no: right_frame_no) == FrameState::Free){
                    no_free_frames++;
                    right_frame_no++;
                }
                else
                    break;
            }
            if(no_free_frames == _n_frames){ // Found contiguous frames which are free
                first_free_frame = frame_no;
                break;
            }
        }
    }
    unsigned int frames_to_allocate = _n_frames;
    if(first_free_frame != -1){
        // Allocating the contiguous frames
        set_state( frame_no: first_free_frame,  state: FrameState::HoS); // Setting First Frame as Head of Sequence
        nFreeFrames--;
        frames_to_allocate--;
        unsigned long frame_no = first_free_frame + 1;
        while(frames_to_allocate>0){
            set_state(frame_no,  state: FrameState::Used);
            frame_no++;
            frames_to_allocate--;
            nFreeFrames--;
        }

        return (first_free_frame + base_frame_no);
    }
    else
        return 0;
```

cont_frame_pool.C: release_frames : Given first frame number as input, this method finds the corresponding frame and checks if it is a HoS frame, in case it is not, it returns by putting a error message to console. But if it a HoS frame, it sets it as "Free", then it traverses down the bitmap and marks every "USED" frame as "FREE". It does so untill it encounters a "HoS" or "Inaccessible" frame. This ensures that all the frames of the sequence are released. One more thing this fucntion does before releasing the frames is finding the frame pool to which input frame belongs to. This is done by traversing the list of frame pool using the next pointer of constframepool object. The ConstFramePool has two static object pointers which point to the start and end of the framepool list.

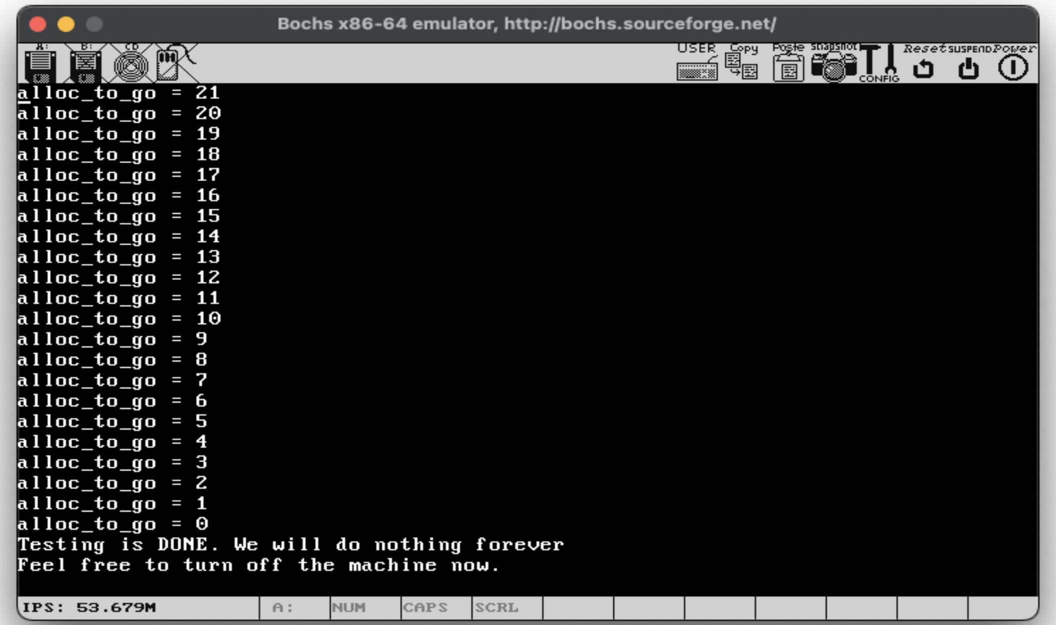


```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    ContFramePool* frame_pool = ContFramePool::start_of_frame_pool_list;
    bool is_frame_pool_available = false;

    // Find frame pool which houses the _first_frame_no
    while(frame_pool != NULL){
        if(frame_pool->base_frame_no <= _first_frame_no && _first_frame_no < frame_pool->base_frame_no + frame_pool->nframes){
            is_frame_pool_available = true;
            break;
        }
        frame_pool = frame_pool->next;
    }

    if(is_frame_pool_available){
        unsigned long frame_index = _first_frame_no - frame_pool->base_frame_no;
        if(frame_pool->get_state( frame_no: frame_index) != FrameState::HoS){
            Console::puts( s: "First frame provided is NOT HoS, provide a correct first frame number");
        }
        else{
            // Free the HoS frame
            frame_pool->set_state( frame_no: frame_index,  state: FrameState::Free);
            frame_pool->nFreeFrames++;
            int frame_no = frame_index + 1;
            // Free rest of the frames
            while(frame_pool->get_state(frame_no) == FrameState::Used){
                frame_pool->set_state(frame_no, state: FrameState::Free);
                frame_no++;
                frame_pool->nFreeFrames++;
            }

        }
    }
    else{
        Console::puts( s: "Frame NOT found in any of the frame pools");
    }

}
```

Testing

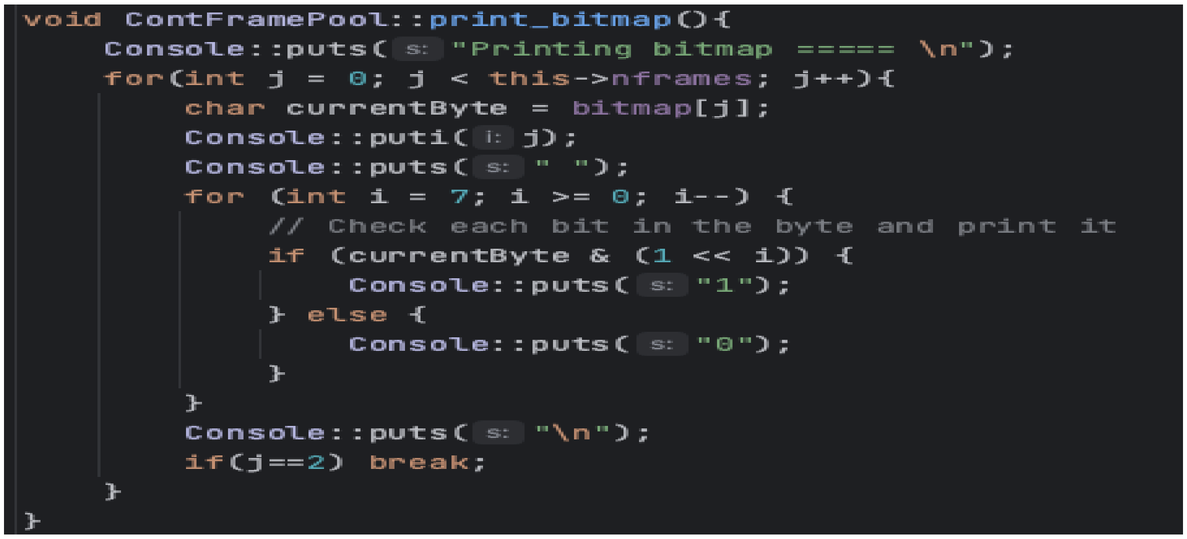
For testing, I used the given test function from kernel.C. After uncommenting the code for Process Pool initialisation, I tested the Process Memory Pool in conjunction with the Kernel Memory Pool. All tests are passing. Other than this, I wrote a function called print_bitmap() to print a first few rows of the character bitmap. Allocated a few frames (group of 2,3,12), released a few sets and again allocated new set of frames. I verified if the frame allocation was being done by printing the bitmap visually.

All the tests given in Kernel.C are passing for both Kernel Mem Pool and Process Mem Pool.



```
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.
```

Print_bitmap() Function :-



```
void ContFramePool::print_bitmap(){
    Console::puts( s: "Printing bitmap ===== \n");
    for(int j = 0; j < this->nframes; j++){
        char currentByte = bitmap[j];
        Console::puti( i: j);
        Console::puts( s: " ");
        for (int i = 7; i >= 0; i--) {
            // Check each bit in the byte and print it
            if (currentByte & (1 << i)) {
                Console::puts( s: "1");
            } else {
                Console::puts( s: "0");
            }
        }
        Console::puts( s: "\n");
        if(j==2) break;
    }
}
```