# MP2: Frame Manager

Cheng-Yun Cheng
UIN: 633002216
CSCE611: Operating System

## Assigned Tasks

**Main:** Completed.

## System Design

The goal of the machine problem is to implement a frame manager. A frame manager could allocate and release frames in its pools. A bitmap is used to track the state of frames. In bitmap, two bits is used to represent the state of one frame.

| State | bits |
|---|---|
| Free | 00 |
| Used | 01 |
| Head-of-Sequence (HoS) | 10 |

## Code Description

I changed cont_frame_pool.h and cont_frame_pool.c to complete the ContFramePool class for this machine problem.

**cont_frame_pool.h: data members in ContFramePool** : The ContFramePool class contains 6 private data members and two public static data members.

- unsigned char *bitmap: Tracking the state of frames
- unsigned int nFreeFrames: The number of free frames in the pool
- unsigned long base_frame_no: The first frame in the pool
- unsigned long n_frames: The number of frames in the pool
- unsigned long info_frame_no: The Frame which is used to store the bitmap
- ContFramePool *next: Pointing to the next pool.
- static ContFramePool *head: Pointing to the first pool.

```
private:
    /* -- DEFINE YOUR CONT FRAME POOL DATA STRUCTURE(s) HERE. */
    unsigned char * bitmap;
    unsigned int    nFreeFrames;
    unsigned long   base_frame_no;
    unsigned long   n_frames;
    unsigned long   info_frame_no;
    ContFramePool * next;
```

```
public:
    // The frame size is the same as the page size, duh...
    static const unsigned int FRAME_SIZE = Machine::PAGE_SIZE;
    static ContFramePool *head;
```

**cont_frame_pool.c: get_state** : Return the state of the given frame. First, one mask is used to check whether the frame is HoS or not. If it is HoS, return the frame state. On the other hand, the other mask is used to check the right bit, and return its state.

```
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no){
    // 00: Free, 01: Used, 10: HoS
    unsigned int bitmap_index = _frame_no / 4;
    unsigned char mask = 0x1 << ((_frame_no % 4) * 2);
    unsigned char mask_head = 0x2 << ((_frame_no % 4) * 2);

    if((bitmap[bitmap_index] & mask_head) != 0){
        return FrameState::HoS;
    } else{
        return ((bitmap[bitmap_index] & mask) == 0) ? FrameState::Free : FrameState::Used;
    }
}
```

**cont_frame_pool.c: set_state** : Set the given frame to the specific state.

- Uesd: 00 $\|$ 01 = 01
- HoS: 00 $\|$ 10 = 10
- Free: 10 & 00 = 00 or 01 & 00 = 00

```
void ContFramePool::set_state(unsigned long _frame_no, FrameState _state){
    unsigned int bitmap_index = _frame_no / 4;
    unsigned char mask;
    switch(_state){
        case FrameState::Used:
            // 00 | 01 = 01
            mask = 0x1 << ((_frame_no % 4) * 2);
            bitmap[bitmap_index] |= mask;
            break;
        case FrameState::Free:
            // HoS: 10 & 00 = 00 or Used: 01 & 00 = 00
            mask = 0x3 << ((_frame_no % 4) * 2);
            bitmap[bitmap_index] &= (~mask);
            break;
        case FrameState::HoS:
            // 00 | 10 = 10
            mask = 0x2 << ((_frame_no % 4) * 2);
            bitmap[bitmap_index] |= mask;
            break;
    }
}
```

**cont_frame_pool.c: ContFramePool constructor** : The constructor is used to assign value to all data members, including setting the bitmap address according to _info_frame_no, initialing all frame as Free, and linking all the pool.

```
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 <= FRAME_SIZE*4);

    base_frame_no = _base_frame_no;
    n_frames = _n_frames;
    info_frame_no = _info_frame_no;
    nFreeFrames = _n_frames;
    next = NULL;

    // set the address of bitmap
    if(info_frame_no == 0){
        bitmap = (unsigned char*) (base_frame_no * FRAME_SIZE);
    } else{
        bitmap = (unsigned char*) (_info_frame_no * FRAME_SIZE);
    }

    // initial all frame as Free
    for(int fno = 0; fno < n_frames; fno++){
        set_state(fno, FrameState::Free);
    }

    // if bitmap is stored in this pool, set frame 0 as HoS
    if(_info_frame_no == 0){
        set_state(0, FrameState::HoS);
        nFreeFrames--;
    }
```

```
    // link all the ContFramePool
    if(head != NULL){
        ContFramePool *temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = this;
    } else{
        head = this;
    }

    Console::puts("Frame Pool Initialized\n");
}
```

**cont_frame_pool.c: get_frame** : Allocate a sequence of contiguous frames from the pool. If there are enough contiguous frames, mark the frames as HoS and Used, and then return the first frame. On the other hand, return 0.

```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
    // There must be enough free frames
    assert(nFreeFrames > _n_frames);

    for(unsigned int frame_no = 0; frame_no < n_frames;){
        if(get_state(frame_no) == FrameState::Free){
            bool succ = true;
            for(int i = 1; i < _n_frames; i++){
                // if one frame is not Free, keep searching from the next frame
                if(get_state(frame_no + i) != FrameState::Free){
                    frame_no = frame_no + i + 1;
                    succ = false;
                    break;
                }
            }
            if(succ){
                // if find contiguous frames, set their state to HoS or Used
                for(int i = 0; i < _n_frames; i++){
                    if(i == 0)
                        set_state(frame_no, FrameState::HoS);
                    else
                        set_state(frame_no + i, FrameState::Used);
                }
                nFreeFrames -= _n_frames;
                return (frame_no + base_frame_no);
            }
        }
        else{
            frame_no++;
        }
    }
    return 0;
}
```

**cont_frame_pool.c: mark_inaccessible** : Mark the sequence of frames as inaccessible.

```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{
    // TODO: IMPLEMENTATION NEEEDED!
    for(int fno = _base_frame_no; fno < _base_frame_no + _n_frames; fno++){
        set_state(fno - base_frame_no, FrameState::HoS);
    }
}
```

**cont_frame_pool.c: release_frames** : Release the contiguous sequence of frames which are allocated. Check which pool contain these frames, and then call the pool's release_frame function. If the pool contains the frame, then the frame number must be in [pool's base frame number, pool's base frame number + pool's number of frames].

```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    // TODO: IMPLEMENTATION NEEEDED!
    // determin which pool contains the frame
    ContFramePool *pool = head;
    while(pool){
        if(_first_frame_no >= pool->base_frame_no &&
           _first_frame_no <= pool->base_frame_no + pool->n_frames){
            pool->release(_first_frame_no);
            break;
        } else{
            pool = pool->next;
        }
    }
}
```

**cont_frame_pool.c: release_frame** : Release the contiguous sequence of frames. First, if the state of the first frame is HoS, set it as Free. And then, traverse the pool to set frames whose state are Used to Free until meeting one frame which is Free or HoS. Also, when one frame is marked as Free, the number of free frames should increase. On the other hand, if the first frame is not HoS, the function would do nothing.

```
void ContFramePool::release_frame(unsigned long _first_frame_no)
{
    int first_no = _first_frame_no - base_frame_no;
    // check the first frame state is HoS
    if(get_state(first_no) == FrameState::HoS){
        set_state(first_no, FrameState::Free);
        nFreeFrames++;
        int i = first_no + 1;
        while(get_state(i) == FrameState::Used){
            set_state(i, FrameState::Free);
            nFreeFrames++;
            i++;
        }
    }
}
```

**cont_frame_pool.c: needed_info_frames** : Return the number of frames needed to manage a frame pool with a given size. Because I used 2 bits to manage a frame, a bitmap whose size is 4KB (one frame) could manage 4*4KB = 16KB frames.

```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    // TODO: IMPLEMENTATION NEEEDED!
    // 2 bits per frame
    // one frame could manege 4KB * 4 = 16KB frames
    return (_n_frames / (16*1024) + (_n_frames % (16*1024) > 0 ? 1 : 0));
}
```

# Testing

The provided test function is used to test the get_frames function. Therefore, I wrote three functions to test mark_inaccessible function, release_frames function and needed_info_frames function.

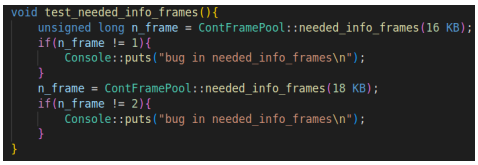
**kernel.c: test_mark_inaccessible** : In this test, I mark all frame as inaccessible. Next, I call get_frames function, and it should return 0.

```
void test_mark_inaccessible(ContFramePool * _pool){
    // set all frame inaccessible
    _pool->mark_inaccessible(PROCESS_POOL_START_FRAME,PROCESS_POOL_SIZE);
    unsigned long frame = _pool->get_frames(10);
    if(frame != 0){
        Console::puts("bug in mark_inaccessible\n");
    }
}
```

**kernel.c: test_release_frames** : In this test, I first allocate some frames, and release them. And then, I allocate the same size again, it should return the same frame number.

```
void test_release_frames(ContFramePool * _pool){
    int n_frames = ((2 MB) / (8 KB));
    unsigned long frame1 = _pool->get_frames(n_frames);
    _pool->release_frames(frame1);
    unsigned long frame2 = _pool->get_frames(n_frames);
    if(frame2 != frame1){
        Console::puts("bug in release_frames\n");
    }
}
```

**kernel.c: test_needed_info_frames** : A bitmap whose size is 4KB (one frame) could manage 4*4KB = 16KB frames. If the input is 16KB, the function should return 1. If the input in 17KB, the function should return 2.



```
void test_needed_info_frames(){
    unsigned long n_frame = ContFramePool::needed_info_frames(16 KB);
    if(n_frame != 1){
        Console::puts("bug in needed_info_frames\n");
    }
    n_frame = ContFramePool::needed_info_frames(18 KB);
    if(n_frame != 2){
        Console::puts("bug in needed_info_frames\n");
    }
}
```