MP2: Simple File System

Nitish Malluru
UIN: 932007196
CSCE 410: Operating System

Assigned Tasks

Main: Completed.

System Design

The goal of machine problem 2 was to create your own frame manager that can allocate and deallocate continuous frames of physical memory. It also keeps track of all the frame pool managers to make it easy to release memory statically given the frame address.

Code Description

I changed cont_frame_pool.h and cont_frame_pool.c for this machine problem. To compile the code use make and make run to execute the code

cont_frame_pool.c : ContFramePool : This function initializes the continuous frame pool object. It first attaches itself to the list of frame pool objects. Then it calculates the number of frames to store the bitmap used to track the state of each frame. Finally it sets the base frame number and sets the info frames as being Used.

```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    // doubly linked list of frame pools
    if (head_frame_pool != nullptr) {
        head_frame_pool->prev_frame_pool = this;
        this->next_frame_pool = head_frame_pool;
    } else {
        this->next_frame_pool = nullptr;
    }

    head_frame_pool = this;
    this->prev_frame_pool = nullptr;

    base_frame_no = _base_frame_no;
    nframes = _n_frames;

    unsigned long n_info_frames = needed_info_frames(_n_frames);

    if (_info_frame_no == 0){
        info_frame_no = _base_frame_no;
        _base_frame_no += n_info_frames;
        nframes -= n_info_frames;
    } else {
        info_frame_no = _info_frame_no;
    }

    unsigned long bmap_size = (nframes * 2 + 7) / 8;
    bitmap = (unsigned char *) (info_frame_no * FRAME_SIZE);
    for(unsigned long i = 0; i < bmap_size; i++){
        bitmap[i] = 0;
    }

    // set info frames as used
    set_state(this, _info_frame_no, FrameState::HOS);
    for (unsigned long i = _info_frame_no + 1; i < n_info_frames; i++) {
        set_state(this, i, FrameState::Used);
    }
}
```

cont_frame_pool.c : get_frames : This function allocates a contiguous set of frames. I first start by checking if there exists n continuous frames within the bitmap that are all free. If I cannot find a continuous set of n frames I will return 0, otherwise I set the first value to Head of Sequence and the remaining values in the set to Used.

```
unsigned long ContFramePool::get_frames(unsigned int _n_frames) {
    unsigned long bmap_size = (nframes * 2 + 7) / 8;
    unsigned long start = 0;

    while (start < nframes) {
        unsigned long curr;
        for (curr = start; curr < start + _n_frames && curr < nframes; curr++) {
            if (get_state(this, curr) != FrameState::Free) {
                break;
            }
        }

        if (curr == start + _n_frames) {
            set_state(this, start, FrameState::HOS);
            for (unsigned long i = start + 1; i < start + _n_frames; i++) {
                set_state(this, i, FrameState::Used);
            }
            return start + base_frame_no;
        }

        start = curr + 1;
    }

    return 0;
}
```

cont_frame_pool.c : mark_inaccessible : This function sets a continuous set of frames as inaccessible. We first check to see if the frame address is contained in the frame pool. If so we then start marking from that index forward as Used.

```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{
    if (_base_frame_no < base_frame_no || _n_frames > nframes) {
        return;
    }

    set_state(this, _base_frame_no, FrameState::HOS);

    for (unsigned long i = _base_frame_no + 1; i < _base_frame_no + _n_frames; i++) {
        set_state(this, i, FrameState::Used);
    }
}
```

cont_frame_pool.c : release_frames : This function releases all the frames it has seen starting from the given frame address until it runs into a head of sequence of free value in the bitmap. Since this method is static we must first check our linked list of frame pools to determine if the address is contained in the current pool, if so we can release the frames.

```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    for (ContFramePool * curr = head_frame_pool; curr != nullptr; curr = curr->next_frame_pool) {
        unsigned long frame_index = _first_frame_no - curr->base_frame_no;
        if (frame_index >= curr->nframes) {
            continue;
        }

        if (get_state(curr, frame_index) != FrameState::HOS) {
            continue;
        }

        set_state(curr, frame_index, FrameState::Free);
        frame_index++;

        while (frame_index < curr->nframes) {
            FrameState state = get_state(curr, frame_index);
            if (state == FrameState::Free || state == FrameState::HOS) {
                break;
            }
            set_state(curr, frame_index, FrameState::Free);
            frame_index++;
        }
        return;
    }
}
```

cont_frame_pool.c : needed_info_frames : This function calculates the number of frames needed to store a bitmap of the entire frame pool. It calculates this by multiplying the number of bits by the number of frames and dividing by the frame size.

```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    return (_n_frames * 2 + FRAME_SIZE - 1) / FRAME_SIZE;
}
```

Testing

In addition to the testing already part of the kernel.c file, I also tested allocating more frames than available, fragmentation handling, and just stress testing by alternation allocations and releases. I think my coverage is around 70%. I could test concurrent access, or extreme overflow attempts