MP2 (Frame Manager) Design Document

Raafay Hemani
934000608
CSCE410: Operating Systems

Assigned Tasks

Main: Completed

System Design

I designed a frame manager for an operating system, using contiguous frame pools to manage contiguous frames in physical memory. I followed the instructor's template source code to inform my system design decisions

Code Description

I modified cont_frame_pool.C and cont_frame_pool.H. To compile the code, run the make command to run the contents of the makefile.

cont_frame_pool.h: private data structures: Each char is a byte and each frame needs 2 bits to represent its state, as such each char represents 4 frames. nFreeFrames is the number of frames with the free state in the pool. The base_frame_no is the absolute location of the start of the pool. Nframes is the number of frames in the pool. info_frame_no gives us the location of the information frame for this pool which may be external depending on how the constructor was invoked.

```
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;   //
    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?
```

cont_frame_pool.h: static pool management: A static array of pool objects used to locate a the pool to which a frame belongs when calling the release method.

```
static ContFramePool * frame_pools[100];
static unsigned int n_frame_pools;
```

cont_frame_pool.C: static member initialization: Initialize the pool management array and size

```
// initialize static members
ContFramePool *ContFramePool::frame_pools[100];
unsigned int ContFramePool::n_frame_pools = 0;
```

cont_frame_pool.C: get and set state: Bit manipulation to change the state of a frame between HoS, free, and used.

```
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no)
{
    unsigned int bitmap_index = _frame_no / 4;
    unsigned int shift = (_frame_no % 4) * 2;
    unsigned char mask = 0x3 << shift;

    unsigned char state = (bitmap[bitmap_index] & mask) >> shift;

    switch (state)
    {
    case 0:
        return FrameState::Free;
    case 1:
        return FrameState::Used;
    case 2:
        return FrameState::HoS;
    default:
        return FrameState::Free;
    }
}
```

```
void ContFramePool::set_state(unsigned long _frame_no, FrameState _state)
{
    unsigned int bitmap_index = _frame_no / 4;
    unsigned int shift = (_frame_no % 4) * 2;
    unsigned char mask = 0x3 << shift;

    bitmap[bitmap_index] &= ~mask;

    switch (_state)
    {
    case FrameState::Free:
        bitmap[bitmap_index] |= (0x0 << shift);
        break;
    case FrameState::Used:
        bitmap[bitmap_index] |= (0x1 << shift);
        break;
    case FrameState::HoS:
        bitmap[bitmap_index] |= (0x2 << shift);
        break;
    }
}
```

cont_frame_pool.C: constructor: Set info frame and set all other frames to free

```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    assert(_n_frames <= FRAME_SIZE * 4);

    base_frame_no = _base_frame_no;
    nframes = _n_frames;
    nFreeFrames = _n_frames;
    info_frame_no = _info_frame_no;

    if (info_frame_no == 0)
    {
        bitmap = (unsigned char *)(base_frame_no * FRAME_SIZE);
    }
    else
    {
        bitmap = (unsigned char *)(info_frame_no * FRAME_SIZE);
    }

    // mark all frames as free except for the info frame if it is not external
    for (int fno = 0; fno < _n_frames; fno++)
    {
        set_state(fno, FrameState::Free);
    }

    if (_info_frame_no == 0)
    {
        set_state(0, FrameState::Used);
        nFreeFrames--;
    }

    // add this frame pool to the list of frame pools
    frame_pools[n_frame_pools] = this;
    n_frame_pools++;
}
```

cont_frame_pool.C: get_frames: Set a contiguous set of _n_frames frames in the pool as being used with the first being HoS

```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
    int hos_candidate = 0;
    int free_frames = 0;
    for (int fno = 0; fno < nframes; fno++)
    {
        if (get_state(fno) == FrameState::Free)
        {
            free_frames++;
            if (free_frames == _n_frames)
            {
                set_state(hos_candidate, FrameState::HoS);
                for (int i = hos_candidate + 1; i < hos_candidate + _n_frames; i++)
                {
                    set_state(i, FrameState::Used);
                }
                nFreeFrames -= _n_frames;
                return base_frame_no + hos_candidate;
            }
        }
        else
        {
            free_frames = 0;
            hos_candidate = fno + 1;
        }
    }
    return 0;
}
```

cont_frame_pool.C: mark_inaccessible: Mark _n_frames as used and the first as HoS, starting from _base_frame_no

```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{
    set_state(_base_frame_no - base_frame_no, FrameState::HoS);
    for (int i = _base_frame_no - base_frame_no + 1; i < _base_frame_no - base_frame_no + _n_frames; i++)
    {
        set_state(i, FrameState::Used);
    }
}
```

cont_frame_pool.C: release_frames: Given a HoS frame index, find the pool it belongs to and mark that segment as free.

```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    // determine which frame pool this frame belongs to
    ContFramePool *pool = nullptr;
    for (int i = 0; i < n_frame_pools; i++)
    {
        if (frame_pools[i]->base_frame_no <= _first_frame_no && _first_frame_no < frame_pools[i]->base_frame_no + frame_pools[i]->nframes)
        {
            pool = frame_pools[i];
            break;
        }
    }
    if (pool == nullptr)
    {
        // no pool found
        return;
    }
    unsigned long frame_ind = _first_frame_no - pool->base_frame_no;
    assert(pool->get_state(frame_ind) == FrameState::HoS); // the first frame must be the head of sequence
    pool->set_state(frame_ind, FrameState::Free);
    frame_ind++;
    while (frame_ind < pool->nframes && pool->get_state(frame_ind) == FrameState::Used)
    {
        pool->set_state(frame_ind, FrameState::Free);
        frame_ind++;
    }
}
```

cont_frame_pool.C: needed_info_frames: Find the number of frames needed to manage n other frames given frame size and knowing how many bits are needed to represent a frame.

```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    // my bitmap uses 2 bits per frame, so each info frame can manage FRAME_SIZE * 4 frames
    return _n_frames / (FRAME_SIZE * 4) + (_n_frames % (FRAME_SIZE * 4) > 0 ? 1 : 0);
}
```

Testing

The kernel runs