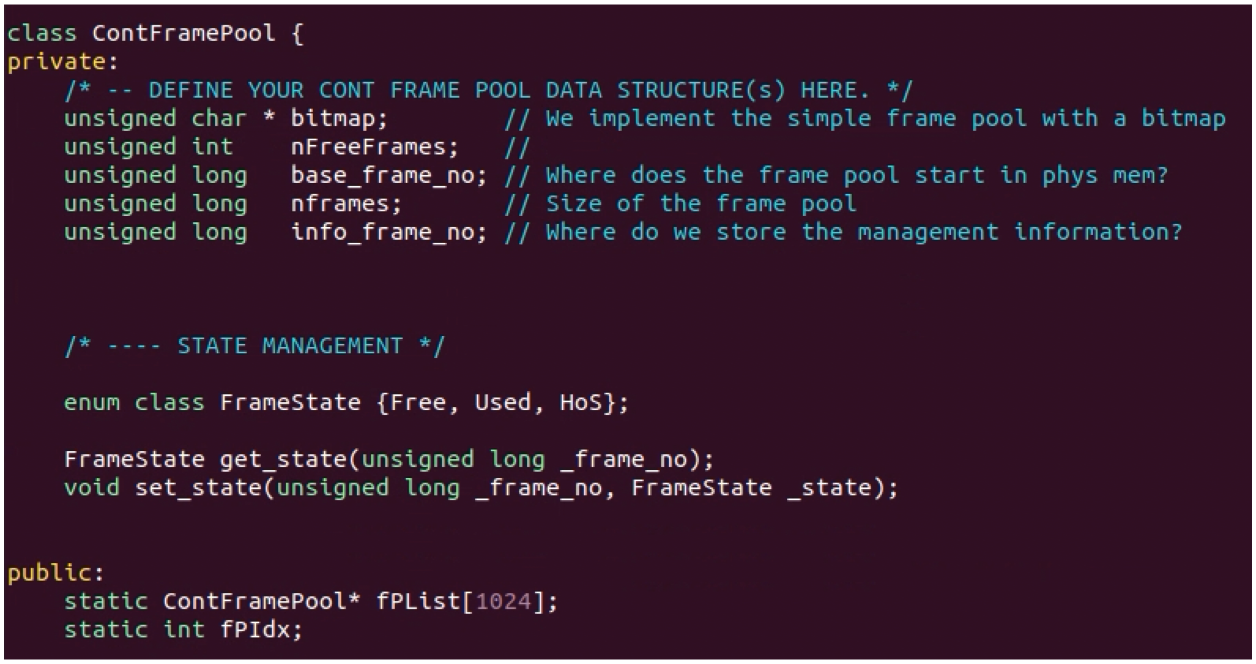
Kwangkyu Hwang

MP2 Implementation document

Editted file:

cont_frame_pool.H
conf_frame_pool.C

cont_frame_pool.H


```
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;   //
    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};

    FrameState get_state(unsigned long _frame_no);
    void set_state(unsigned long _frame_no, FrameState _state);


public:
    static ContFramePool* fPList[1024];
    static int fPIdx;
```
Edited part

unsigned char * bitmap;
unsigned int   nFreeFrames;
unsigned long   base_frame_no;
unsigned long   nframes;
unsigned long   info_frame_no;

variables above are same name and same role with simple_frame_pool

static ContFramePool* fPList[1024];

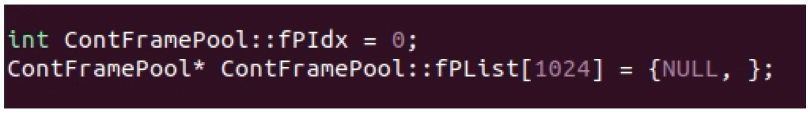
data structure for saving multi frame pool

static int fPIdx;

index of fPList where next frame pool will be located

cont_frame_pool.C

added part

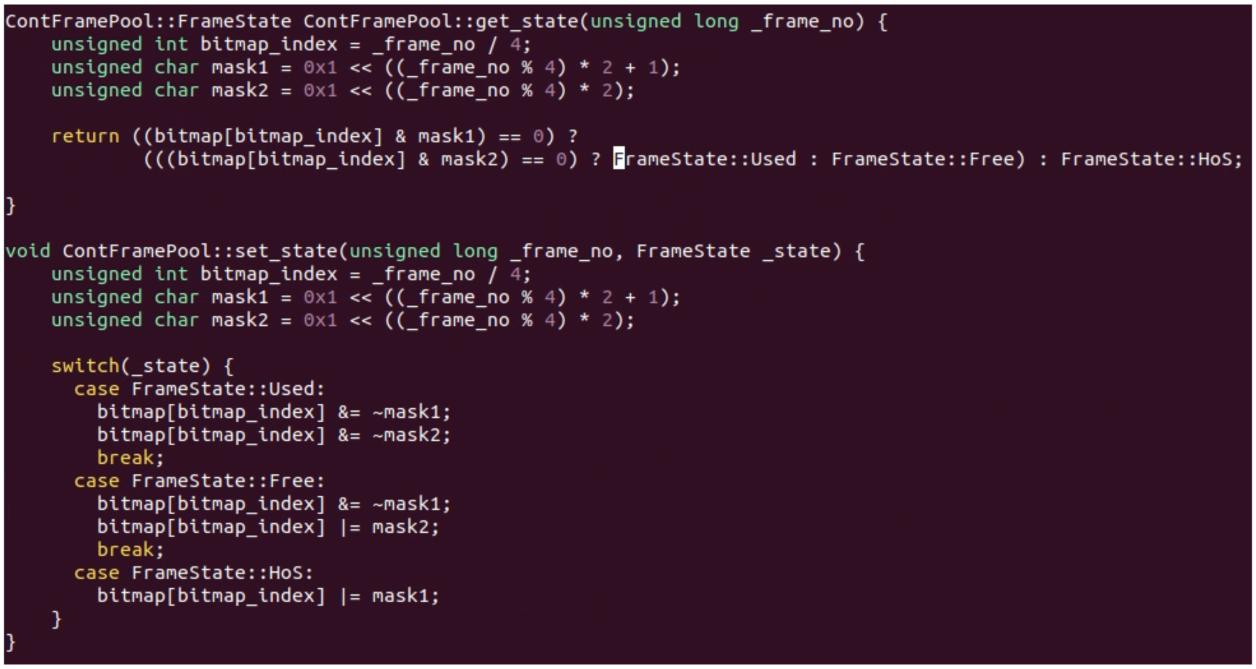


```
int ContFramePool::fPIdx = 0;
ContFramePool* ContFramePool::fPList[1024] = {NULL, };
```

Initialization of static variables for multi frame pools

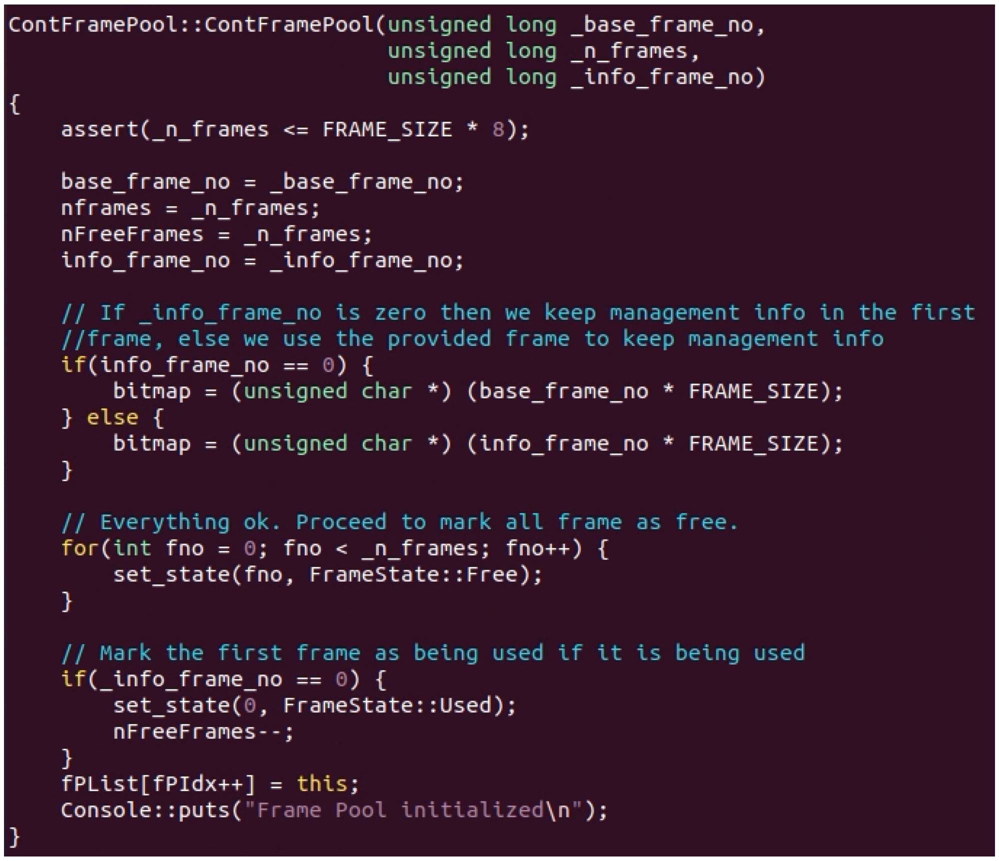
Implemented functions

- get_state
- set_state
- ContFramePool constructor
- get_frames
- release_frames
- mark_inaccessible
- needed_info_frames



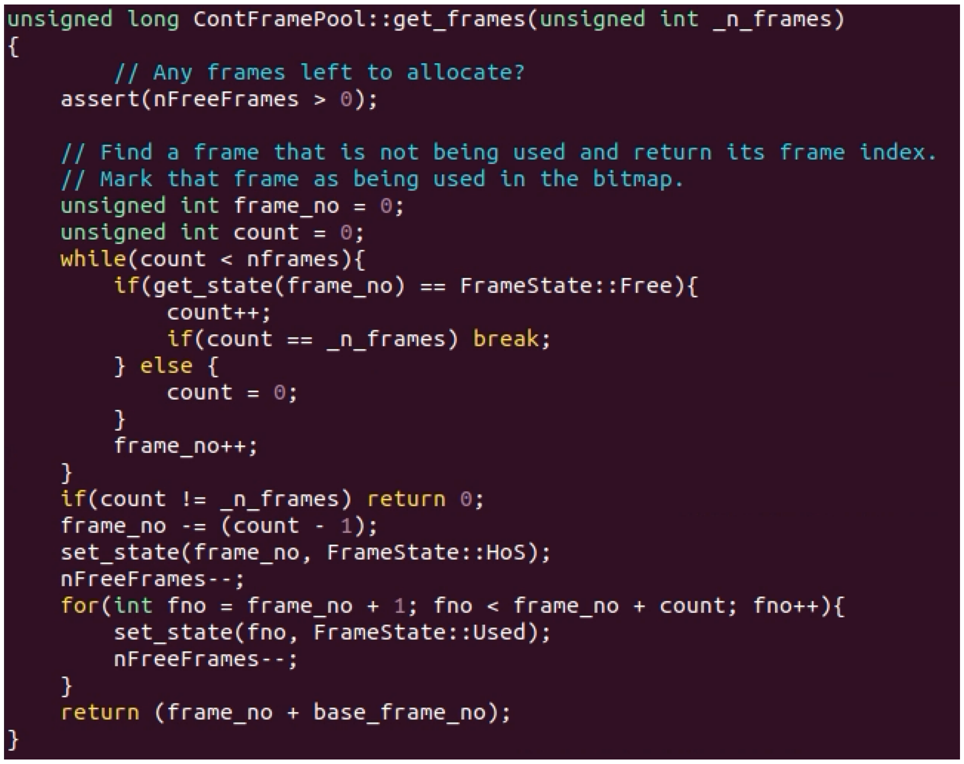
```
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no) {
    unsigned int bitmap_index = _frame_no / 4;
    unsigned char mask1 = 0x1 << ((_frame_no % 4) * 2 + 1);
    unsigned char mask2 = 0x1 << ((_frame_no % 4) * 2);

    return ((bitmap[bitmap_index] & mask1) == 0) ?
            (((bitmap[bitmap_index] & mask2) == 0) ? FrameState::Used : FrameState::Free) : FrameState::HoS;

}

void ContFramePool::set_state(unsigned long _frame_no, FrameState _state) {
    unsigned int bitmap_index = _frame_no / 4;
    unsigned char mask1 = 0x1 << ((_frame_no % 4) * 2 + 1);
    unsigned char mask2 = 0x1 << ((_frame_no % 4) * 2);

    switch(_state) {
      case FrameState::Used:
        bitmap[bitmap_index] &= ~mask1;
        bitmap[bitmap_index] &= ~mask2;
        break;
      case FrameState::Free:
        bitmap[bitmap_index] &= ~mask1;
        bitmap[bitmap_index] |= mask2;
        break;
      case FrameState::HoS:
        bitmap[bitmap_index] |= mask1;
    }
}
```

ContFramePool::set_state, ContFramePool::get_state:
Unlike simple_frame_pool, cont_frame_pool use 2 bits to store state of a frame
FrameState::Used : 00
FrameState::Free: 01
FrameState::HoS: 1X
Thus, we need to divide one unsigned char (which is 8 bit) to 4 partitioin (01), (23), (45), (67) and apply bit operation to store and get information of frames.



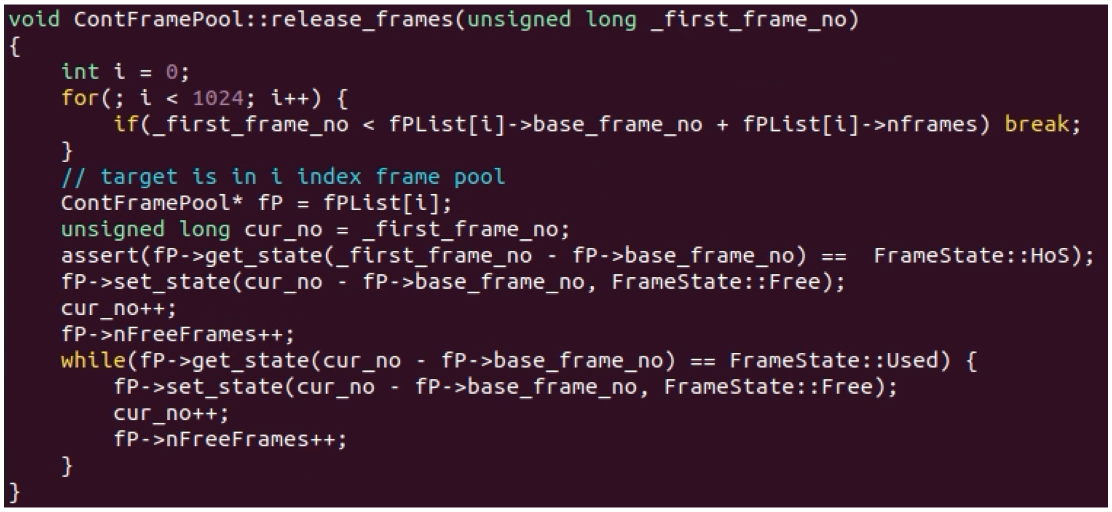
```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    assert(_n_frames <= FRAME_SIZE * 8);

    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);
    }

    // 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::Used);
        nFreeFrames--;
    }
    fPList[fPIdx++] = this;
    Console::puts("Frame Pool initialized\n");
}
```

ContFramePool constructor:
Do similar things to SimpleFramePool, but store itself to static variable fPList and increase fPIdx to store next framePool



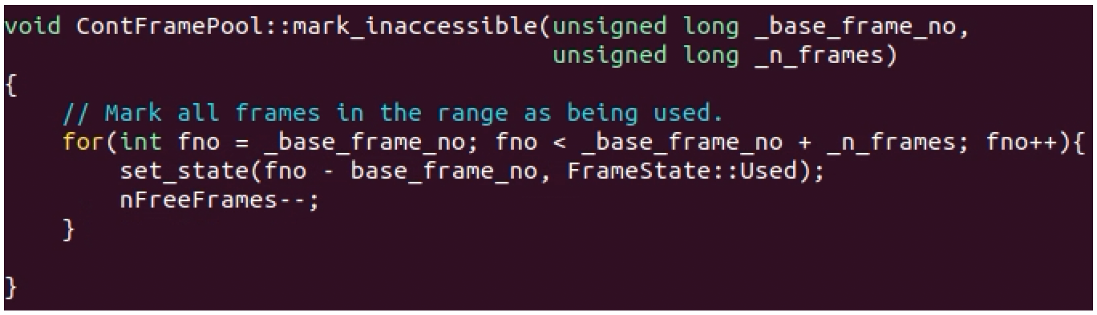
```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
        // Any frames left to allocate?
    assert(nFreeFrames > 0);

    // Find a frame that is not being used and return its frame index.
    // Mark that frame as being used in the bitmap.
    unsigned int frame_no = 0;
    unsigned int count = 0;
    while(count < nframes){
        if(get_state(frame_no) == FrameState::Free){
            count++;
            if(count == _n_frames) break;
        } else {
            count = 0;
        }
        frame_no++;
    }
    if(count != _n_frames) return 0;
    frame_no -= (count - 1);
    set_state(frame_no, FrameState::HoS);
    nFreeFrames--;
    for(int fno = frame_no + 1; fno < frame_no + count; fno++){
        set_state(fno, FrameState::Used);
        nFreeFrames--;
    }
    return (frame_no + base_frame_no);
}
```

get_frames:
Find start frame number that allocate _n_frames continuous frames. Thus, we need to find continuous Free state frame. Continuously increasing frame_no and find until it arrive to the end of the frame pool.
If continuous Free frames are less then required frames, return 0;
Else, since the function has to return the start position frame, find start position by decrease count -1 amount.
Set HoS to start position frame, and fill other frames' position to Used state.
Then return the absolute frame position by adding base_frame_no of frame pool.



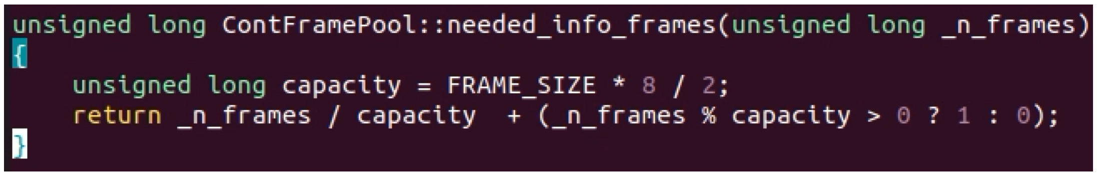
```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    int i = 0;
    for(; i < 1024; i++) {
        if(_first_frame_no < fPList[i]->base_frame_no + fPList[i]->nframes) break;
    }
    // target is in i index frame pool
    ContFramePool* fP = fPList[i];
    unsigned long cur_no = _first_frame_no;
    assert(fP->get_state(_first_frame_no - fP->base_frame_no) ==  FrameState::HoS);
    fP->set_state(cur_no - fP->base_frame_no, FrameState::Free);
    cur_no++;
    fP->nFreeFrames++;
    while(fP->get_state(cur_no - fP->base_frame_no) == FrameState::Used) {
        fP->set_state(cur_no - fP->base_frame_no, FrameState::Free);
        cur_no++;
        fP->nFreeFrames++;
    }
}
```

release_frames:
Since we only have frame_no, we need to find which frame pool this frame_no exist.
Each frame pool contains frames from ‘base_frame_no’ to ‘base_frame_no + nframes -1’
Thus, we can find where does _first_frame_no belongs.
After find proper frame pool, check target frame is HoS state. And then change state of first frame to Free and change state of next frames continuously. If it arrives to the HoS or Free state, stop.



```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{
    // Mark all frames in the range as being used.
    for(int fno = _base_frame_no; fno < _base_frame_no + _n_frames; fno++){
        set_state(fno - base_frame_no, FrameState::Used);
        nFreeFrames--;
    }

}
```

mark_inaccessible:
mark Used state _n_frames amount from _base_frame_no.



```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    unsigned long capacity = FRAME_SIZE * 8 / 2;
    return _n_frames / capacity  + (_n_frames % capacity > 0 ? 1 : 0);
}
```

needed_info_frames:
Since 1 frame has Frame_size(byte) * 8 bit, it can store (Frame_size * 8 / 2) frames’ state. This is because 1 frame’s state is stored in 2 bits. So, we can calculate how many frames needed to indicate _n_frames frames.