MP2: Frame Manager

Sushant Vijay Shelar
UIN: 733001479
CSCE611: Operating System

Assigned Tasks

Main: Completed.

System Design

In this machine problem, we will have access to 32MB of memory for allocating frames to both the kernel and processes. Each frame will be sized in 4KB. The initial 2MB of memory is designated for exclusive kernel usage. The subsequent 2MB can be employed as a kernel pool, facilitating the allocation of contiguous memory blocks for kernel operations. The remaining 28MB will serve as the process pool, offering continuous memory allocation for user processes. It's important to note that the memory range between 15MB and 16MB is reserved and unavailable for general use.

Within the scope of this machine problem, I made modifications to two files: cont_frame_pool.C and cont_frame_pool.H. The central objective of this endeavor is to proficiently manage frames and allocate them as necessary. Each frame can assume one of the following distinct states:

- **Free**: In this state, a frame is unoccupied and can be assigned upon request.
- **Used**: Frames in this state have already been allocated and are thus ineligible for further assignment.
- **HoS (Head of Sequence)**: When a frame is in this state, it signifies that it has been assigned and serves as the initial frame allocated for a memory request.
- **Unavailable**: To denote that assigning this frame is not permitted, we have extended the FrameState enum to include this state.

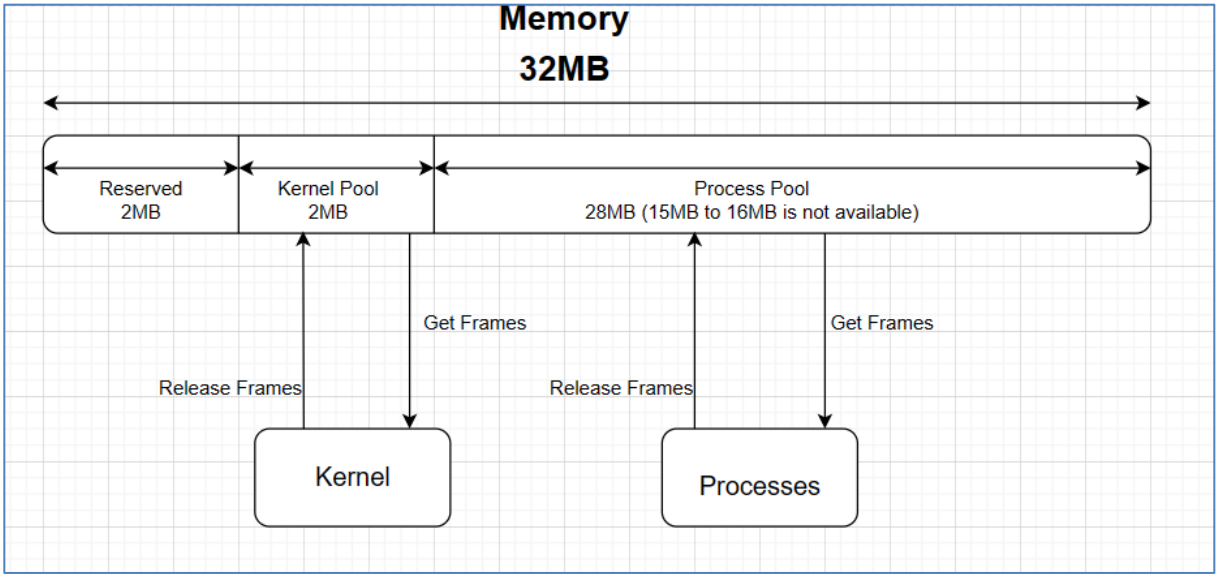

Figure 1: Frame Manager

Code Description

Constructor:

- The base frame number is updated.
- The total number of available frames is updated.
- The count of free frames is updated.
- Initially, all frames are set as free.
- The list of available frame pools is also initialized.

```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    // TODO: IMPLEMENTATION NEEEDED!
    Console::puts("ContframePool::Constructor implemented!\n");

        baseFrameNumber=_base_frame_no;
        frames=_n_frames;
        numberOfFreeFrames=_n_frames;
        infoFrameNo=_info_frame_no;

    //assert(false);

        if(infoFrameNo ==0)
        {
                bitMap = (unsigned char *) (baseFrameNumber * FRAME_SIZE);
        }
        else
        {

                bitMap = (unsigned char *) (infoFrameNo * FRAME_SIZE);
        }

        for (int i=0;i<numberOfFreeFrames;i++)
        {
```

get_frames:

- It identifies a continuous block of frames matching the requested quantity, if available.
- Within this identified block, the first frame is designated as the "Head of Sequence" (HoS), while the remaining frames are marked as "Used."
- Additionally, the count of free frames is appropriately reduced to reflect the allocation.

```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
    unsigned long strt=0;
    unsigned int freeBlock=0;
    // TODO: IMPLEMENTATION NEEEDED!
    Console::puts("ContframePool::get_frames implemented!\n");
        //assert(false);
    bool flag1;

    for(unsigned long i=0; i<frames;i++)
    {
        //      Console::puts("free cont block");
        //  Console::puti(strt);
        //      Console::puts("\n");
            if(get_state(i)!=FrameState::Free)
            {
                    freeBlock=0;
                       flag1=false;
            }
            else
            {
                    freeBlock++;
                    if(freeBlock==1)
```

mark_inaccessible:

- This function serves the purpose of marking all frames within the specified range as "Unavailable."
- In doing so, it effectively removes these frames from the count of free frames.

```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{

    // TODO: IMPLEMENTATION NEEEDED!
    Console::puts("ContframePool::mark_inaccessible implemented!\n");
   // assert(false);
        unsigned long idx=_base_frame_no;
        for(unsigned long i=0;i<_n_frames;i++)
        {
                //Console::puts("marking memory as Unavailable for use");
                set_state(i+_base_frame_no,FrameState::Unavailable);
        }

        numberOfFreeFrames = numberOfFreeFrames - _n_frames;
        //      Console::puts("updated free frames");
        //  Console::puti(numberOfFreeFrames);
        //      Console::puts("\n");
}
```

release_frames:

- In this function, we begin by searching for the frame pool associated with the given first frame number.
- If the frame pool is found, it's crucial to note that the first frame number within this pool must be the "Head of Sequence" (HoS).
- Subsequently, this HoS frame is marked as free.
- Additionally, all contiguous frames within this pool that were previously in the "Used" state are also reset as free.
- As a result of these actions, all released frames are effectively added back to the pool of available free frames, increasing the count accordingly.

```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    // TODO: IMPLEMENTATION NEEEDED!
   Console::puts("ContframePool::release_frames  implemented!\n");
    //assert(false);
        bool flag2;
    unsigned long indx;
    ContFramePool* frame_pool = ContFramePool::start;

    while(frame_pool!=NULL)
    {
            if(_first_frame_no>= frame_pool->baseFrameNumber && _first_frame_no < frame_pool->base
            {
                indx= _first_frame_no - frame_pool->baseFrameNumber;
                flag2=true;
                break;
            }
                    else
                    {
                            flag2=false;
                    }

            frame_pool=frame_pool->next;
    }
```

needed_info_frames:

- Each frame consists of 40968 bits, we can conclude that a single frame can accommodate the state information for 40968/2 = 16K.
- To determine the required number of information frames, we divide the total number of frames to be stored by 16K. This calculation yields the quantity of information frames necessary to store the frame states effectively.

```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
        unsigned long num=0;
    // TODO: IMPLEMENTATION NEEEDED!
    Console::puts("ContframePool::need_info_frames implemented!\n");
    //assert(false);
        if( (_n_frames % (16 K))!=0)
                num++;
        num = num + _n_frames/ (16 K);
        return num;
}
```

get_state:

- The function uses a bitmap concept to access the state of a frame.
- In this system, the states of four frames are grouped and stored together in a single entry within a bitmap array.
- To extract the state of a specific frame, the function employs a mask and carries out bit operations on the bitmap.
- This approach enables efficient and streamlined access to frame states, allowing the determination of whether a frame is free, in use, the head of a sequence, or in an unavailable state.

```
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no)
{
    // Calculate the index within the bitmap array
    unsigned int m = _frame_no / 4;

    // Calculate the bit shift to isolate the frame state within the byte
    unsigned int shift = (_frame_no % 4) * 2;

    // Create a mask to extract the frame state
    unsigned char state_mask = 0x3 << shift;

    // Initialize the frame state to Free
    FrameState state = FrameState::Free;

    // Retrieve the frame state from the bitmap
    unsigned char frame_state = (bitMap[m] & state_mask) >> shift;

    // Determine and set the corresponding FrameState
    switch (frame_state)
    {
        case USED:
            state = FrameState::Used;
            break;
```

set_state:

- It shares a similar implementation structure with get_state.
- Its primary purpose is to update the state of a single frame by replacing the existing state with a new one.
- When handling consecutive frames, these functions are invoked iteratively, addressing one frame at a time to set or retrieve their respective states.

```
void ContFramePool::set_state(unsigned long frame_no, FrameState frameState)
{
    // Calculate the index and shift values
    unsigned long frameNum = frame_no;
    unsigned int j = frameNum / 4;
    unsigned int shift = (frameNum % 4) * 2;

    // Calculate the frame state value based on frameState
    unsigned char frame_state = 0;
    switch (frameState)
    {
        case FrameState::Used:
            frame_state = (unsigned char)USED << shift;
            break;
        case FrameState::Free:
            frame_state = (unsigned char)FREE << shift;
            break;
        case FrameState::HoS:
            frame_state = (unsigned char)HOS << shift;
            break;
```

Testing

I used the 'make' command to compile all the files in the "mp2" folder. Then, I ran a script called 'copy kernel' to copy the bin file to the disk. After that, I executed the code using the "bochs -f bochsrc.bxrc" command.

All the test cases worked as expected, and I've attached a screenshot for reference: