MP2: Frame Manager
Chonglin Zhang
UIN: 833003072
CSCE611: Operating System

Assigned Tasks

For the machine problem 2, we are creating a frame pool manager. The frame pool manager will manage the allocation and release of frames, and it will use for the kernel frame pool and process frame pool. The kernel frame pool will be 2MB, and the process frame pool will be 28MB. For the frame pool manager, we also need to design specific frames 15MB which will be off-limit to the user.

System Design

In 32MB memory, there are two pools: the kernel pool and the process pool. The kernel pool will be 2MB to 4MB, and the process pool will be 4MB to 32MB. For indicating the state of frame, the system will employ a bit map methodology where two bits of information frames are mapped to a frame. Using two bits, a byte of bit map will be 4 frames. When the frame is allocated, will get 00. When the frame is free, will get 11. When the frame is the contiguous frames and is the first frame, will get 10.

The figure 1 will show the memory design.

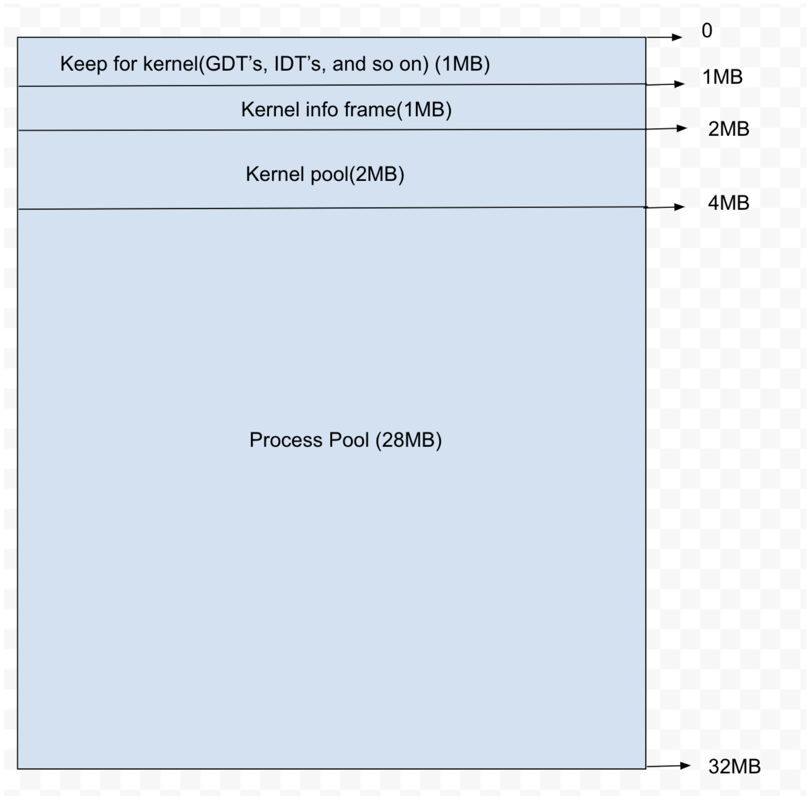


Figure 1: memory design

For the implementation: First, a frame uses 8 bits which the initial bit initial bit indicates whether the frame is allocated. For the second bit, if it is the start of the sequence, and rest of 6 bits are unused. Second, consider the frame pool in the form of a bitmap.

Code Description

cont_frame_pool.c: constructor: Sets up the data structures for the contiguous frame pool beginning from the starting base frame number and extending up to (base frame number + number of frames). Additionally, an info frame number is utilized to record the frame's status in a bitmap. If the info frame number is 0, the initial base frame number will be used as the info frame.

```
ContFramePool::ContFramePool(unsigned long _base_frame_no,
                             unsigned long _n_frames,
                             unsigned long _info_frame_no)
{
    base_frame_no = _base_frame_no;
    nframes = _n_frames;
    nFreeFrames = _n_frames;

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

    assert((nframes%8) == 0);

    BitmapInit(_info_frame_no);
    ptrHeadToNext();

    Console::puts("initialized\n");
}

//helper function: BitmapInit() which initialized bitmap
void ContFramePool::BitmapInit(unsigned long _info_frame_no) {
     int start = 0;
     if (_info_frame_no == 0){
        bitmap[0] = 0x7F;
        nFreeFrames--;
        start = 1;
     }

     for (int i = start; i < nframes; i++){
          bitmap[i] = 0xFF;
     }
}

//helper function: ptrHeadToNext
void ContFramePool::ptrHeadToNext(){
     ContFramePool** temp = &head;
     while(*temp){
           temp = &(*temp)->next;
     }

     *temp = this;
     this->next = NULL;
}
```

cont_frame_pool.c: get_frames: The API accepts the number of frames desired as input and delivers the head frame number in return. It looks for available frames, specifically "11" in the bitmap, marks the first available frame as the head frame with the label "10", and labels the rest of frames as allocated with "00". The API then provides the head frame number. This API only assigns contiguous frames and will bypass non-contiguous ones, even if they're accessible.

```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{

    if (_n_frames > nframes || _n_frames > nFreeFrames){
       Console::puts("ERROR:Allocate frames are more than available frames\n");
       assert(false);
    }

    for (unsigned int i = 0; i <= nframes - _n_frames; i++){
        if (checkFrame(i, _n_frames)) {
           checkBitmap(i, _n_frames);
           return base_frame_no + i;
        }
    }
    Console::puts("ERROR: Free Frame may not continuous\n");
    assert(false);
    return 0;

}

//helper function: checkFrame() which is check available frame
bool ContFramePool::checkFrame(unsigned index, unsigned int _n_frames){
     unsigned int j;
     for(j = index; j < index + _n_frames; j++){
        if (bitmap[j] == 0x7F || bitmap[j] == 0x3F) {
           return false;
        }
     }
     return true;
}

//helper function: checkBitmap()
void ContFramePool::checkBitmap(unsigned int i, unsigned int _n_frames){
     unsigned char mask = 0x80;
     unsigned int k;
     for(k = 0; k < _n_frames; k++){
        bitmap[i] = (k==0) ? (bitmap[i] ^ 0xC0) : (bitmap[i] ^ mask);
        nFreeFrames--;
        i++;
     }
}
```

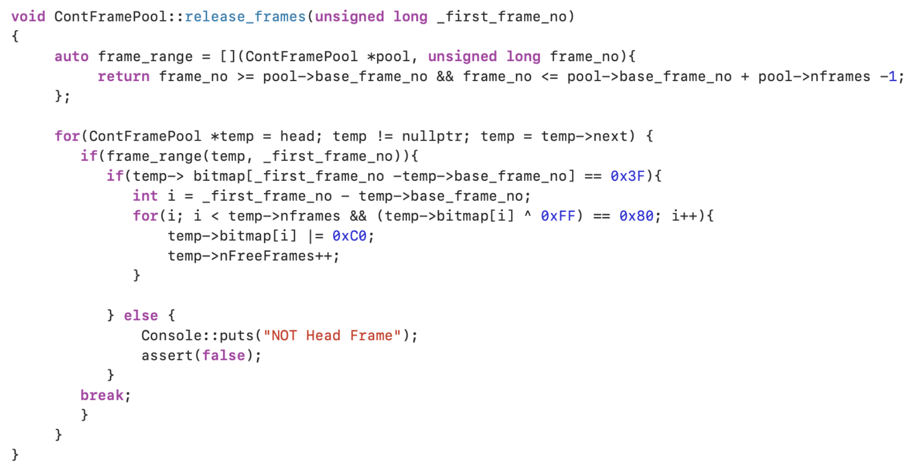
cont_frame_pool.c: mark_inaccessible : The API accepts both the base frame number and the frame count as input and doesn't produce any output. This function points and marks frames—from the initial base frame number to the sum of the base frame number and the specified frame count—as head frames, ensuring they become inaccessible .

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

{
     unsigned long frame_mark = _base_frame_no + _n_frames;
     unsigned char mask = 0xC0;

     if(_base_frame_no < base_frame_no || frame_mark > base_frame_no + nframes){
        Console::puts("OUT OF BOUNDS");
        assert(false);
     }
     for(unsigned long i = _base_frame_no; i < frame_mark; i++){
         unsigned int bitmap_i = i - base_frame_no;
         assert((bitmap[bitmap_i] & mask) !=0);
         bitmap[bitmap_i] ^= mask;
     }
     nFreeFrames -= _n_frames;
}
```

cont_frame_pool.c: release_frames: The API accepts the head frame number as input and doesn't return a value. This function initially determines whether the frame is part of the process pool or kernel pool. It then labels the head frame and the rest of allocated frames as free until it encounters another head frame or a free frame.

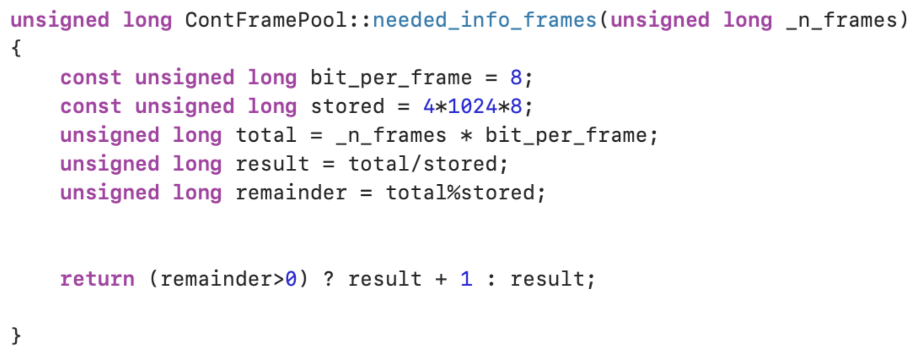


```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
     auto frame_range = [](ContFramePool *pool, unsigned long frame_no){
          return frame_no >= pool->base_frame_no && frame_no <= pool->base_frame_no + pool->nframes -1;
     };

     for(ContFramePool *temp = head; temp != nullptr; temp = temp->next) {
        if(frame_range(temp, _first_frame_no)){
           if(temp-> bitmap[_first_frame_no -temp->base_frame_no] == 0x3F){
              int i = _first_frame_no - temp->base_frame_no;
              for(i; i < temp->nframes && (temp->bitmap[i] ^ 0xFF) == 0x80; i++){
                  temp->bitmap[i] |= 0xC0;
                  temp->nFreeFrames++;
              }

           } else {
               Console::puts("NOT Head Frame");
               assert(false);
           }
        break;
        }
     }
}
```

cont_frame_pool.c: needed info frames : The API receives the total count of frames as input and provides the number of info frames needed to house those frames. In a basic pool, one bit is designated for frame addressing, allowing storage of up to 32k frames in the bitmap. Conversely, in the contiguous pool, where 2 bits are allocated for each frame, the bitmap can store up to 16k frames. Therefore, here is function:
 Total frame = 8* nframes;
Stored = 16K;
Remainder = total frame%stored;
If remainder >0 then result + 1. Otherwise, just result



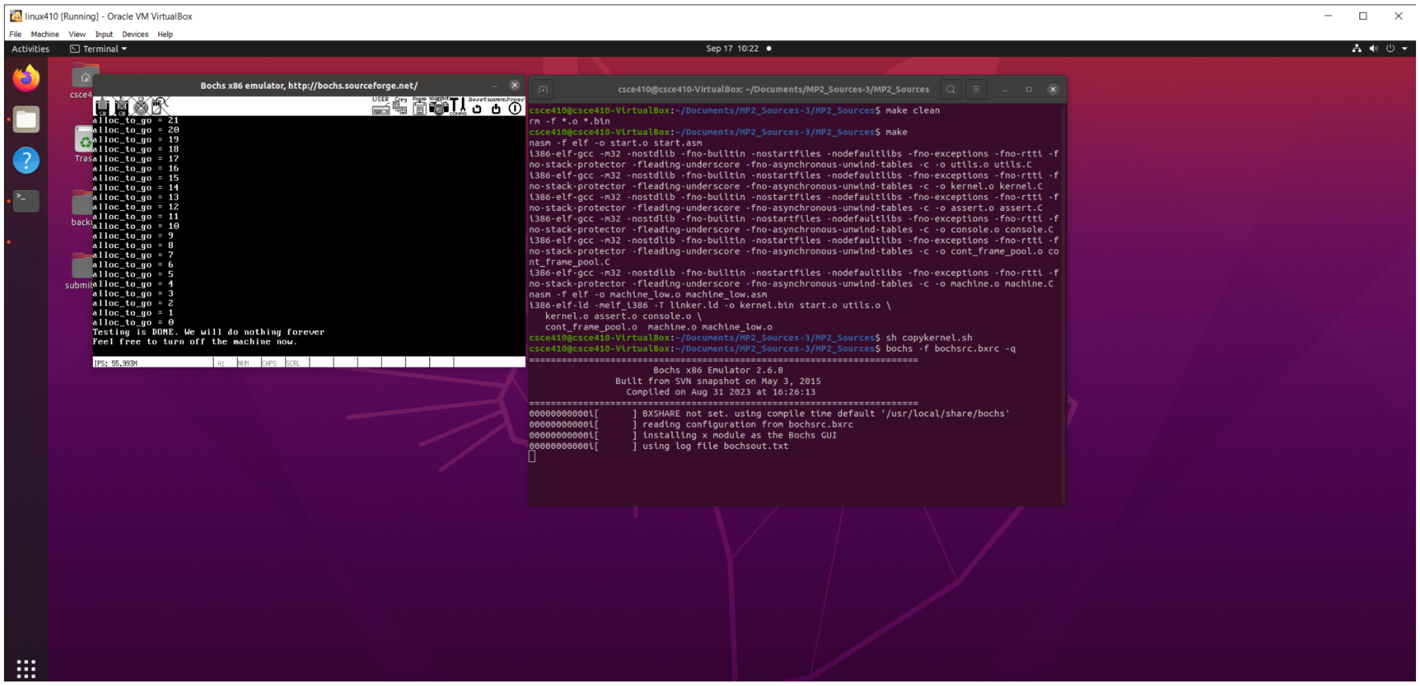
```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    const unsigned long bit_per_frame = 8;
    const unsigned long stored = 4*1024*8;
    unsigned long total = _n_frames * bit_per_frame;
    unsigned long result = total/stored;
    unsigned long remainder = total%stored;


    return (remainder>0) ? result + 1 : result;

}
```

Testing

Figure : This is output from the main()


linux410 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Sep 17 10:22
Bochs x86 emulator, http://bochs.sourceforge.net/
csce410@csce410-VirtualBox: ~/Documents/MP2_Sources-3/MP2_Sources

```
alloc_to_go = 21
alloc_to_go = 20
alloc_to_go = 19
alloc_to_go = 18
alloc_to_go = 17
alloc_to_go = 16
alloc_to_go = 15
alloc_to_go = 14
alloc_to_go = 13
alloc_to_go = 12
alloc_to_go = 11
alloc_to_go = 10
alloc_to_go = 9
alloc_to_go = 8
alloc_to_go = 7
alloc_to_go = 6
alloc_to_go = 5
alloc_to_go = 4
alloc_to_go = 3
alloc_to_go = 2
alloc_to_go = 1
alloc_to_go = 0
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.
IPS: 55.993M
```

```
csce410@csce410-VirtualBox:~/Documents/MP2_Sources-3/MP2_Sources$ make clean
rm -f *.o *.bin
csce410@csce410-VirtualBox:~/Documents/MP2_Sources-3/MP2_Sources$ make
nasm -f elf -o start.o start.asm
i386-elf-gcc -m32 -nostdlib -fno-builtin -nostartfiles -nodefaultlibs -fno-exceptions -fno-rtti -f
no-stack-protector -fleading-underscore -fno-asynchronous-unwind-tables -c -o utils.o utils.C
i386-elf-gcc -m32 -nostdlib -fno-builtin -nostartfiles -nodefaultlibs -fno-exceptions -fno-rtti -f
no-stack-protector -fleading-underscore -fno-asynchronous-unwind-tables -c -o kernel.o kernel.C
i386-elf-gcc -m32 -nostdlib -fno-builtin -nostartfiles -nodefaultlibs -fno-exceptions -fno-rtti -f
no-stack-protector -fleading-underscore -fno-asynchronous-unwind-tables -c -o assert.o assert.C
i386-elf-gcc -m32 -nostdlib -fno-builtin -nostartfiles -nodefaultlibs -fno-exceptions -fno-rtti -f
no-stack-protector -fleading-underscore -fno-asynchronous-unwind-tables -c -o console.o console.C
i386-elf-gcc -m32 -nostdlib -fno-builtin -nostartfiles -nodefaultlibs -fno-exceptions -fno-rtti -f
no-stack-protector -fleading-underscore -fno-asynchronous-unwind-tables -c -o cont_frame_pool.o co
nt_frame_pool.C
i386-elf-gcc -m32 -nostdlib -fno-builtin -nostartfiles -nodefaultlibs -fno-exceptions -fno-rtti -f
no-stack-protector -fleading-underscore -fno-asynchronous-unwind-tables -c -o machine.o machine.C
nasm -f elf -o machine_low.o machine_low.asm
i386-elf-ld -melf_i386 -T linker.ld -o kernel.bin start.o utils.o \
   kernel.o assert.o console.o \
   cont_frame_pool.o  machine.o machine_low.o
csce410@csce410-VirtualBox:~/Documents/MP2_Sources-3/MP2_Sources$ sh copykernel.sh
csce410@csce410-VirtualBox:~/Documents/MP2_Sources-3/MP2_Sources$ bochs -f bochsrc.bxrc -q
========================================================================
                       Bochs x86 Emulator 2.6.8
                Built from SVN snapshot on May 3, 2015
                  Compiled on Aug 31 2023 at 16:26:13
========================================================================
00000000000i[      ] BXSHARE not set. using compile time default '/usr/local/share/bochs'
00000000000i[      ] reading configuration from bochsrc.bxrc
00000000000i[      ] installing x module as the Bochs GUI
00000000000i[      ] using log file bochsout.txt
```

Test Case Scenario: Return Error Case
To examine the error situation in which the requested frames aren't contiguous, we allocate a process pool of 8 frames.

get_frames(4) -> 1024,1025,1026,1027
mark_inaccessible(1029 ,3) -> 1029,1030,1031,1032 which are marked inaccessible.
Therefore, free frames -> 1028, 1033
get_frames(2) -> Return Error

Test Case Scenario: Testing function for marking frame ->passed
To examine certain frames in the kernel pool as 'mark_inaccessible', 'get_frames', 'release_frames'. Marking inaccessible frame 510 and 511. Get 509 for get_frames(1) when requesting 1 frame. Requesting 5 frames which allocates 512-516. Do the release_frames(516) which will release contiguous frame from 512 to 516.