Machine Problem 2

Jicheng Lu, 525004048

1. Introduction

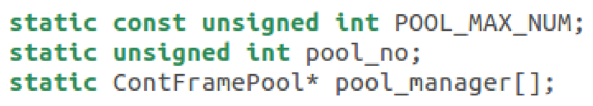
In Machine Problem 2, we build a frame manager, which is used to manage the physical memory in the manager. The total amount of memory in the machine is 32MB with a kernel space from 0 to 4MB and a user space from 4MB to 32MB. Memory within the first 4MB is direct-mapped to the physical memory.

We build two frame pools: the kernel frame pool and the process frame pool. The kernel frame pool manages the memory between 2MB to 4MB (Note that the first 1MB contains all global data, video memory and other staff and the second 1MB contains the kernel code. Thus, the kernel frame pool actually manages the memory from 2MB to 4MB). On the other hand, the process frame pool manages the memory above 4MB. Each frame is 4KB in size.

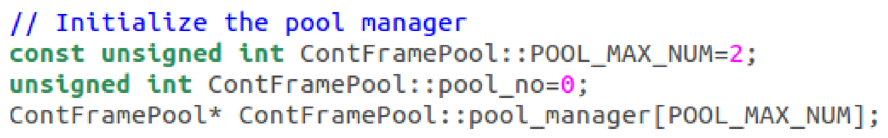
2. Frame Pool Manager

We first define a pool manager, which is implemented to identify the specific frame pool that is handling the desired frames.

We give the maximum pool number to determine the quantity of pools in the pool manager. Here we have two frame pools, so the maximum pool number is 2. We define the pool manager in cont_frame_pool.H file and initialize is in cont_frame_pool.C file.

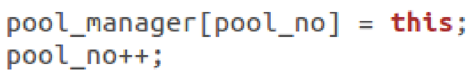


```
static const unsigned int POOL_MAX_NUM;
static unsigned int pool_no;
static ContFramePool* pool_manager[];
```



```
// Initialize the pool manager
const unsigned int ContFramePool::POOL_MAX_NUM=2;
unsigned int ContFramePool::pool_no=0;
ContFramePool* ContFramePool::pool_manager[POOL_MAX_NUM];
```

Whenever we initialize a new frame pool, we store the information of the current frame pool to the pool manager. The following the code is added in the initialize function in cont_frame_pool.C.



```
pool_manager[pool_no] = this;
pool_no++;
```

3. Frame Pool

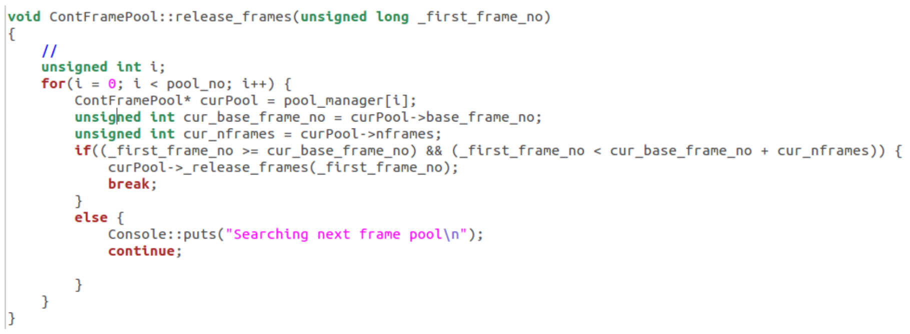
The Contiguous Frame Pool has the ability to allocate either a single frame or sequences of contiguous frames. Instead of maintaining whether a frame is FREE or ALLOCATED, we maintain whether the frame is FREE, or ALLOCATED, or HEAD-OF-SEQUENCE.

In this case, we use one character per frame: 0 -> FREE, 1 -> ALLOCATED, 2 -> HEAD-OF-SEQUENCE. (I am not so good at bit manipulation because of non-CSE background. I can figure out how to use two bits per frame later.)

In the pool initialization, we first set all the states in bitmap to be zero. Then we mark the frames that are used to store the management information.

The 'get_frames' function searches a number of contiguous frames within the frame pool. To achieve this, we find the first zero in the bitmap of current frame pool, and then look for a sequence of zeros with desired length. After we get these frames, we mark their states as ALLOCATED, and the first one is marked as HEAD-OF-SEQUENCE. We achieve this by updating the bitmap and the number of free frames in the function 'mark_inaccessible'.

When it comes to releasing frames, we first need to identify whether the frame pool actually owns the frame. Here we use the 'pool manager' to achieve the identification.



```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    //
    unsigned int i;
    for(i = 0; i < pool_no; i++) {
        ContFramePool* curPool = pool_manager[i];
        unsigned int cur_base_frame_no = curPool->base_frame_no;
        unsigned int cur_nframes = curPool->nframes;
        if((_first_frame_no >= cur_base_frame_no) && (_first_frame_no < cur_base_frame_no + cur_nframes)) {
            curPool->_release_frames(_first_frame_no);
            break;
        }
        else {
            Console::puts("Searching next frame pool\n");
            continue;

        }
    }
}
```

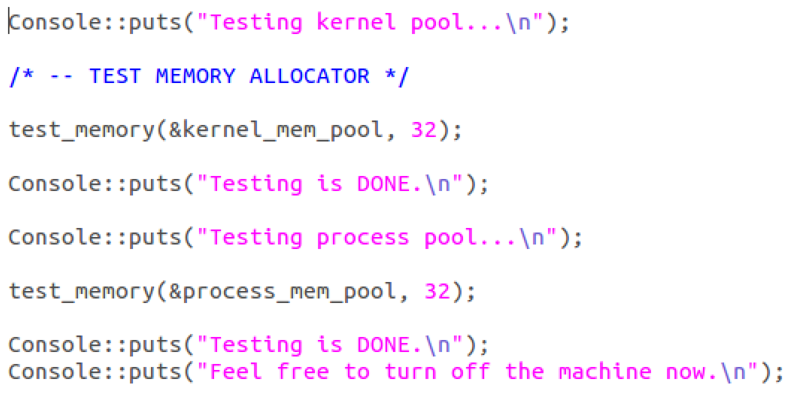
After we have identified the frame pool, we find a sequence of frames that are previously used and change their states to be zero (i.e., release those frames). We update the bitmap and the number of free frames in the non-static function ‘_release_frames’.

In the function ‘needed_info_frames’, we returns the number of frames needed to manage a frame pool of size ‘_n_frames’. For example, for the kernel frame pool, we have 512 frames, so we only need one frame to store the management information. For the process frame pool, we have 7*1024 frames, so we need two frames to store the management information. (each frame size is 4KB).

4. Test

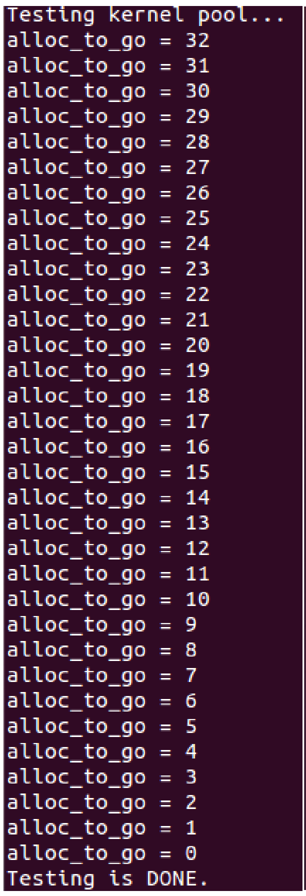
We test the correctness of this frame pool using the default method. i.e., test_memory in kernel.C. That is, we pick different numbers of free frames inside the kernel pool or process pool and put different integers into the frames. For example, when _alloc_to_go = 32, we get one frame and put 1024 intergers of 32 inside that frame, since an integer takes up 4 bytes. When _alloc_to_go = 31, we get 4 frames and put 1024 intergers of 31 inside each frame. We keep doing this by decreasing the value of ‘_alloc_to_go’. Once we finish the integer input, we start to check whether the integers in each frame are the same as we desired. If yes, then we know that the frame pool can handle the free frames. To be specific, when we do a lot of ‘get frames’, the frame pool can successfully obtain the contiguous free frames with desired numbers and release them once we finish the task. Moreover, if we have more than one process frame pool, we can change the maximum pool number in the pool manager and redefine the memory size in each frame pool.

The following figure shows the modified part in kernel.C.

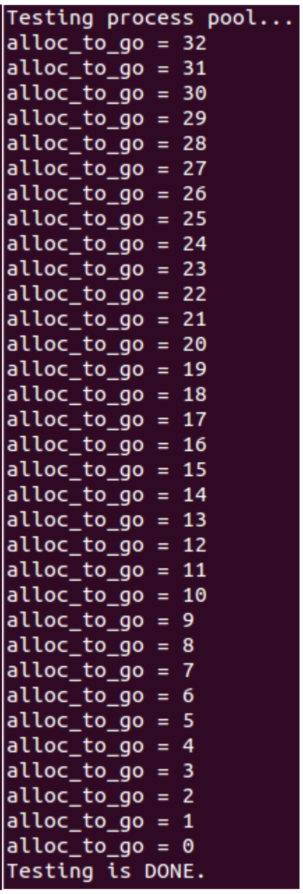


```
Console::puts("Testing kernel pool...\n");

/* -- TEST MEMORY ALLOCATOR */

test_memory(&kernel_mem_pool, 32);

Console::puts("Testing is DONE.\n");

Console::puts("Testing process pool...\n");

test_memory(&process_mem_pool, 32);

Console::puts("Testing is DONE.\n");
Console::puts("Feel free to turn off the machine now.\n");
```

The testing results are given below:



```
Testing kernel pool...
alloc_to_go = 32
alloc_to_go = 31
alloc_to_go = 30
alloc_to_go = 29
alloc_to_go = 28
alloc_to_go = 27
alloc_to_go = 26
alloc_to_go = 25
alloc_to_go = 24
alloc_to_go = 23
alloc_to_go = 22
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.
```



```
Testing process pool...
alloc_to_go = 32
alloc_to_go = 31
alloc_to_go = 30
alloc_to_go = 29
alloc_to_go = 28
alloc_to_go = 27
alloc_to_go = 26
alloc_to_go = 25
alloc_to_go = 24
alloc_to_go = 23
alloc_to_go = 22
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.
```