

Machine Problem 2: Frame Manager

Design :

In this machine programming assignment, we have implemented a Frame Manager, which manages the allocation of frames (physical pages). The frame manager is responsible for the allocation and the release of frames, i.e, it needs to keep track of which pages are being used and which ones are free. Our system has a total amount of 32 MB memory. We have kernel space from 0 to 4 MB and the process space from 4MB to 32MB. There also is a 1MB hole of inaccessible memory starting at 15 MB. The kernel space will be direct mapped from virtual memory to physical and shared between process page tables. User space memory will be managed. Both the frames and the pages are 4 KB in size.

The frame pool manages contiguous frames of memory, memory below 4 MB is managed by our kernel frame pool and memory above 4 MB is managed by our process frame pool. I have created a linked list structure to bind all the constructed `cont_frame_pool` objects.

In this assignment, the status of each frame is recorded in the bitmap as follows:

BITMAP VALUE	STATUS
00	FREE
01	HEAD OF SEQUENCE
10	INACCESSIBLE
11	ALLOCATED

Changes made in file `cont_frame_pool.H` :

Defined the following data structures :

```
unsigned char * bitmap;          // We implement the frame pool with a bitmap
unsigned int  nFreeFrames;      // size in frames of the frame pool
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?
unsigned long n_info_frames;    // the number of consecutive frames needed to store the management
information for the frame pool
```

```
static ContFramePool * pools; // linked list that stores all the frame pools
static ContFramePool * head; // pointer to the starting node of the list
ContFramePool * next;        // points to the next node in the list
```

FUNCTIONS USED :

- **Initialize**

We initialize all the data structures here in the constructor.

- **get_frames(_n_frames):** Traverse the “bitmap” of states and look for a sequence of at least _n_frames entries that are FREE. If you find one, mark the first one as HEAD-OF-SEQUENCE and the remaining _n_frames-1 as ALLOCATED. If successful, it returns the frame number of the first frame else 0.

Since each bitmap value can store the state of 4 frames, we can consider the bitmap as a 2-D array, where m is for row number and n is for column number. Here we need to find the starting values of (m,n) so that we will be able to allocate _n_frames. Once we get these values, we mark the first one as 01 and the remaining _n_frames as 11.

Bit manipulation done here :

```
unsigned char mask1 = 0x3F; //0011 1111
mask1 = mask1 >> (n*2) | mask1 << (8 - n*2); //right circular bit shift
unsigned char mask2 = 0x40; //0100 0000
mask2 = mask2 >> (n*2);
bitmap[m] = ((bitmap[m] & mask1) | mask2); //01 - Head of sequence
mask = 0xC0; //1100 0000
bitmap[i] = (bitmap[i] | mask); // done inside a for loop, 11 - allocated
```

- **mark_inaccessible(_base_frame_no, _n_frames):** This is similar with get_frames, except that we don't need to search for the free sequence. We tell the allocator exactly which frame to mark as HEAD-OF-SEQUENCE and how many frames after that to mark as ALLOCATED.

Bit manipulation done here:

```
unsigned char mask1 = 0x80; // 1000 0000
unsigned char mask2 = 0x3F; // 0011 1111
unsigned char mask3 = 0x40; // 0100 0000
```

```

mask1 = mask1 >> (n*2);
mask3 = mask3 >> (n*2);
mask2= mask2 >> (n*2) | mask2 << (8 - n*2);
bitmap[m] = (bitmap[m] & mask2) | mask3; //01 - head of sequence
bitmap[m] = (bitmap[m] & mask2) | mask1; //10 - inaccessible, inside loop

```

- **release_frames(_first_frame_no):** Check whether the first frame is marked as HEAD-OF-SEQUENCE. If not, something went wrong. If it is, mark it as FREE. Traverse the subsequent frames until you reach one that is FREE or HEAD-OF-SEQUENCE. Until then, mark the frames that you traverse as FREE

Here we first find out which pool this frame belongs to using the linked list we had initialized earlier. If the first frame is not Head of sequence, then we throw an error, otherwise, we find the values of (m,n) and do the following bit manipulation to mark the frames as 00 :

```

unsigned char *bits = cur->bitmap; //bitmap of the current pool
unsigned char mask1 = 0xC0; // 1100 0000
unsigned char mask2 = 0x40; // 0100 0000
unsigned char mask = 0x3F; // 0011 1111
mask1 = mask1 >> (n*2);
mask2 = mask2 >> (n*2);
mask = mask >> (n*2) | mask << (8 - n*2); //right circular bit shift
bits[m] = bits[m] & mask;

```

- **needed_info_frames(_n_frames):** This depends on how many bits you need to store the state of each frame. Returns the number of info frames required.

Here page size is 4KB => 4 * 1024 B and each byte can hold the status of 4 frames

Therefore, one page can hold the status of 16KB frames.

So number of info frames = ($_n_frames / (16 \text{ KB}) + (_n_frames \% (16 \text{ KB}) > 0 ? 1 : 0)$)

Note : The code is well commented, so the logic will mostly be evident from the code itself.

Output :

```
1> test_memory(&kernel_mem_pool, 32)
```

Bochs x86-64 emulator, http://bochs.sourceforge.net/

lib/bochs/plugins'
s'

o-rtti -fno-stack-protect

=====

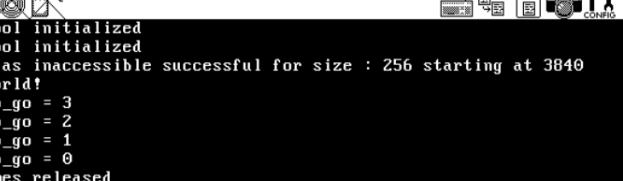
Bochs x86 Emulator 2.4.6
Build from CVS snapshot, on February 22, 2011
Compiled at Nov 11 2011, 09:31:18

=====

```
00000000000i[ ] LTDL_LIBRARY_PATH not set. using compile time default '/usr/lib/bochs/plugins'
00000000000i[ ] BXSHARE not set. using compile time default '/usr/share/bochs'
00000000000i[ ] reading configuration from bochsrc.bxrc
00000000000i[ ] lt_dlhandle is 0x238f980
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[ ] installing x module as the Bochs GUI
00000000000i[ ] using log file bochsout.txt
```

CTRL + 3rd button enables mouse A: NUM CAPS SCRL

```
2> test_memory(&kernel_mem_pool, 3)
    test_memory(&process_mem_pool, 3)
```



Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

USER Copy Paste Reset suspend Power

Frame Pool initialized
Frame Pool initialized
Marking as inaccessible successful for size : 256 starting at 3840
Hello World!
alloc_to_go = 3
alloc_to_go = 2
alloc_to_go = 1
alloc_to_go = 0
All frames released
All frames released
All frames released

Process mem pool test
alloc_to_go = 3
alloc_to_go = 2
alloc_to_go = 1
alloc_to_go = 0
All frames released
All frames released
All frames released
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.