CSCE 611: MP2 : DESIGN DOCUMENT

AUTHOR: SRIVIDHYA BALAJI
UIN: 827007169

PROBLEM STATEMENT:

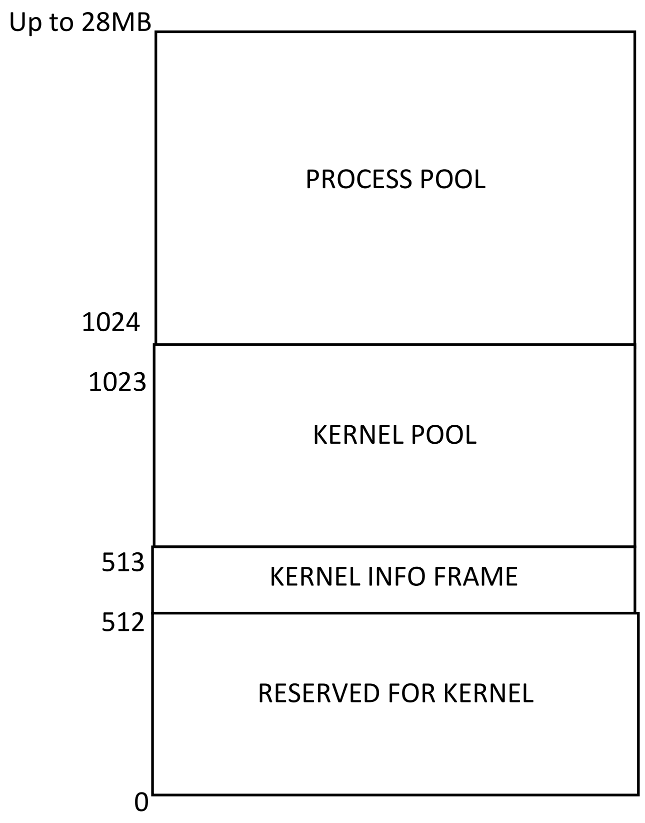
To implement a simple Frame Pool Manager.

CONSTRAINTS:

1) Total Memory in the machine = 32 MB
2) Kernel Memory Pool : 4 MB ; Location : 512 -1023
3) Process Memory Pool: 28MB ; Location : 1024 – 8191
4) Frame Size : 4KB = 4 *1024 * 8 = 32k bits

OVERALL DESIGN:

SCHEMATIC OF OVERALL MEMORY



APPROACH BASED ASSUMPTIONS:

1) Considering frame pool as bit-map.
2) 8 bits is used for a frame, where first bit represents if the frame is allocated or not and second bit represents if it is head of sequence and remaining 6 bits are free. (Initial approach)

NOTE: OPTIMIZATION MEASURE: To use only 2 bits per frame, and hence represent information for 4 frames in 8 bits. (APPENDIX A)

STATE TABLE:

STATE	ALLOCATED	HEAD OF SEQUENCE	DESCRIPTION
11	No	No	Represents Free Frame
01	Yes	No	Represents allocated non- head frame
00	Yes	Yes	Represents allocated Head of Frame

The objects and interfaces of class ConstFramePool is used to create and manage the Framepools

PART 1: CREATING A FRAME POOL

Creating an object of class type ConstFramePool by passing the necessary parameters such as Start Frame, Pool Size, Info Frame, Number of Info Frames.

Scenario 1: Creating a Kernel Frame Pool

Determine the number of info frames required for creating a bitmap of the required size . For Kernel Pool, this is 0 as the info frame is created within the Kernel based on the Pool size and the base frame number.

Pseudo Code for Kernel Pool :

```
ContFramePool kernel_mem_pool(KERNEL_POOL_START_FRAME,
                              KERNEL_POOL_SIZE,
                               0,  0);
```

Here, Kernel_mem_pool object is created and initialized with START FRAME, POOL SIZE. Since information frame is mentioned as zero, the first frame is assigned as the Info frame.

```
If info_frame_no == 0
   bitmap[0] = 0x7F : Assigning as Info Frame for Kernel
```

Scenario 2: Creating Process Frame Pool

For Process Frame Pool it is necessary to decide on the number of info frames required and then get n_info_frames from the Kernel Pool to store the Process Frame Pool Management information

(A) **Needed_info_frames(n frames) :**

Based on memory size allocated for the pool and the Frame size per frame, the number of info_frames can be calculated. Here,

Frame Size = 4KB = 4 *1024*8 = 32K bits
Number of Frames in 28 MB process pool = 28 MB / 4 KB = 7168 Frames
Number of info Frames (considering 1 bit per frame) = (7168/ 32000) + (7168 % 32000 > 0 ? 1 :0)
= 1 frame

Considering 8 bits per frame, n_info_frames = (7168*8/ 32000) + ((7168*8) % 32000 > 0 ? 1 :0)
= 2 frames

In general ,

n_info_frames = [(_n_frames * _n_info_bits)/ Frame size in bits] + [(_n_frames % n_info_bits)/Frame Size) > 0 ? 1 :0]

, where _n_info_bits = Number of bits per info frame

(B) **GetFrames of _n_info_frame from Kernel Pool :**

unsigned long process_mem_pool_info_frame = kernel_mem_pool.get_frames(n_info_frames);

where, process_mem_pool_info_frame : returns the head frame from where the info frames start. In this case, it returns the first available space in the Kernel pool from 513.

get_frames() shall be discussed in PART 2 in detail.

(C) **Creating Frame Pool Object :**

```
ContFramePool process_mem_pool(PROCESS_POOL_START_FRAME,
                               PROCESS_POOL_SIZE,
                               process_mem_pool_info_frame,
                               n_info_frames);
```

This creates an object process pool object called process_mem_pool , with the corresponding parameters

TEST SCENARIOS CONSIDERED AND PASSED :

1) Creating a Kernel Frame Pool
2) Creating a Process Frame Pool
3) Creating Multiple Process Frame Pool

PART 2: GETFRAMES(N FRAMES)

DESCRIPTION : This function, checks for first available contiguous memory space for N FRAMES , allocates it in the requested pool and returns the location of the HEAD of the contiguous space.
Below, is a rough Pseudo Code for the function

PSEUDO CODE :

```
get_frames(_n_frames)

If _n_frames > nMaxFrames or > nFreeFrames
  Return Error
X:
While bitmap[i] == allocated
   i++
start_frame_no = base_frame_no + i                    // Returns start location of first free frame
for : j = startframe to startframe + _n_frames
     if bitmap[j] allocated,
     Repeat X
 /*  End of Loop  indicates startframe of available contiguous requested space * /
while _n_frames! =0
     Allocate bitmap[i] and first bitmap[i] as HEAD
     nFreeFrames -> nFreeFrames -1
    _n_frames -> _n_frames -1
     i++

return(startframe)
```

TEST SCENARIOS CONSIDERED AND PASSED:

1) _n_frames is within range of allocated memory. And is available continuously.
2) _n_frames > nMaxFrames in a pool
3) _n_frames > nFreeFrames in the pool
4) _n_frames within range but not continuously available from the first free space
5) _n_frames within range but not continuously available throughout nMaxFrames
6) _n_frames in a sequence containing allocated and Inaccessible frames in between

PART 3: RELEASE FRAMES(FRAME_NUMBER)

DESCRIPTION : This function checks for the pool, FRAME_NUMBER is part of and based on info_frame of the pool, checks if it is a HEAD FRAME and continuously de-allocates or frees the frames till the next HEAD. That is, it releases the frames that are allocated as part of the contiguous memory allocation from FRAME_NUMBER as HEAD

(A) FINDING THE FRAME POOL :

While creating the frame pools, the pools are added in a Linked list of ContFramePool object type. Based on baseframeNumber and Number of frames parameters of the object, the Frame Pool (Node) corresponding to the Frame Number is found. Within the Frame Pool, the corresponding Frame Number is accessed and released.

Pseudo Code for Adding Pools in Linked List :

```
If HEAD = NULL
   Head = this
   Head-> next = NULL
Else
  While(temp! = NULL);
temp->next = this;
```

Pseudo Code for Finding the Pool corresponding to Frame Number:

```
For : temp = head ; temp!= NULL; temp = temp->next
    If(( FRAME NUMBER > = temp->base_frame_no)  && (FRAME NUMBER <= temp->nFrames-1)
        temp->release_frame(FRAME NUMBER);
// Calling the corresponding Pool object's release frame
```

(B) RELEASING THE FRAMES FROM THE FRAME POOL

Retrieve the corresponding offset of the frame in the pool and check if the bit-map of it is allocated as HEAD frame. If yes, continue de-allocating till next HEAD frame. Else throw error

Pseudo Code for Release Frames in a pool:

```
If FRAME NUMBER – BASE FRAME NUMBER != HEAD
   Print ERROR
For : i = (FRAME NUMBER –BASE FRAME NUMBER) ; i<nMaxFrames; i++
     Deallocate bitmap[i]
     If(bitmap[i] ==  HEAD)
        break;
```

TEST SCENARIOS CONSIDERED AND PASSED :

1) Release frame number within Kernel Pool
2) Release frame number with Process Pool
3) Release frame number in none of the Pools
4) Release non- HEAD frame number

PART 4: MARK INACCESSIBLE (FRAME NUMBER, NFRAMES) :

To mark inaccessible, start from the FRAME NUMBER and till NFRAMES, see if already in use. If so, throw error. Else mark them all as HEAD. (ASSUMPTION: CONSIDERING MARKING AS HEAD AS MARKING INACCESSIBLE)

Pseudo Code:

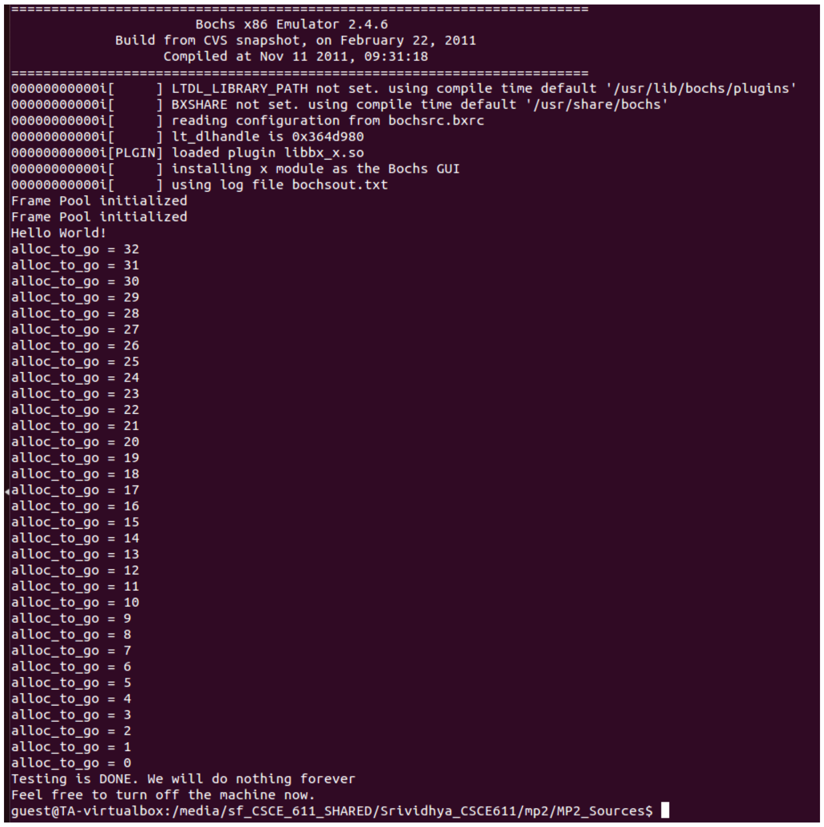
```
Mark_inaccessible (_base_frame_no , _n_frames)
For : i= _base_frame_no ; i < _base_frame_no+_n_frame; i++
    If I < = pool_base_frame_no && i >=pool_base_frame_no+ MAXPoolFrames
      Print Out of Bounds Error
    Else
      If(Bitmap[i] already in use)
        Print already in USE
      Else
        Bitmap[i] = HEAD //mark 00 STATE
```

TEST SCENARIOS CONSIDERED:

1) Mark inaccessible frames within a frame pool
2) Mark inaccessible frames with _base_frame_number in pool and _n_Frames exceeding nMaxPoolFrames
3) Mark inaccessible frames which are already in use

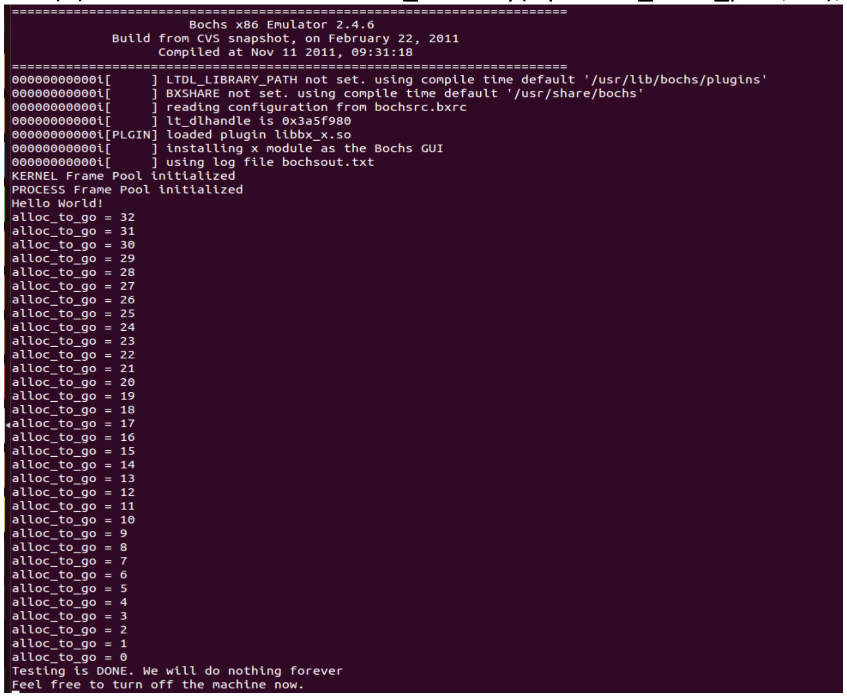
SCREENSHOTS OF RUNNING TEST MEMORY IN KERNEL.C

(A) FOR KERNEL POOL: test_memory(&kernel_mem_pool, 32);



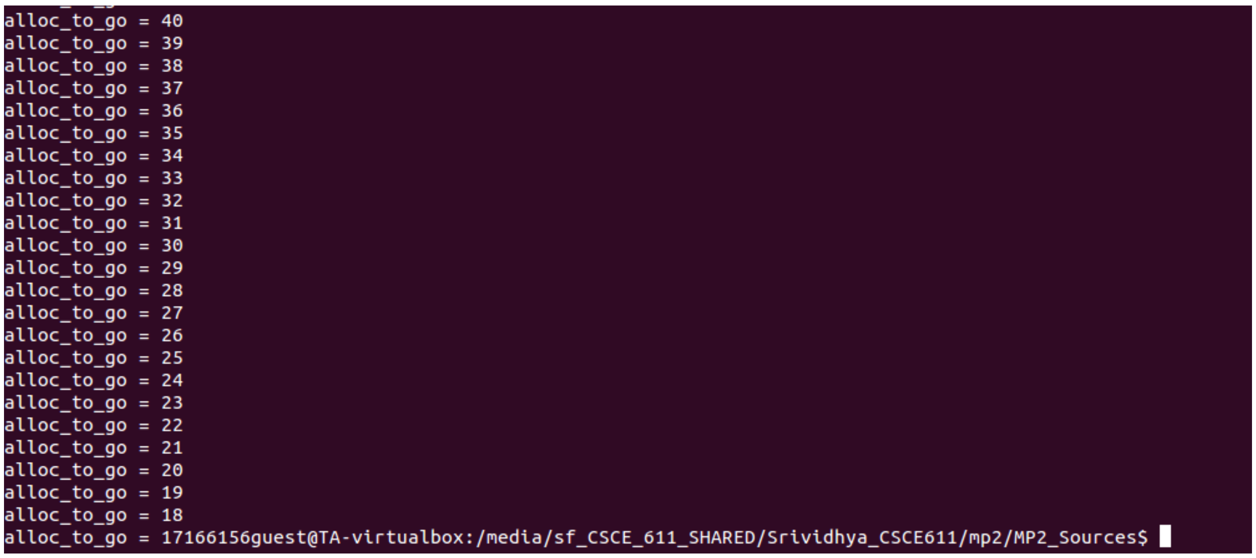
```
========================================================================
                       Bochs x86 Emulator 2.4.6
             Build from CVS snapshot, on February 22, 2011
                   Compiled at Nov 11 2011, 09:31:18
========================================================================
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 0x364d980
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
Frame Pool initialized
Frame Pool initialized
Hello World!
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. We will do nothing forever
Feel free to turn off the machine now.
guest@TA-virtualbox:/media/sf_CSCE_611_SHARED/Srividhya_CSCE611/mp2/MP2_Sources$
```

(B) FOR PROCESS POOL : test_memory(&process_mem_pool, 32);



```
========================================================================
                       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 0x3a5f980
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
KERNEL Frame Pool initialized
PROCESS Frame Pool initialized
Hello World!
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. We will do nothing forever
Feel free to turn off the machine now.
```

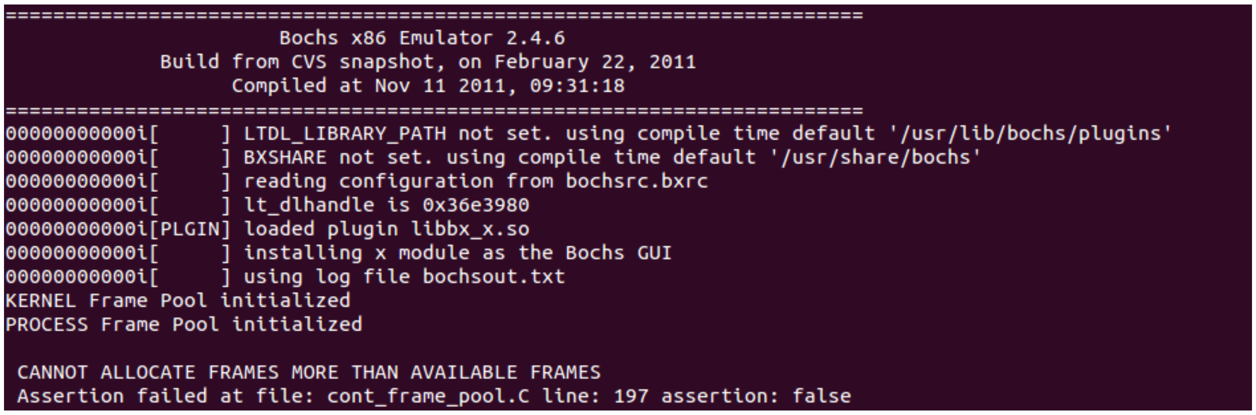
(C) ERROR SCENARIO WHILE TESTING TEST MEMORY WITH A VERY LARGE VALUE GREATER THAN 190 Ie, FOR KERNEL POOL: test_memory(&kernel_mem_pool, 200);



```
alloc_to_go = 40
alloc_to_go = 39
alloc_to_go = 38
alloc_to_go = 37
alloc_to_go = 36
alloc_to_go = 35
alloc_to_go = 34
alloc_to_go = 33
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 = 17166156guest@TA-virtualbox:/media/sf_CSCE_611_SHARED/Srividhya_CSCE611/mp2/MP2_Sources$
```

In this case, the system crashes due to insufficient memory for the recursion(probably). However, in the test suite getFrames(4000) returns an ERROR Message according to the expected functionality. This can be seen in the screenshot below.

(D) unsigned int k = kernel_mem_pool.get_frames(4000);

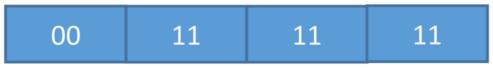


```
========================================================================
                       Bochs x86 Emulator 2.4.6
             Build from CVS snapshot, on February 22, 2011
                   Compiled at Nov 11 2011, 09:31:18
========================================================================
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 0x36e3980
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
KERNEL Frame Pool initialized
PROCESS Frame Pool initialized

 CANNOT ALLOCATE FRAMES MORE THAN AVAILABLE FRAMES
 Assertion failed at file: cont_frame_pool.C line: 197 assertion: false
```

APPENDIX: A : OPTIMIZATION :

Since the approach considered, uses 8 bits per frame in which only 2 bits are used to save the information regarding the frames and 6 bits are left free or unused. An optimized technique would be to use two bits per frame.



Get_frames 2 bit Implementation Pseudo Code:

```
While( i<=nFrames)
For : j =each two bits in bitmap[i]
          k = bitmap[i] & mask   //   Mask the bitmap[i] , to extract two bits at a time
          if(k == state 01 || k == state 00)
               mask = mask>>2   // Check the next two bits
          else
              save i  // Frame Number
             save j // Offset within the Frame
       /* This means the jth offset in ith frame is free . Check for continuous availability and allocate*/
While(nFrames! = 0)
           Till mask != 0
                       Bitmap[i] = Bitmap[i]^mask
                       Mask = mask>>2
            nFreeFrames --
            nFrames --
i++
Reset Mask
```

APPENDIX B : USED TEST SUITE:

Below are a set of few test scenarios that were used to test the implementation.

SCENARIO: 1 TEST RESULT : PASSED :

Description: To mark few frames in the Kernel pool as inaccessible, getframes , release specific frames and get specific frames.

Here initially marking 514, 515 as inaccessible. Requesting for 1 frame, which returns 513 and allocates it.

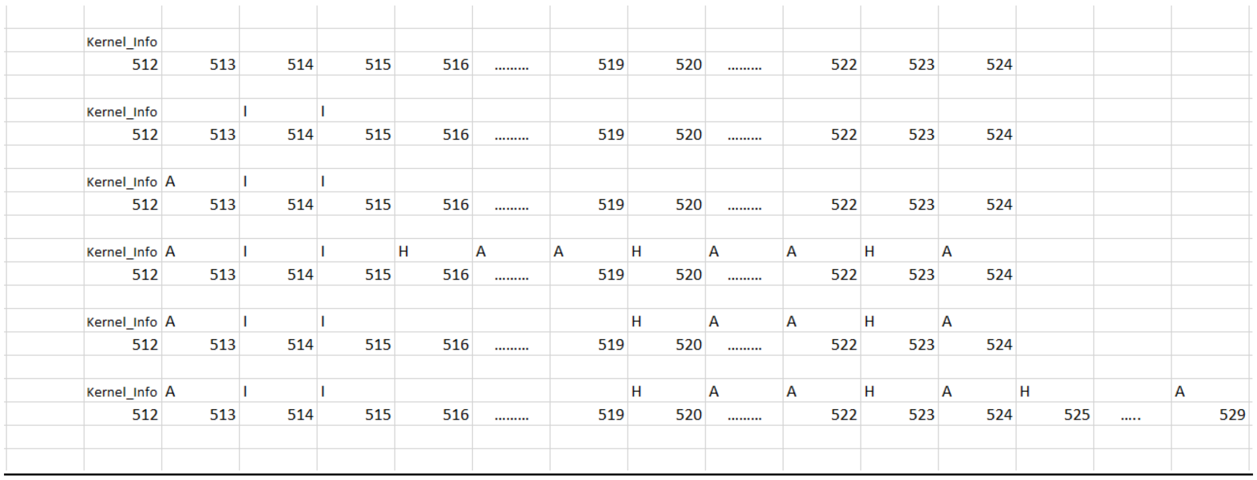
Requesting for 4 frames, which allocates 516-519 , similarly for 3, 2 frames, thus allocating till 524. Releasing(516). This releases continuous frames allocated from 516-519. Finally requesting for 5 frames which allocates from 525-529. The step wise output has been represented in the Output .

Code SNIPPET part of Kernel.C

```
…
…
kernel_mem_pool.mark_inaccessible(514, 2);
 for(int i = 4; i> 0; i--)
  {
 unsigned long process_mem_pool_info_frame2 = kernel_mem_pool.get_frames((i%4)+1);
Console::puts("\n alloc :\n");
Console::putui(process_mem_pool_info_frame2);
}
ContFramePool::release_frames(516);
unsigned long process_mem_pool_info_frame1 = kernel_mem_pool.get_frames(5);
Console::puts("\n ALLOC :\n");
Console::putui(process_mem_pool_info_frame1);
….
….
```

<u>OUTPUT :</u>

The memory allocation could be represented as,



	Kernel_Info														
	512	513	514	515	516	....	519	520	....	522	523	524			
	Kernel_Info		I	I											
	512	513	514	515	516	....	519	520	....	522	523	524			
	Kernel_Info	A	I	I											
	512	513	514	515	516	....	519	520	....	522	523	524			
	Kernel_Info	A	I	I	H	A	A	H	A	A	H	A			
	512	513	514	515	516	....	519	520	....	522	523	524			
	Kernel_Info	A	I	I				H	A	A	H	A			
	512	513	514	515	516	....	519	520	....	522	523	524			
	Kernel_Info	A	I	I				H	A	A	H	A	H		A
	512	513	514	515	516	....	519	520	....	522	523	524	525	.....	529

<u>SCENARIO: 2 : TEST RESULT : PASSED</u>

Description: To check the error scenario where requested frames = nFreeFrames but aren't continuous. Allocating a process pool of 8 frames.

- GetFrames(2) : 1024, 1025
- Mark_Inaccessible(1027 ,4) : 1027,1028,1029,1030 marked inaccessible.
- Remainin Free Frames : 1026 , 1031 : not continuous
- Get_Frames(2) : Returns NOT CONTINUOUS ERROR

Code SNIPPET part of Kernel.C

```
…
…

  ContFramePool process_mem_pool(PROCESS_POOL_START_FRAME, 8,
                                                    process_mem_pool_info_frame, n_info_frame s);
    unsigned long process_mem_pool_info_frame1 = process_mem_pool.get_frames(2);
    Console::puts("\n SECOND :\n");
    Console::putui(process_mem_pool_info_frame1);
     process_mem_pool.mark_inaccessible(1027, 4)
     Console::puts("\n 1st mark :\n");
     process_mem_pool_info_frame1 = process_mem_pool.get_frames(2);
    Console::puts("\n SECOND :\n");
    Console::putui(process_mem_pool_info_frame1);
….
…..
```

OUTPUT:

```
NO CONTINUOUS FREE FRAMES AVAILABLE
ASSERT : 218
```