MP 2: Frame Manager

Pranav Anantharam

UIN: 734003886

CSCE611: Operating System

Assigned Tasks

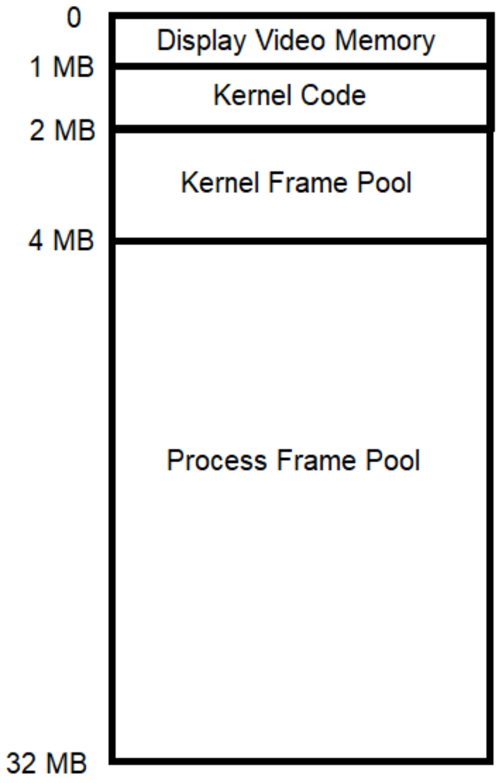
Main: Completed

System Design

The goal of the machine problem is to implement a frame manager, which manages the allocation of frames (physical pages). The frame manager is responsible for the allocation and release of frames, i.e., it needs to keep track of which pages are being used and which ones are free. The memory layout of the machine is given below:

- Total memory in the machine = 32 MB
- Memory reserved for Kernel = 4 MB
- Process memory pool = 28 MB
- Frame size = 4 KB
- Inaccessible memory region = 15 MB – 16 MB

Memory Layout Diagram:



Each frame pool separately manages a collection of frames that can be allocated and released. When the kernel needs frames, it retrieves them from the kernel frame pool, which is located between 2 MB and 4 MB. When a process needs frames, it retrieves them from the process frame pool, which is located beyond 4 MB.

To implement a frame pool, I used a bitmap describing the state of each frame in the frame pool. To efficiently track the state of each frame, I used 2 bits to represent one frame. The bit representation is given below:

Bit Representation	Frame State
00	Free
01	Used / Allocated
11	Head of Sequence (Allocated)

Kernel Pool Frames Location: 512 – 1024 → 512 frames in kernel pool

Process Pool Frames Location: 1024 - 8192 → 7168 frames in process pool

If we take 2 bits to store the state of each frame in the bitmap, then:

Kernel pool bitmap size will be 1024 bits or 128 bytes, which will fit in 1 frame (since frame size = 4 KB)

Process pool bitmap size will be 7168 bits or 896 bytes, which will also fit in 1 frame (since frame size = 4KB)

To manage multiple frame pools (kernel and process memory) which have their individual bitmaps to track the state of frames, I implemented a static linked list of frame pools. In this manner, we can traverse the linked list to access each frame pool and manage the frame pool bitmaps, get frames and release frames. The linked list approach was taken since the release frames functionality was static and is common across all the frame pools.

List of files modified

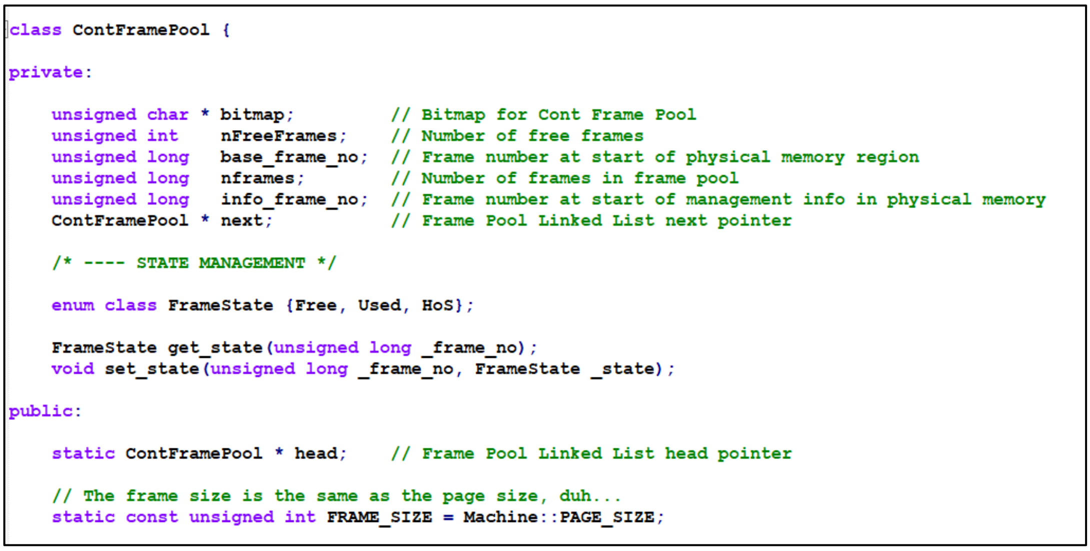
1. cont_frame_pool.H
2. cont_frame_pool.C

Code Description

1. **cont_frame_pool.H : class ContFramePool :**

   In class ContFramePool, data structures for the frame pools were declared:
   a) unsigned char * bitmap - Bitmap for Cont Frame Pool
   b) unsigned int nFreeFrames - Number of free frames
   c) unsigned long base_frame_no - Frame number at start of physical memory region
   d) unsigned long nframes - Number of frames in frame pool
   e) unsigned long info_frame_no - Frame number at start of management info in physical memory

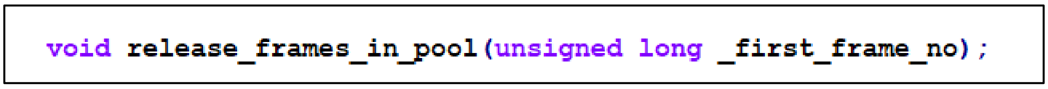
   Frame Pool Linked List pointers are also declared the in the class:
   a) static ContFramePool * head - Frame Pool Linked List head pointer
   b) ContFramePool * next - Frame Pool Linked List next pointer



```
class ContFramePool {

private:

    unsigned char * bitmap;         // Bitmap for Cont Frame Pool
    unsigned int    nFreeFrames;    // Number of free frames
    unsigned long   base_frame_no;  // Frame number at start of physical memory region
    unsigned long   nframes;        // Number of frames in frame pool
    unsigned long   info_frame_no;  // Frame number at start of management info in physical memory
    ContFramePool * next;           // Frame Pool Linked List next pointer

    /* ---- STATE MANAGEMENT */

    enum class FrameState {Free, Used, HoS};

    FrameState get_state(unsigned long _frame_no);
    void set_state(unsigned long _frame_no, FrameState _state);

public:

    static ContFramePool * head;    // Frame Pool Linked List head pointer

    // The frame size is the same as the page size, duh...
    static const unsigned int FRAME_SIZE = Machine::PAGE_SIZE;
```

2. cont_frame_pool.H : release_frames_in_pool() :

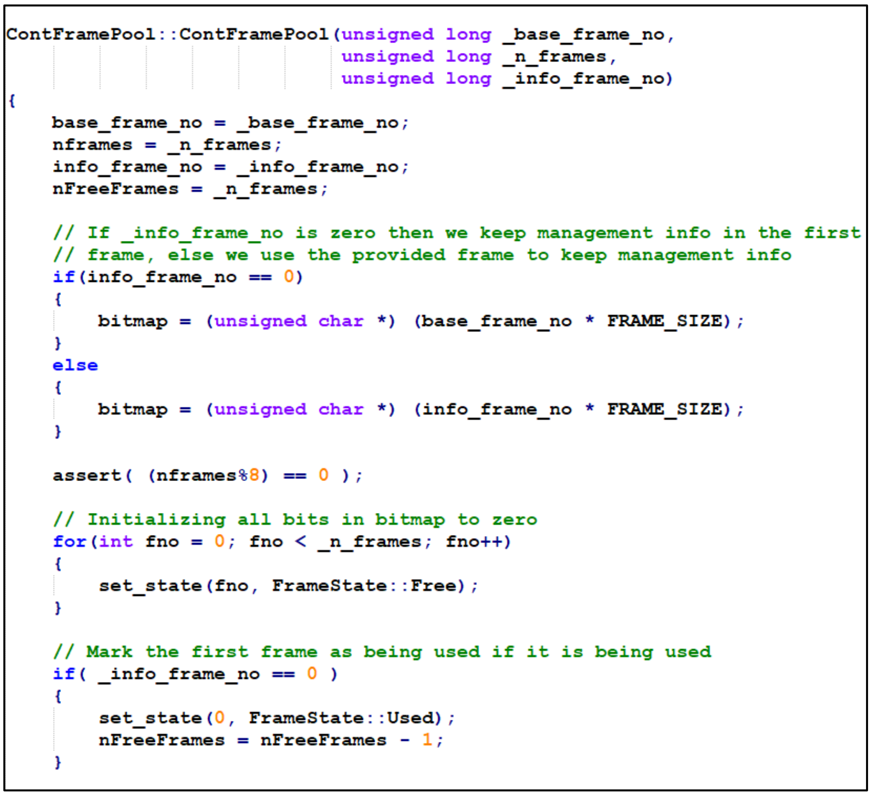
Since release_frames() is a static function, I defined the prototype for a non-static internal version of release_frames() which can be called to release the frames of the correct frame pool.



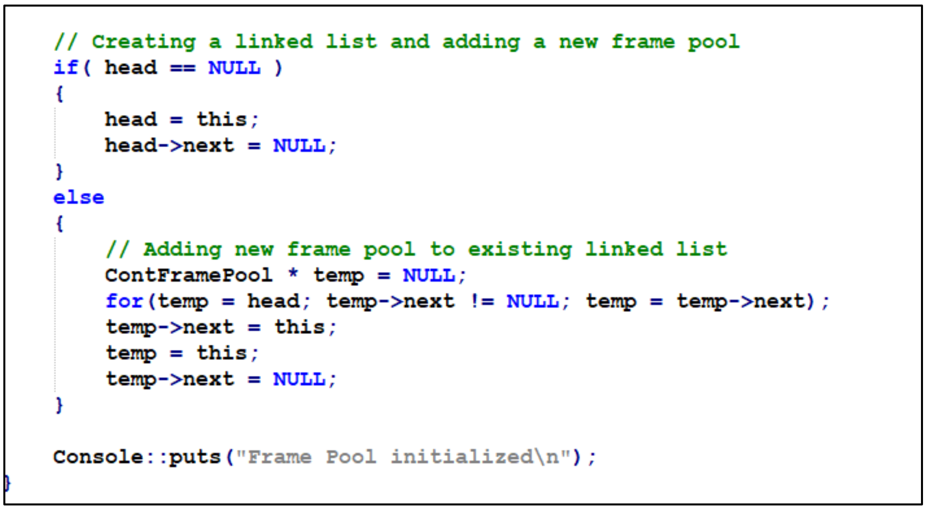
```
void release_frames_in_pool(unsigned long _first_frame_no);
```

3. cont_frame_pool.C : constructor ContFramePool():

This method is the constructor for class ContFramePool. It initializes all the data structures needed for the management of the frame pool. It initializes all the class data members to the values passed in the arguments to the constructor. All the bits in the frame pool bitmap are initialized to 'Free' state. A linked list is created (if not created earlier) or a new frame pool is added to the linked list.



```
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;
    info_frame_no = _info_frame_no;
    nFreeFrames = _n_frames;

    // If _info_frame_no is zero then we keep management info in the first
    // frame, else we use the provided frame to keep management info
    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 );

    // Initializing all bits in bitmap to zero
    for(int fno = 0; fno < _n_frames; fno++)
    {
        set_state(fno, FrameState::Free);
    }

    // Mark the first frame as being used if it is being used
    if( _info_frame_no == 0 )
    {
        set_state(0, FrameState::Used);
        nFreeFrames = nFreeFrames - 1;
    }
```

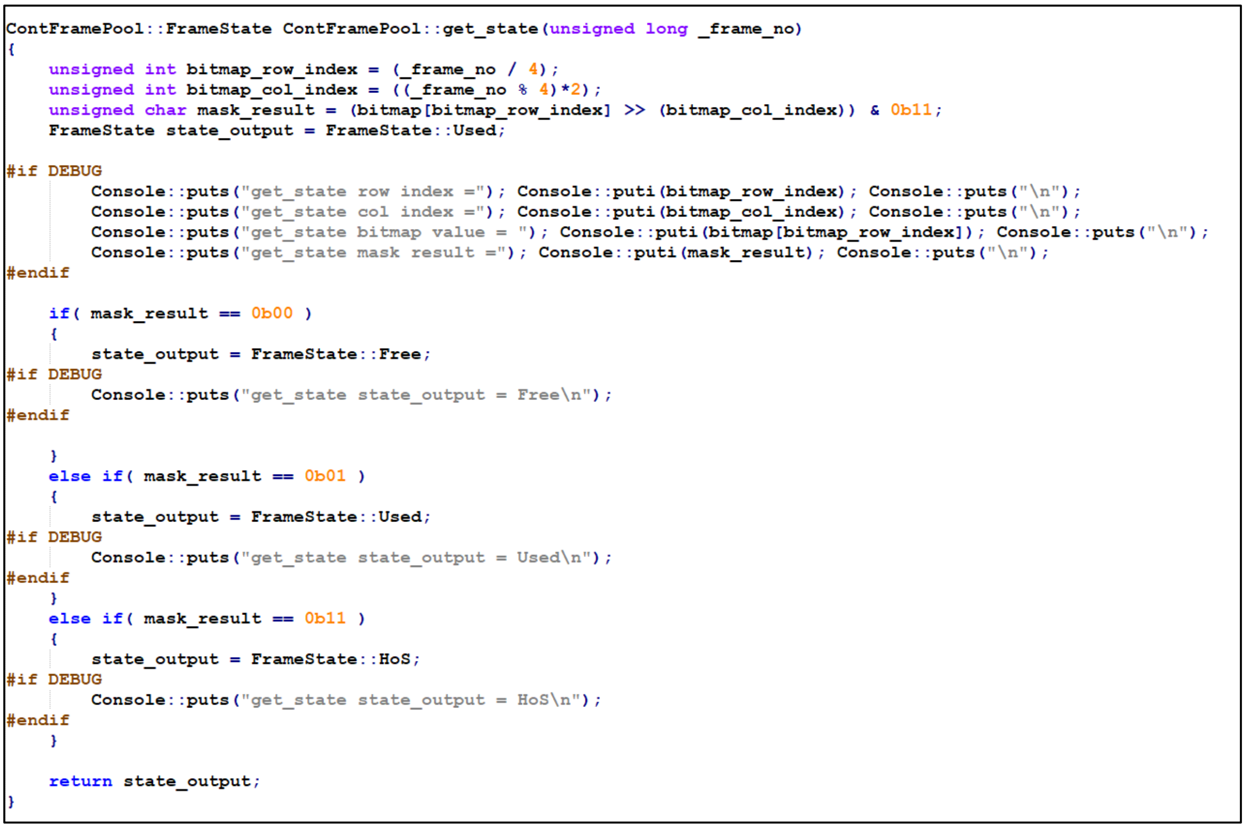


```
    // Creating a linked list and adding a new frame pool
    if( head == NULL )
    {
        head = this;
        head->next = NULL;
    }
    else
    {
        // Adding new frame pool to existing linked list
        ContFramePool * temp = NULL;
        for(temp = head; temp->next != NULL; temp = temp->next);
        temp->next = this;
        temp = this;
        temp->next = NULL;
    }

    Console::puts("Frame Pool initialized\n");
}
```

4. **cont_frame_pool.C : get_state() :**

This method is used to obtain the state of a frame by using the _frame_no argument. Assuming the bitmap was a set of rows and columns, we can obtain the row index and column index of the 2 bits that are used to represent the desired frame. By performing bitwise operations (right shift operation), we can extract the desired 2 bits from the bitmap and check the value to find out the state of the frame allocation. Example of bitmap row and column index calculation:

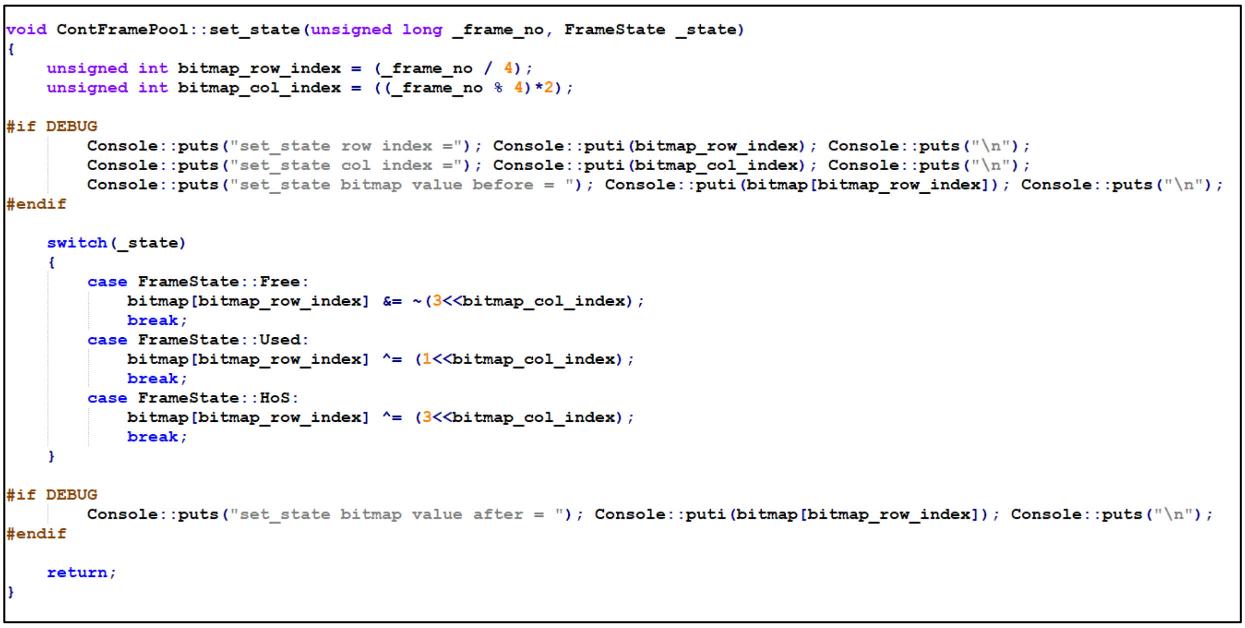
To calculate the location of the 2 bits used to describe the frame state for frame number 10:
Bitmap row index = (11 / 4) = 2 (Integer Division) → 3rd row
Bitmap column index = (11 % 4) * 2 = 6 → 7th bit
Hence, the desired frame state bits are the 7th and 8th bits of the 3rd bitmap element (bitmap[2]).



```
ContFramePool::FrameState ContFramePool::get_state(unsigned long _frame_no)
{
    unsigned int bitmap_row_index = (_frame_no / 4);
    unsigned int bitmap_col_index = ((_frame_no % 4)*2);
    unsigned char mask_result = (bitmap[bitmap_row_index] >> (bitmap_col_index)) & 0b11;
    FrameState state_output = FrameState::Used;

#if DEBUG
        Console::puts("get_state row index ="); Console::puti(bitmap_row_index); Console::puts("\n");
        Console::puts("get_state col index ="); Console::puti(bitmap_col_index); Console::puts("\n");
        Console::puts("get_state bitmap value = "); Console::puti(bitmap[bitmap_row_index]); Console::puts("\n");
        Console::puts("get_state mask result ="); Console::puti(mask_result); Console::puts("\n");
#endif

    if( mask_result == 0b00 )
    {
        state_output = FrameState::Free;
#if DEBUG
        Console::puts("get_state state_output = Free\n");
#endif

    }
    else if( mask_result == 0b01 )
    {
        state_output = FrameState::Used;
#if DEBUG
        Console::puts("get_state state_output = Used\n");
#endif
    }
    else if( mask_result == 0b11 )
    {
        state_output = FrameState::HoS;
#if DEBUG
        Console::puts("get_state state_output = HoS\n");
#endif
    }

    return state_output;
}
```

5. **cont_frame_pool.C : set_state() :**

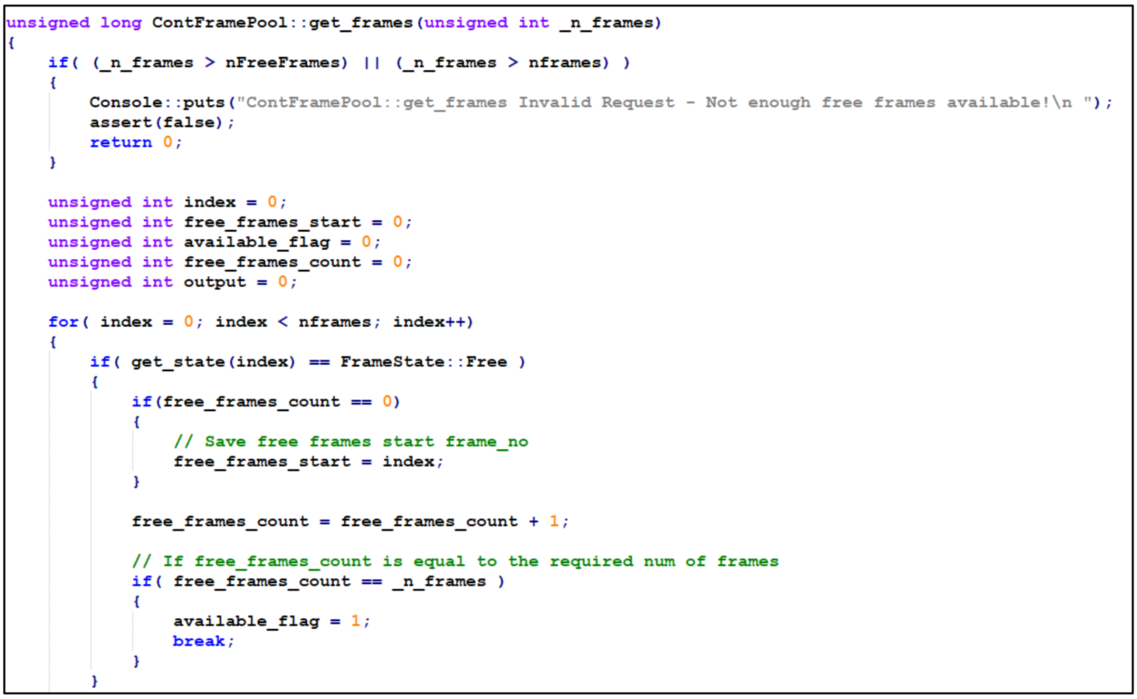
This method is used to set the state of a particular frame. The state is set to either Free, Used or Head of Sequence (HoS) based on the _state argument passed to the method. Once again, we obtain the row index and column index of the bits for _frame_no and perform bitwise operations (AND, NOT, XOR operations) to set the desired state.



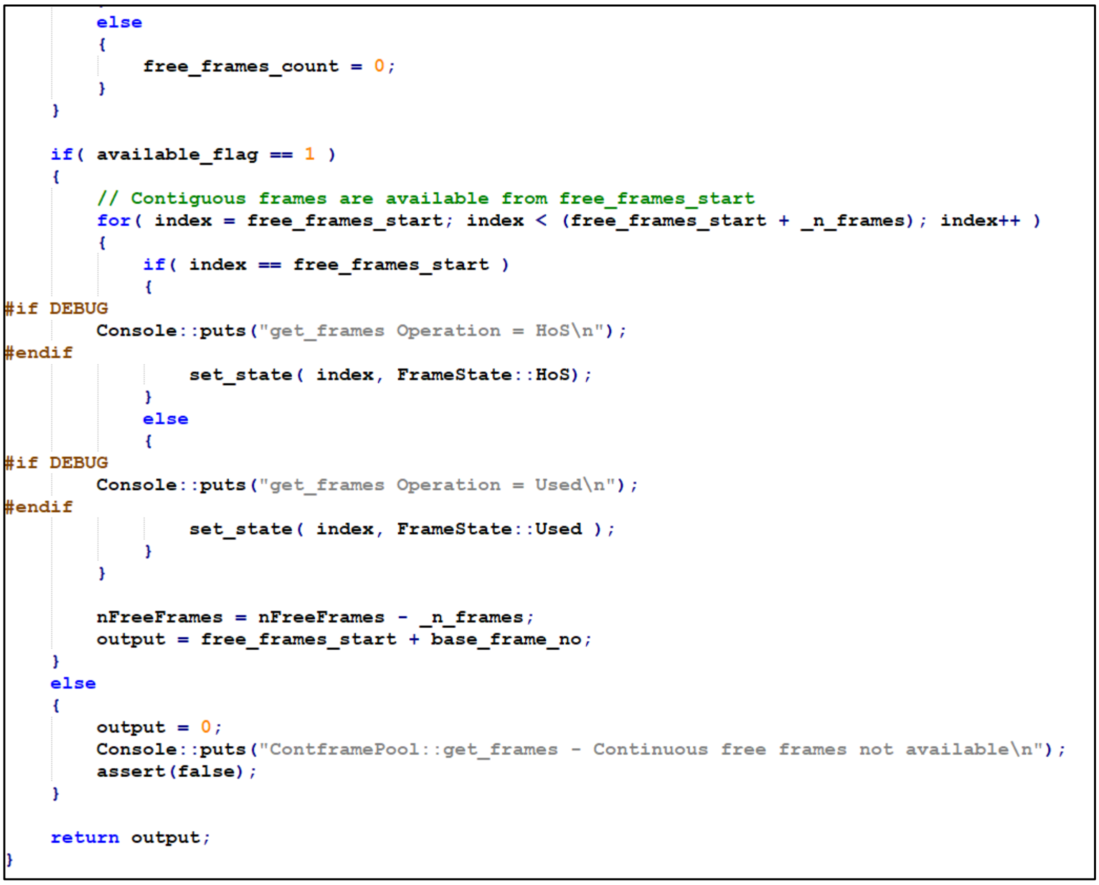
```
void ContFramePool::set_state(unsigned long _frame_no, FrameState _state)
{
    unsigned int bitmap_row_index = (_frame_no / 4);
    unsigned int bitmap_col_index = ((_frame_no % 4)*2);

#if DEBUG
        Console::puts("set_state row index ="); Console::puti(bitmap_row_index); Console::puts("\n");
        Console::puts("set_state col index ="); Console::puti(bitmap_col_index); Console::puts("\n");
        Console::puts("set_state bitmap value before = "); Console::puti(bitmap[bitmap_row_index]); Console::puts("\n");
#endif

    switch(_state)
    {
        case FrameState::Free:
            bitmap[bitmap_row_index] &= ~(3<<bitmap_col_index);
            break;
        case FrameState::Used:
            bitmap[bitmap_row_index] ^= (1<<bitmap_col_index);
            break;
        case FrameState::HoS:
            bitmap[bitmap_row_index] ^= (3<<bitmap_col_index);
            break;
    }

#if DEBUG
        Console::puts("set_state bitmap value after = "); Console::puti(bitmap[bitmap_row_index]); Console::puts("\n");
#endif

    return;
}
```

6. **cont_frame_pool.C : get_frames() :**

This method is used to allocate a number of contiguous frames from the frame pool. A check is performed to verify that enough free frames are available to be allocated for the get_frames request. Next, using a loop, we check if there exists a set of '_n_frames' contiguous frames available in the frame pool. If this set of frames are available, then using a loop, we set the state of the first frame to Head of Sequence (HoS) and the remaining ('_n_frames' – 1) frame states to Used/Allocated. If successful, the frame number of the first frame is returned. If a failure occurs, the value 0 is returned. The search algorithm is optimized to run in O(n) time, i.e., the length of the frame pools.



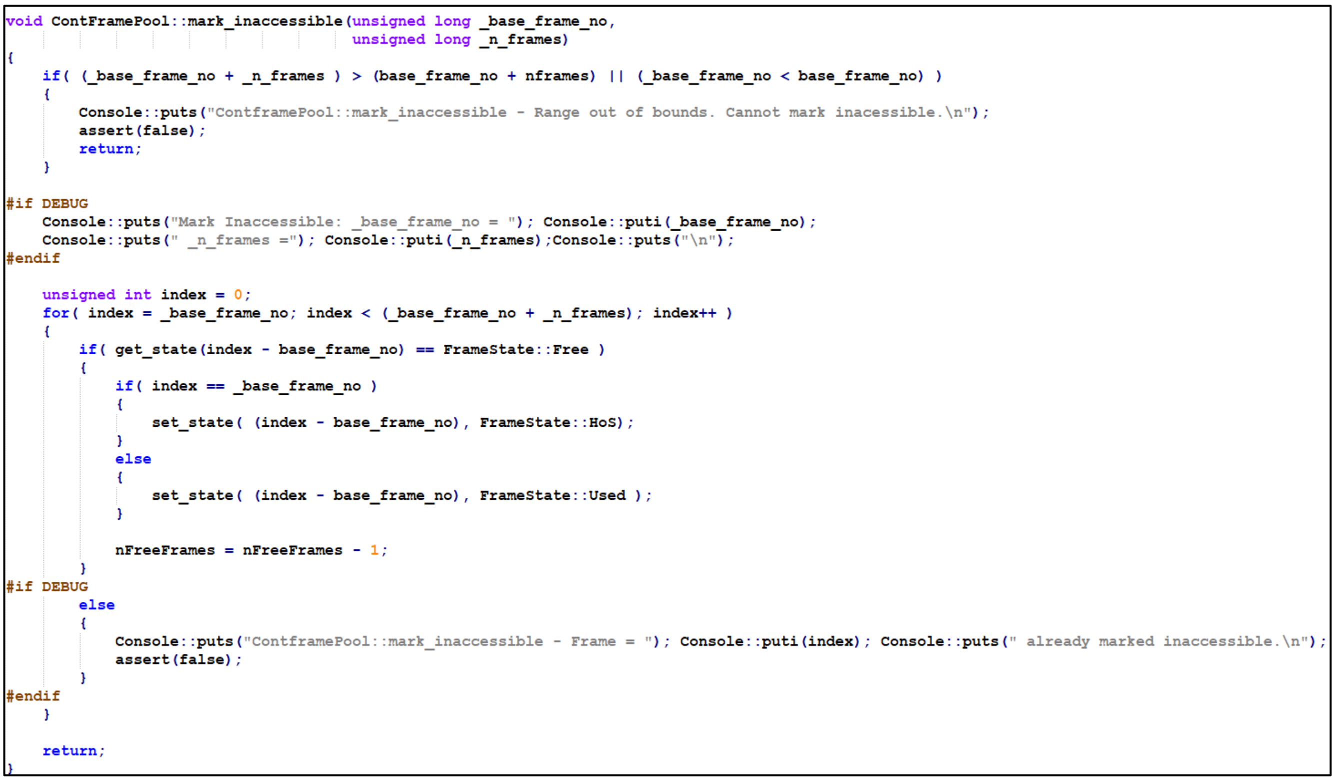
```
unsigned long ContFramePool::get_frames(unsigned int _n_frames)
{
    if( (_n_frames > nFreeFrames) || (_n_frames > nframes) )
    {
        Console::puts("ContFramePool::get_frames Invalid Request - Not enough free frames available!\n ");
        assert(false);
        return 0;
    }

    unsigned int index = 0;
    unsigned int free_frames_start = 0;
    unsigned int available_flag = 0;
    unsigned int free_frames_count = 0;
    unsigned int output = 0;

    for( index = 0; index < nframes; index++)
    {
        if( get_state(index) == FrameState::Free )
        {
            if(free_frames_count == 0)
            {
                // Save free frames start frame_no
                free_frames_start = index;
            }

            free_frames_count = free_frames_count + 1;

            // If free_frames_count is equal to the required num of frames
            if( free_frames_count == _n_frames )
            {
                available_flag = 1;
                break;
            }
        }
```



```
        else
        {
            free_frames_count = 0;
        }
    }

    if( available_flag == 1 )
    {
        // Contiguous frames are available from free_frames_start
        for( index = free_frames_start; index < (free_frames_start + _n_frames); index++ )
        {
            if( index == free_frames_start )
            {
#if DEBUG
        Console::puts("get_frames Operation = HoS\n");
#endif
                set_state( index, FrameState::HoS);
            }
            else
            {
#if DEBUG
        Console::puts("get_frames Operation = Used\n");
#endif
                set_state( index, FrameState::Used );
            }
        }

        nFreeFrames = nFreeFrames - _n_frames;
        output = free_frames_start + base_frame_no;
    }
    else
    {
        output = 0;
        Console::puts("ContframePool::get_frames - Continuous free frames not available\n");
        assert(false);
    }

    return output;
}
```

7. cont_frame_pool.C : mark_inaccessible() :

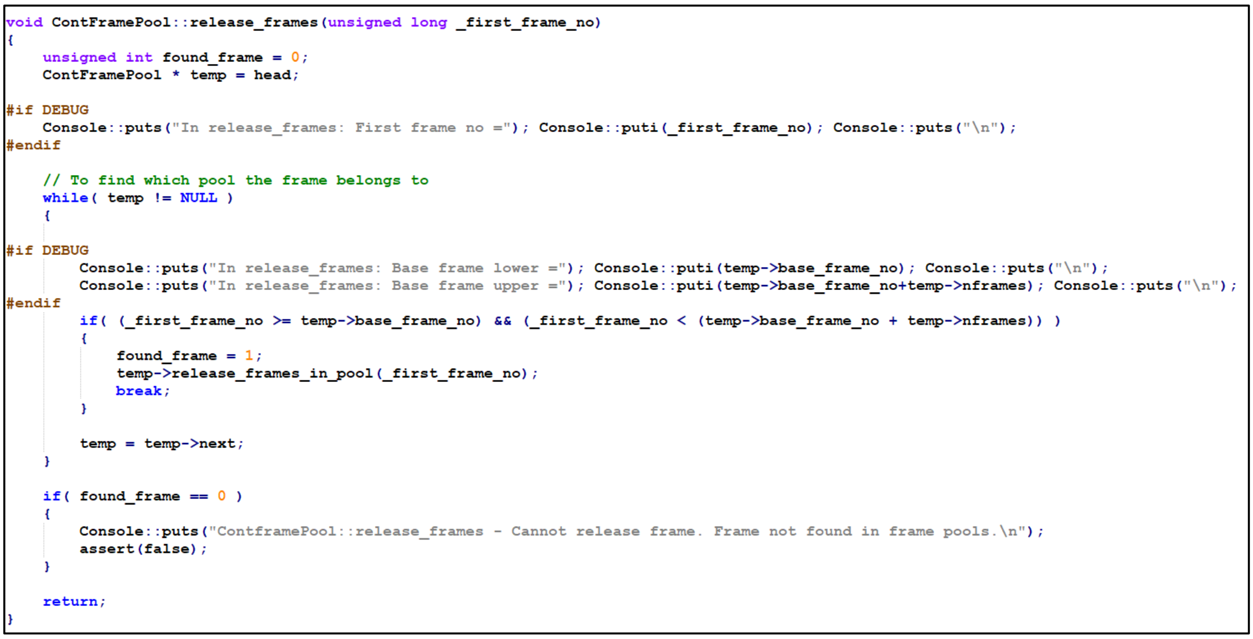
This method is used to mark a contiguous sequence of physical frames as inaccessible. First, a check is performed to verify that the frames to be marked as inaccessible are within the range of the frame pool. To mark frames as inaccessible, we use a loop to iterate through the frame numbers and use the ‘set_state’ method to set the state of the frame as Head of Sequence for the first frame and the remaining frame states as Used in the frame pool bitmap.



```
void ContFramePool::mark_inaccessible(unsigned long _base_frame_no,
                                      unsigned long _n_frames)
{
    if( (_base_frame_no + _n_frames ) > (base_frame_no + nframes) || (_base_frame_no < base_frame_no) )
    {
        Console::puts("ContframePool::mark_inaccessible - Range out of bounds. Cannot mark inacessible.\n");
        assert(false);
        return;
    }

#if DEBUG
    Console::puts("Mark Inaccessible: _base_frame_no = "); Console::puti(_base_frame_no);
    Console::puts(" _n_frames ="); Console::puti(_n_frames);Console::puts("\n");
#endif

    unsigned int index = 0;
    for( index = _base_frame_no; index < (_base_frame_no + _n_frames); index++ )
    {
        if( get_state(index - base_frame_no) == FrameState::Free )
        {
            if( index == _base_frame_no )
            {
                set_state( (index - base_frame_no), FrameState::HoS);
            }
            else
            {
                set_state( (index - base_frame_no), FrameState::Used );
            }

            nFreeFrames = nFreeFrames - 1;
        }
#if DEBUG
        else
        {
            Console::puts("ContframePool::mark_inaccessible - Frame = "); Console::puti(index); Console::puts(" already marked inaccessible.\n");
            assert(false);
        }
#endif
    }

    return;
}
```

8. cont_frame_pool.C : release_frames() :

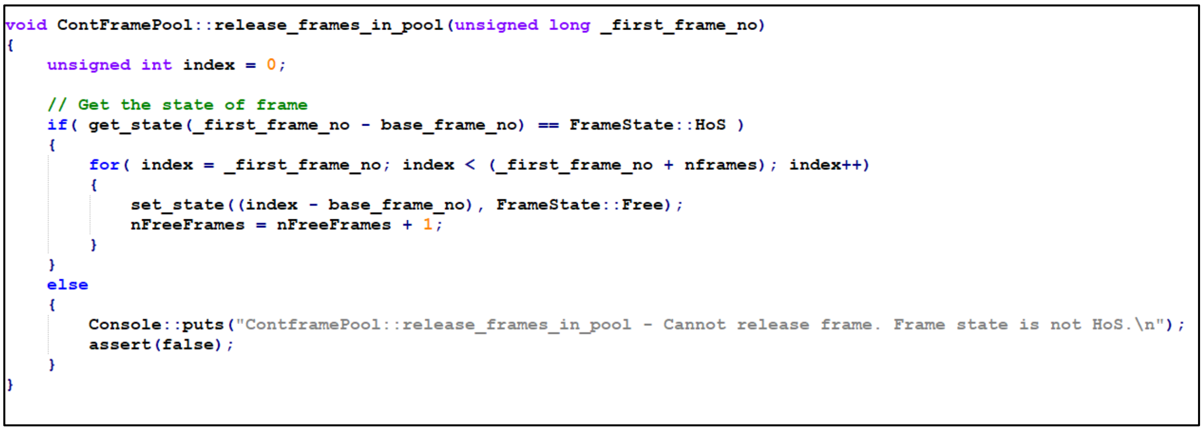
This method is used to release a previously allocated contiguous sequence of frames back to its frame pool. The frame sequence is identified by the number of the first frame. Using a loop, we iterate through the linked list and check whether the frame number exists within the range of frame numbers of the frame pool. If the frame number exists in the frame pool, we have identified the frame pool the frame belongs to. Next, we call the 'release_frames_in_pool' method to release all the frames of the identified frame pool.



```
void ContFramePool::release_frames(unsigned long _first_frame_no)
{
    unsigned int found_frame = 0;
    ContFramePool * temp = head;

#if DEBUG
    Console::puts("In release_frames: First frame no ="); Console::puti(_first_frame_no); Console::puts("\n");
#endif

    // To find which pool the frame belongs to
    while( temp != NULL )
    {

#if DEBUG
        Console::puts("In release_frames: Base frame lower ="); Console::puti(temp->base_frame_no); Console::puts("\n");
        Console::puts("In release_frames: Base frame upper ="); Console::puti(temp->base_frame_no+temp->nframes); Console::puts("\n");
#endif
        if( (_first_frame_no >= temp->base_frame_no) && (_first_frame_no < (temp->base_frame_no + temp->nframes)) )
        {
            found_frame = 1;
            temp->release_frames_in_pool(_first_frame_no);
            break;
        }

        temp = temp->next;
    }

    if( found_frame == 0 )
    {
        Console::puts("ContframePool::release_frames - Cannot release frame. Frame not found in frame pools.\n");
        assert(false);
    }

    return;
}
```

9. cont_frame_pool.C : release_frames_in_pool() :

This method is used to release all the frames of a particular frame pool. We first verify whether the bitmap state of the first frame is Head of Sequence. Next, using a loop, we iterate through the frame numbers and set the state of the frame to 'Free'. Also, the variable nFreeFrames is incremented.

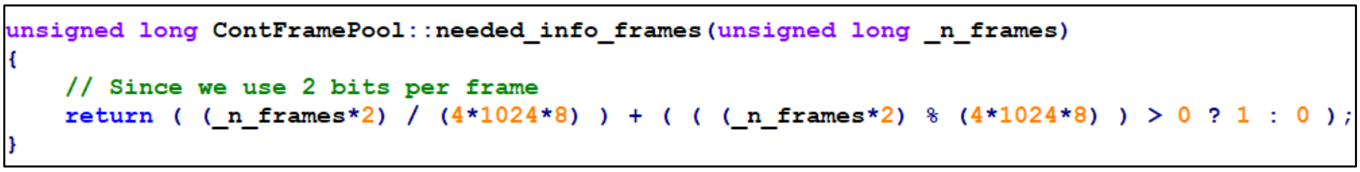


```
void ContFramePool::release_frames_in_pool(unsigned long _first_frame_no)
{
    unsigned int index = 0;

    // Get the state of frame
    if( get_state(_first_frame_no - base_frame_no) == FrameState::HoS )
    {
        for( index = _first_frame_no; index < (_first_frame_no + nframes); index++)
        {
            set_state((index - base_frame_no), FrameState::Free);
            nFreeFrames = nFreeFrames + 1;
        }
    }
    else
    {
        Console::puts("ContframePool::release_frames_in_pool - Cannot release frame. Frame state is not HoS.\n");
        assert(false);
    }
}
```

10. cont_frame_pool.C : needed_info_frames() :

This method returns the number of frames needed to manage a frame pool of size _n_frames.
For frame size = 4096 bytes and a bitmap with two bits per frame, we can calculate the number of info frames needed to be:
(_n_frames*2) / 32k + ((_n_frames*2) % 32k > 0 ? 1 : 0)

For example, for the process pool, we can perform the following calculation:
Number of frames in 28 MB process memory pool = 7168 frames
Considering 2 bits per frame, then
Number of info frames needed = ((7168 * 2) / 32000) + (((7168*2) % 32000) > 0 ? 1: 0) = 1 Frame



```
unsigned long ContFramePool::needed_info_frames(unsigned long _n_frames)
{
    // Since we use 2 bits per frame
    return ( (_n_frames*2) / (4*1024*8) ) + ( ( (_n_frames*2) % (4*1024*8) ) > 0 ? 1 : 0 );
}
```

Testing

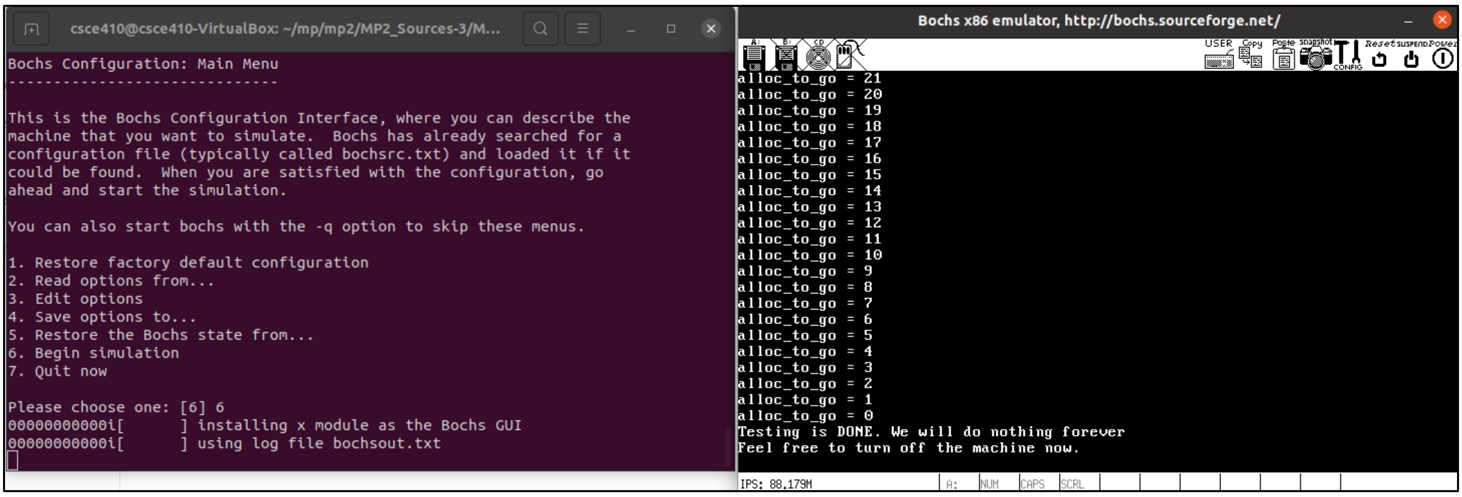
Serial No.	Testcase	Desired Result	Actual Result	Description
1	Create kernel frame pool	Kernel frame pool is created	PASS Kernel frame was created successfully	Creating a single kernel pool of size 2 MB
2	Create process frame pool	Process frame pool is created	PASS Process frame pool was created successfully	Creating a single process pool of size 28 MB
3	Create multiple process frame pools	Multiple process frame pools are created	PASS Multiple process pools were created	Kernel.C was edited to create 2 process pools of size 14 MB each
4	Get frames : Number of frames requested is less than nFreeFrames	Frames are allocated	PASS Frames were allocated successfully	This test was performed for both kernel and process pools
5	Get frames : Number of frames requested is greater than nFreeFrames	Frames are not allocated – get_frames fails	PASS Frames were not allocated (insufficient memory)	This test was performed for both kernel and process pools.
6	Get frames : Contiguous frames not available in frame pool	Frames are not allocated – get_frames fails	PASS Frames were not allocated (contiguous memory not available)	
7	Release frames : Frame does not belong to any pool	Frames are not released – release_frames fails	PASS Frames are not released	
8	Release frames : Frame belongs to kernel pool	Kernel pool frames are released	PASS Kernel pool frames are released	Tested using test_memory method
9	Release frames : Frame belongs to process pool	Process pool frames are released	PASS Process pool frames are released	Tested using test_memory method

10	Get frames and release frames in succession for kernel pool	Frames are allocated and released for kernel pool	PASS Frames were allocated and released for kernel pool	Tested using test_memory method
11	Get frames and release frames in succession for process pool	Frames are allocated and released for process pool	PASS Frames are allocated and released for process pool	Tested using test_memory method
12	Mark Inaccessible : Frame number is within range	Frames are marked as inaccessible	PASS Frames are marked as inaccessible	
13	Mark Inaccessible : Frame number is not within range	Frames are not marked inaccessible – method returns fail	PASS Frames are not marked inaccessible	
14	Mark Inaccessible : Frame number already marked inaccessible	Frames are not inaccessible – method returns fail	PASS Frames are not marked inaccessible	
15	Needed Info frames : Print out number of info frames required	Number of info frames required is returned	PASS Number of info frames required was returned	

The ‘test_memory’ method provided in kernel.C was used to test the get frames and release frames functionality. The kernel.C file was modified to perform the above testcases and call each of the implemented methods (get_frames, release_frames, mark_inaccessible, needed_info_frames) and verify the output. I have tested the basic functionality for all implemented methods and their basic edge cases.

Outputs:

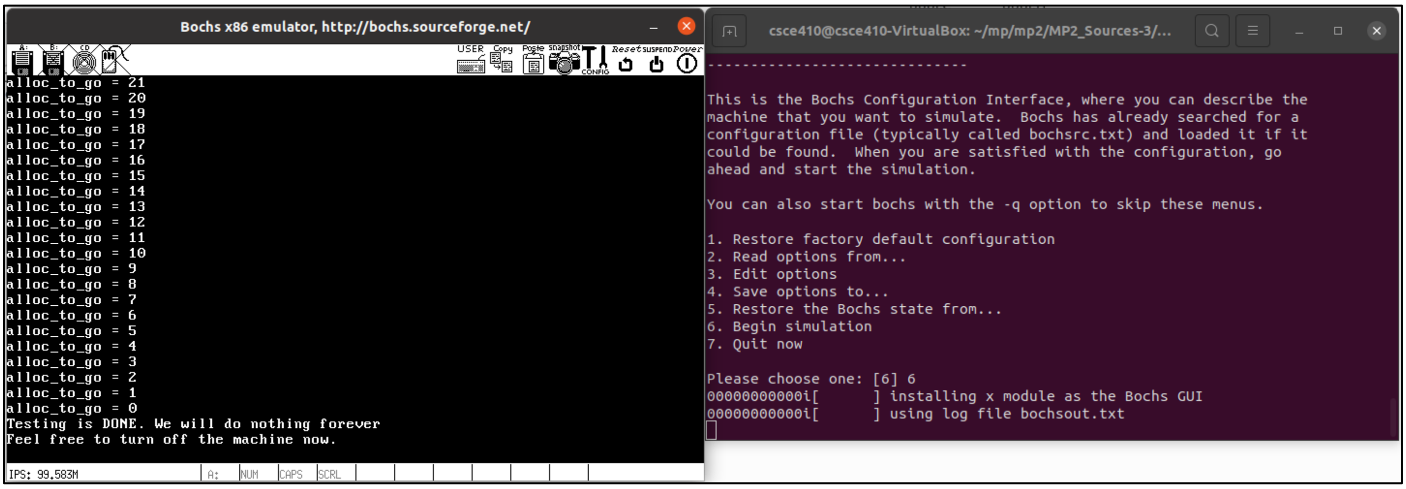
1. test_memory(&kernel_mem_pool, 32);


csce410@csce410-VirtualBox: ~/mp/mp2/MP2_Sources-3/M...
Bochs x86 emulator, http://bochs.sourceforge.net/
USER
Copy
Paste
Snapshot
CONFIG
Reset
SUSPEND
Power
IPS: 88.179M
A:
NUM
CAPS
SCRL

```
Bochs Configuration: Main Menu
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found.  When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [6] 6
00000000000i[      ] installing x module as the Bochs GUI
00000000000i[      ] using log file bochsout.txt
```

```
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.
```

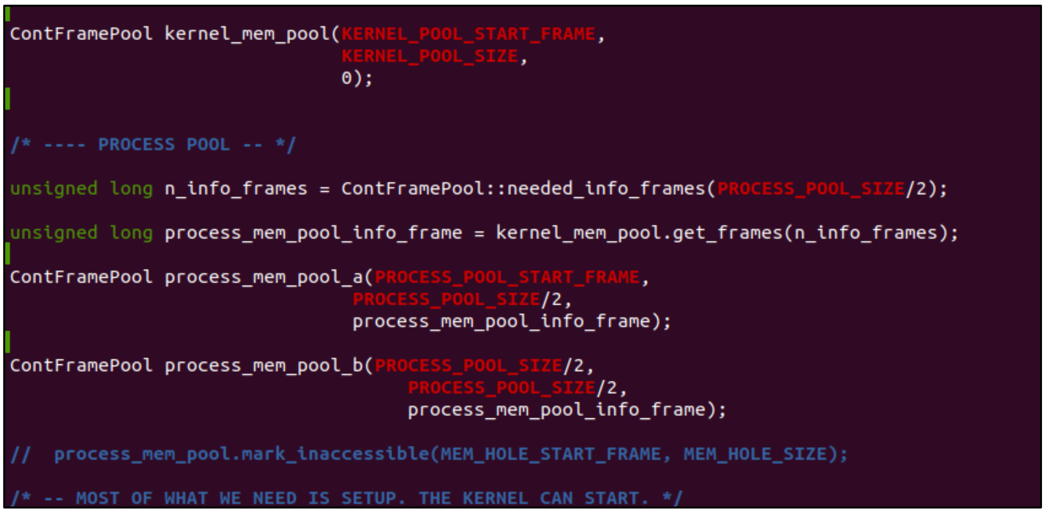
2. test_memory(&process_mem_pool, 32);


Bochs x86 emulator, http://bochs.sourceforge.net/
USER
Copy
Paste
snapshot
CONFIG
Reset
SUSPEND
Power
IPS: 99.583M
A:
NUM
CAPS
SCRL
csce410@csce410-VirtualBox: ~/mp/mp2/MP2_Sources-3/...

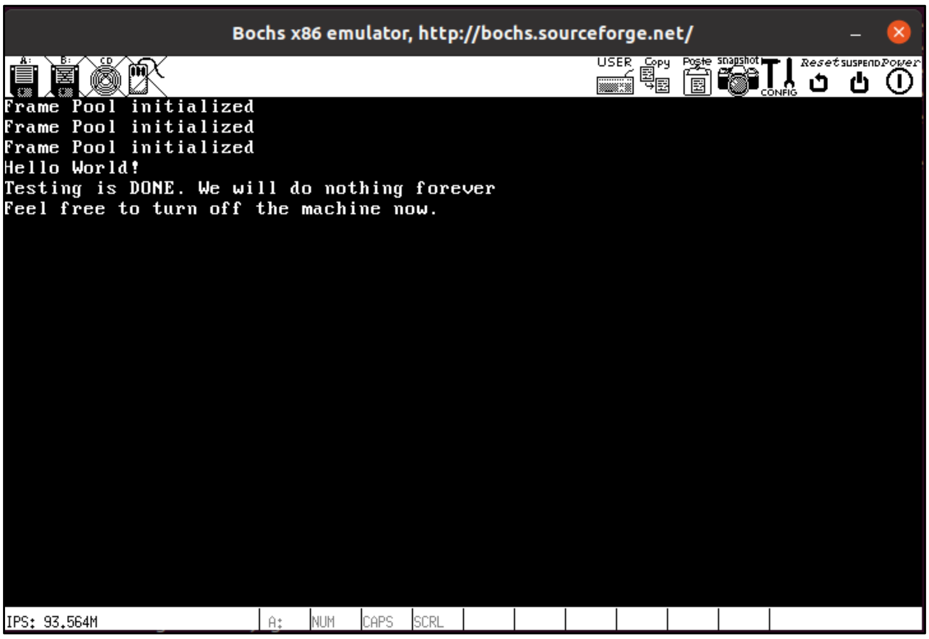
```
alloc_to_go = 21
alloc_to_go = 20
alloc_to_go = 19
alloc_to_go = 18
alloc_to_go = 17
alloc_to_go = 16
alloc_to_go = 15
alloc_to_go = 14
alloc_to_go = 13
alloc_to_go = 12
alloc_to_go = 11
alloc_to_go = 10
alloc_to_go = 9
alloc_to_go = 8
alloc_to_go = 7
alloc_to_go = 6
alloc_to_go = 5
alloc_to_go = 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.
```

```
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found.  When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [6] 6
00000000000i[      ] installing x module as the Bochs GUI
00000000000i[      ] using log file bochsout.txt
```

3. Modifications made to kernel.C to perform Testcase No. 3:
Two process pools of size 14 MB were successfully created.



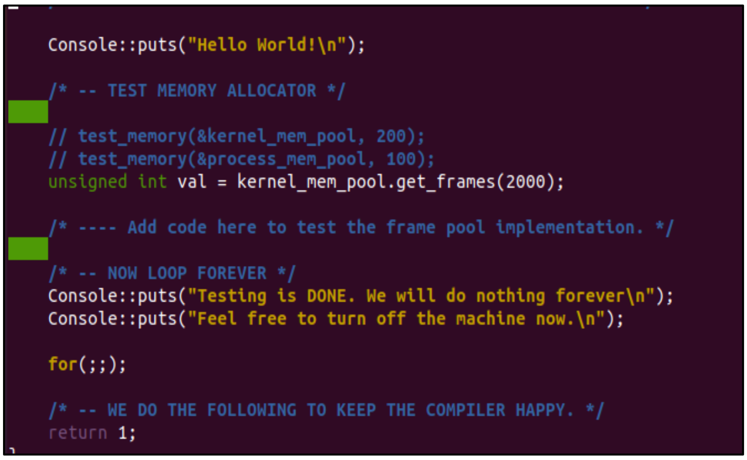
```
ContFramePool kernel_mem_pool(KERNEL_POOL_START_FRAME,
                              KERNEL_POOL_SIZE,
                              0);

/* ---- PROCESS POOL -- */

unsigned long n_info_frames = ContFramePool::needed_info_frames(PROCESS_POOL_SIZE/2);

unsigned long process_mem_pool_info_frame = kernel_mem_pool.get_frames(n_info_frames);

ContFramePool process_mem_pool_a(PROCESS_POOL_START_FRAME,
                               PROCESS_POOL_SIZE/2,
                               process_mem_pool_info_frame);

ContFramePool process_mem_pool_b(PROCESS_POOL_SIZE/2,
                                    PROCESS_POOL_SIZE/2,
                                    process_mem_pool_info_frame);

//  process_mem_pool.mark_inaccessible(MEM_HOLE_START_FRAME, MEM_HOLE_SIZE);

/* -- MOST OF WHAT WE NEED IS SETUP. THE KERNEL CAN START. */
```



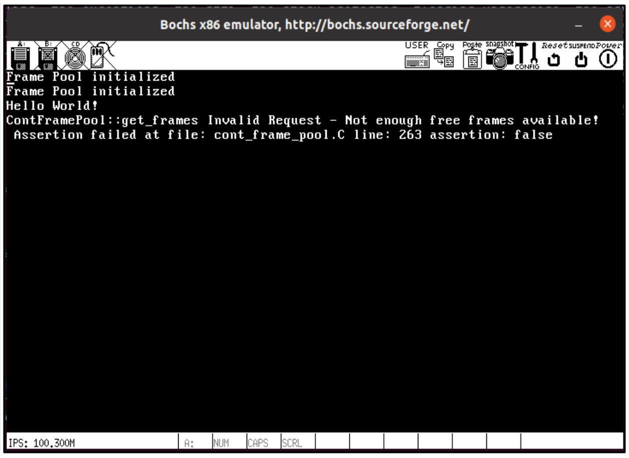
```
Frame Pool initialized
Frame Pool initialized
Frame Pool initialized
Hello World!
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.
```

4. Modifications made to kernel.C to perform Testcase No. 5:

unsigned int val = kernel_mem_pool.get_frames(2000);

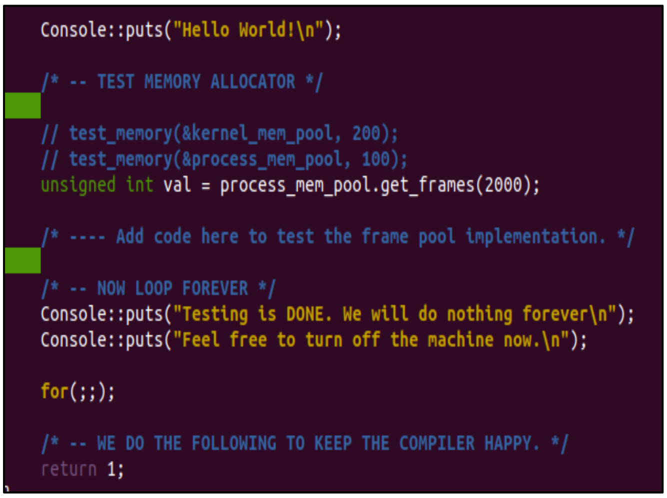


```
Console::puts("Hello World!\n");

/* -- TEST MEMORY ALLOCATOR */

// test_memory(&kernel_mem_pool, 200);
// test_memory(&process_mem_pool, 100);
unsigned int val = kernel_mem_pool.get_frames(2000);

/* ---- Add code here to test the frame pool implementation. */

/* -- NOW LOOP FOREVER */
Console::puts("Testing is DONE. We will do nothing forever\n");
Console::puts("Feel free to turn off the machine now.\n");

for(;;);

/* -- WE DO THE FOLLOWING TO KEEP THE COMPILER HAPPY. */
return 1;
```

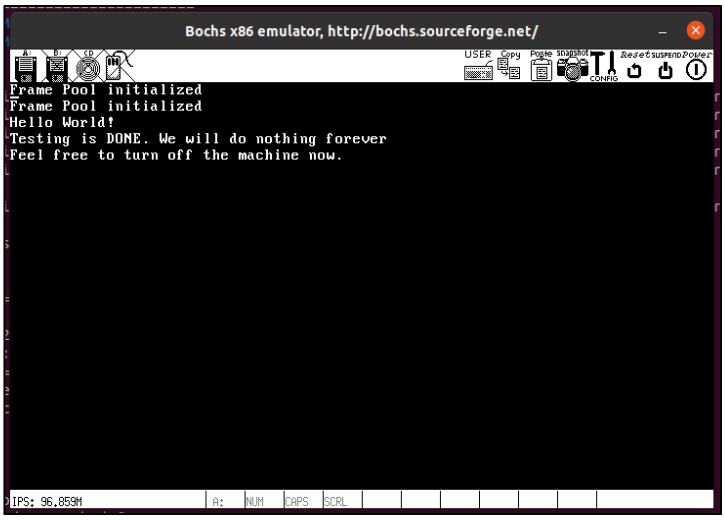


```
Frame Pool initialized
Frame Pool initialized
Hello World!
ContFramePool::get_frames Invalid Request - Not enough free frames available!
 Assertion failed at file: cont_frame_pool.C line: 263 assertion: false
```

unsigned int val = process_mem_pool.get_frames(2000);

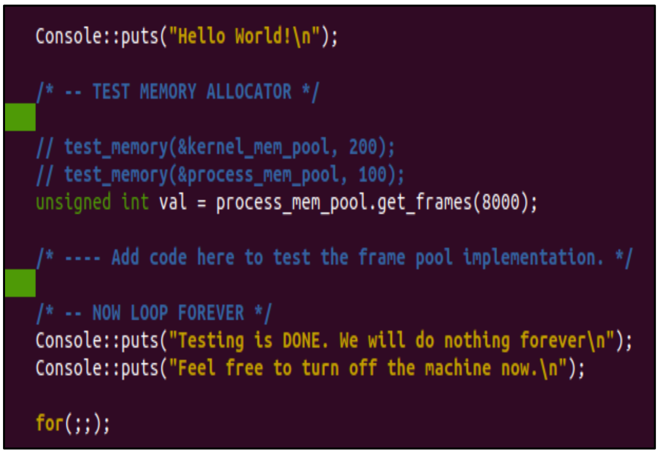


```
Console::puts("Hello World!\n");

/* -- TEST MEMORY ALLOCATOR */

// test_memory(&kernel_mem_pool, 200);
// test_memory(&process_mem_pool, 100);
unsigned int val = process_mem_pool.get_frames(2000);

/* ---- Add code here to test the frame pool implementation. */

/* -- NOW LOOP FOREVER */
Console::puts("Testing is DONE. We will do nothing forever\n");
Console::puts("Feel free to turn off the machine now.\n");

for(;;);

/* -- WE DO THE FOLLOWING TO KEEP THE COMPILER HAPPY. */
return 1;
```

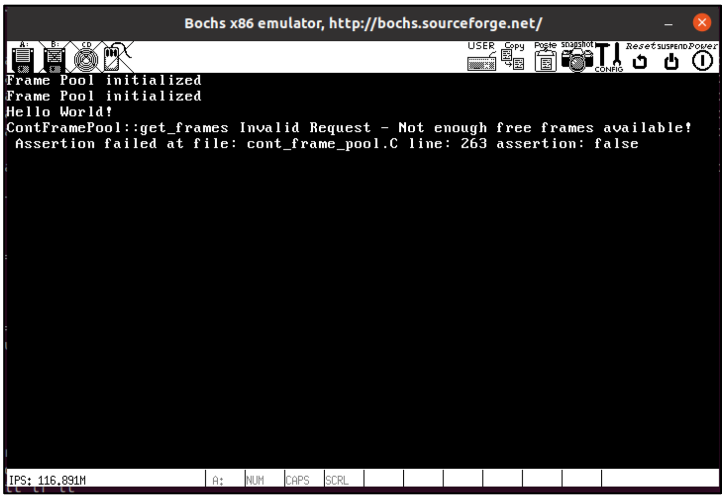


```
Frame Pool initialized
Frame Pool initialized
Hello World!
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.
```

unsigned int val = process_mem_pool.get_frames(8000);

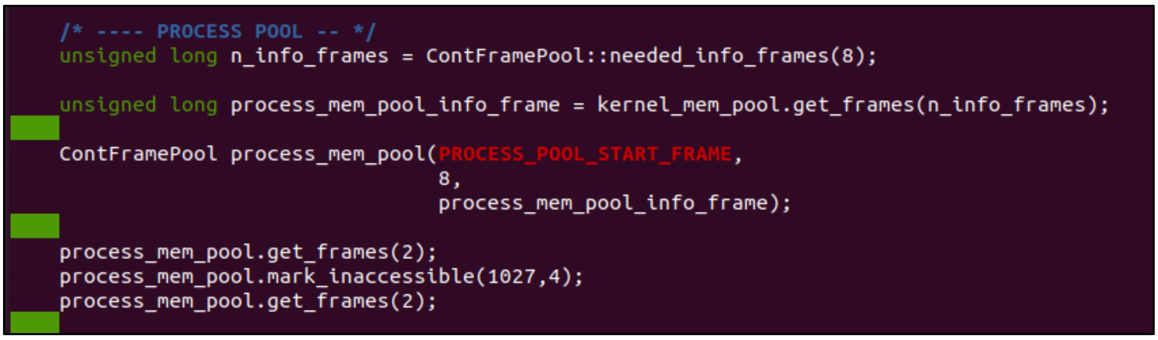


```
Console::puts("Hello World!\n");

/* -- TEST MEMORY ALLOCATOR */

// test_memory(&kernel_mem_pool, 200);
// test_memory(&process_mem_pool, 100);
unsigned int val = process_mem_pool.get_frames(8000);

/* ---- Add code here to test the frame pool implementation. */

/* -- NOW LOOP FOREVER */
Console::puts("Testing is DONE. We will do nothing forever\n");
Console::puts("Feel free to turn off the machine now.\n");

for(;;);
```

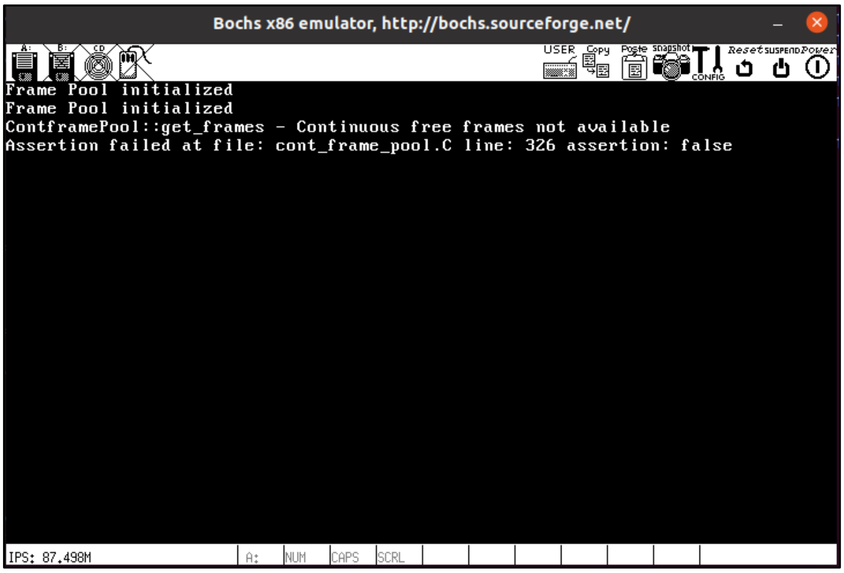


```
Frame Pool initialized
Frame Pool initialized
Hello World!
ContFramePool::get_frames Invalid Request - Not enough free frames available!
 Assertion failed at file: cont_frame_pool.C line: 263 assertion: false
```

5. Modifications made to kernel.C to perform Testcase No. 6:



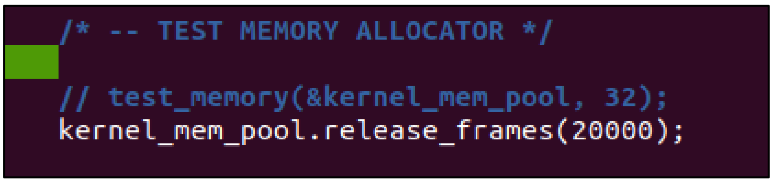
```
/* ---- PROCESS POOL -- */
unsigned long n_info_frames = ContFramePool::needed_info_frames(8);

unsigned long process_mem_pool_info_frame = kernel_mem_pool.get_frames(n_info_frames);

ContFramePool process_mem_pool(PROCESS_POOL_START_FRAME,
                               8,
                               process_mem_pool_info_frame);

process_mem_pool.get_frames(2);
process_mem_pool.mark_inaccessible(1027,4);
process_mem_pool.get_frames(2);
```



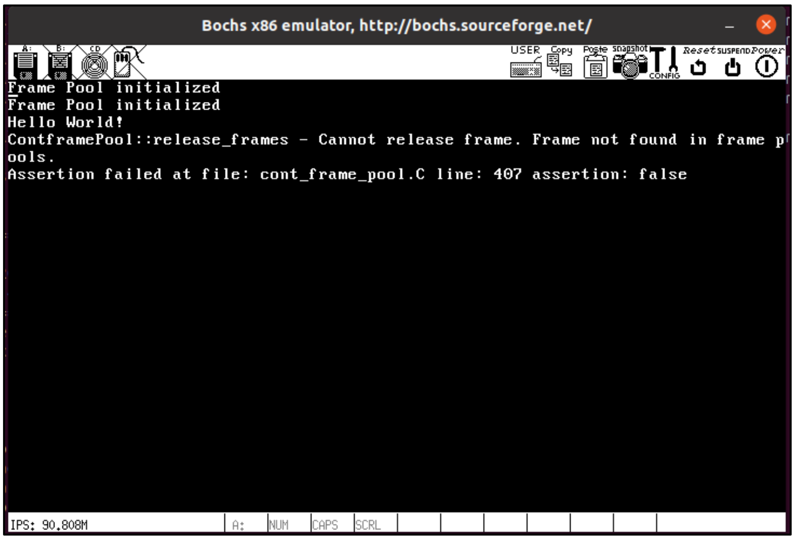
Bochs x86 emulator, http://bochs.sourceforge.net/
USER Copy Paste snapshot CONFIG Reset SUSPEND Power
IPS: 87.498M
A: NUM CAPS SCRL

```
Frame Pool initialized
Frame Pool initialized
ContframePool::get_frames - Continuous free frames not available
Assertion failed at file: cont_frame_pool.C line: 326 assertion: false
```

6. Modifications made to kernel.C to perform Testcase No. 7:



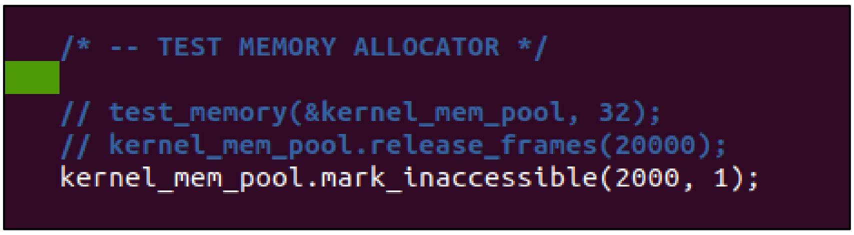
```
/* -- TEST MEMORY ALLOCATOR */

// test_memory(&kernel_mem_pool, 32);
kernel_mem_pool.release_frames(20000);
```



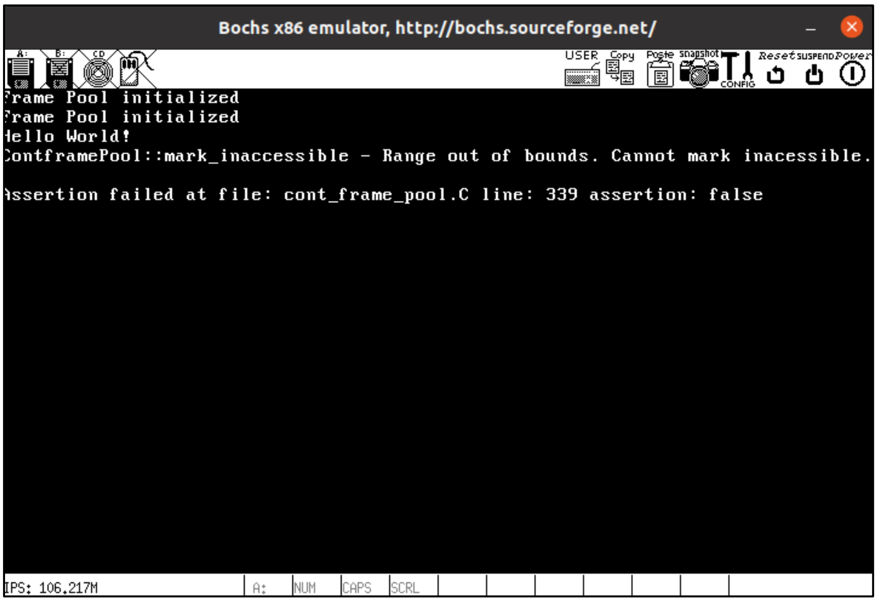
Bochs x86 emulator, http://bochs.sourceforge.net/
USER Copy Paste snapshot CONFIG Reset SUSPEND Power
IPS: 90.808M
A: NUM CAPS SCRL

```
Frame Pool initialized
Frame Pool initialized
Hello World!
ContframePool::release_frames - Cannot release frame. Frame not found in frame p
ools.
Assertion failed at file: cont_frame_pool.C line: 407 assertion: false
```

7. Modifications made to kernel.C to perform Testcase No. 13:

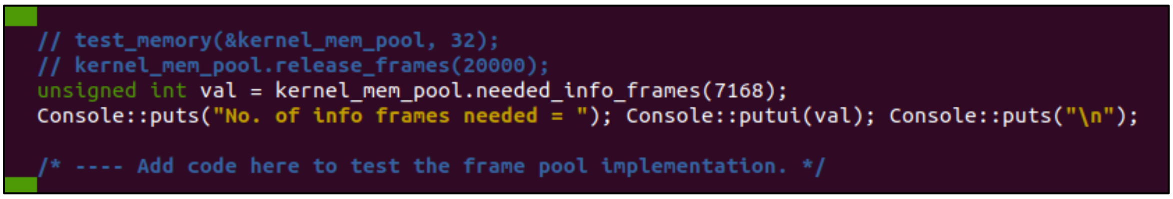


```
    /* -- TEST MEMORY ALLOCATOR */

    // test_memory(&kernel_mem_pool, 32);
    // kernel_mem_pool.release_frames(20000);
    kernel_mem_pool.mark_inaccessible(2000, 1);
```

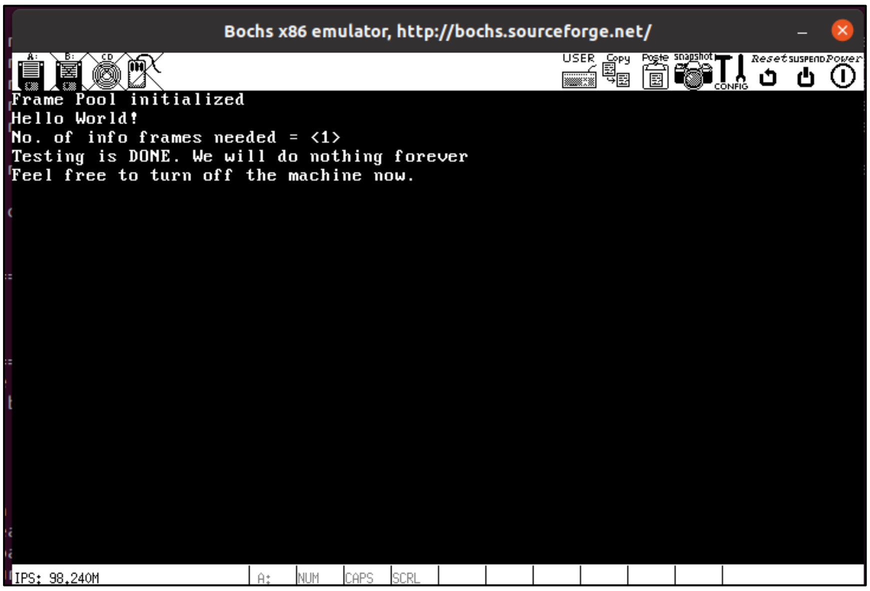


```
Frame Pool initialized
Frame Pool initialized
Hello World!
ContframePool::mark_inaccessible - Range out of bounds. Cannot mark inacessible.

Assertion failed at file: cont_frame_pool.C line: 339 assertion: false
```

8. Modifications made to kernel.C to perform Testcase No. 15:



```
    // test_memory(&kernel_mem_pool, 32);
    // kernel_mem_pool.release_frames(20000);
    unsigned int val = kernel_mem_pool.needed_info_frames(7168);
    Console::puts("No. of info frames needed = "); Console::putui(val); Console::puts("\n");

    /* ---- Add code here to test the frame pool implementation. */
```



```
Frame Pool initialized
Hello World!
No. of info frames needed = <1>
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.
```