

MP2: Simple File System

Sathvik Kote Revanasiddappa

UIN: 634005706

CSCE611: Operating System

Assigned Tasks

Main: Implement a simple frame pool manager as defined in file `cont frame pool.H/C`: Completed.

System Design

Objective

The main goal of MP2 is to create a Frame Pool Manager that can effectively handle the assignment of frames (physical frames) to both Kernel and Processor memory Locations.

Kernel assumptions and Frame Pool Specifications

The total amount of memory is 32MB. The first 4MB is reserved for kernel and 4MB to 32MB is for Process pool.

- Kernel pool: The first 2MB is reserved for emulator video and kernel stack. Kernel memory is from 2MB to 4MB.
- Process pool: The locations from 15MB to 16MB is reserved and should not be used as part of process pool. The process pool is from 4MB to 32MB.
- Page size: is 4KB

Code Description

BitMap Design

To track the frames that are used to by kernel or processes, bitmap is used. Bitmap stores the information regarding each frame under its belt. A frame is represented using 2 bits within the bitmap. Hence, a byte can store information regarding 4 frames. The total number of frames information that can be stores in a page is $4kb \times 4$ i.e 16k frames.

The states of the frame:

- 00: This state represents the frame is free.
- 01: This state represents the frame is used.
- 02: This state is used to mark the first frame in a frame block. This information can be used while deallocating or releasing the frames.
- 03: This state represents the frame is inaccessible. This state is used for the function `mark_inaccessible`.

Implementation Details

The below files are modified. And description gives information regarding the modification.

cont frame pool.H

- Static Variables: A variable pointer that points to the list of pool manager.
- Private Variables: A Set variables to hold the state of the pool manager.
 - `unsigned char* bitmap`: Pointer to the frame that hold information.
 - `unsigned int nFreeFrames`: Counter for number of free frames.
 - `unsigned long base_frame_no`: Holds the base frame number.
 - `unsigned long nframes`: Holds total number of frames.
 - `unsigned long info_frame_no`: Holds frame number of information.
 - `ContFramePool* nextPool`: Stores the pointer to the next pool.
 - `enum class FrameState Free, Used, HoS, Head`: Enum which helps to store 4 different states of a frame.
- Private Method declarations: Few methods are declared to implement the overall functionality. Below are the method declarations. Description about the methods can be found in next section.
 - `FrameState get_state(unsigned long_frame_no);`
 - `void set_state(unsigned long _frame_no, FrameState _state);`
 - `unsigned char get_state_value(FrameState _state);`
 - `bool is_valid_frame(unsigned long _frame_no);`
 - `void free_frames(unsigned long _frame_no);`

cont frame pool.C

This file has the implementation of the above declarations.

- `FrameState get_state(unsigned long_frame_no)`: The get state method returns the state of the given frame number. It uses bit operations to obtain the state of the frame. `frame_no / 4` will give the index of the byte which holds the information. `frame_no % 4` will give the column index within the byte.
 - If the value of two bits is 0, then the frame is Free.
 - If the value of two bits is 1, then the frame is Used.
 - If the value of two bits is 3, then the frame is inaccessible, HoS.
 - If the value of two bits is 2, then the frame is the Head frame. The head frame marks the beginning of the allocated frames.
- `unsigned char get_state_value(FrameState _state)`: The function gives a mapping between the FrameState enum and the bit value. Below are the mappings.
 - `FrameState::Free - 0`
 - `FrameState::Used - 1`
 - `FrameState::Head - 2`
 - `FrameState::Hos - 3`
- `void set_state(unsigned long _frame_no, FrameState _state)`: As the name suggests the set state methods, sets the state of the given frame no. It uses similar logic to extract the location of the frame in the bitmap. `frame_no / 4` will give the index of the byte which holds the information. `frame_no % 4` will give the column index within the byte. It uses `get_state_value` function to get the value mapping of a given FrameState.
- `bool is_valid_frame(unsigned long _frame_no)`: The function checks whether the given frame no is a valid frame. A frame is valid, if it belongs to the frame manager or pool manager. This can be used while releasing the frames and allocating the frames.

- **void free_frames(unsigned long _frame_no):** The function is used to free the frames that starts at the given frame number. The function validates whether the given frame no is the head frame. If the frame is the head frame, it will free all the used frames following the head frame.
- **unsigned long get_frames(unsigned int _n_frames):** Get frames is a public method exposed by the pool manager. It follows the below steps to allocate the continous frames.
 - It starts searching for the free frame from the base frame.
 - It searches for the set of n requested frames.
 - If found, it will set the starting frame to Head state and the rest of the frames to the Used states.
 - If not found, it will return the 0
- **release_frames(unsigned long _first_frame_no):** The static method is used to release the frames (set the state to Free). It follows the below steps:
 - It traverses the list of the all the pool managers using the static variable pools.
 - Every pool has a method to valid the frame no. If the frame belongs to the given pool, then free frames function is called.
 - Free frame starts from the head frame and deletes all the used frames which followed the head frame.
- **ContFramePool::needed_info_frames(unsigned long _n_frames):** The need info frames function returns the number of frames required to store the information regarding n_frames. The calculation is as follows:
 - Every frame requires two bits. Total number of bits = $n_frames * 2$
 - Then the total number of bytes = total bits / 8
 - Then the total frames = Ceil (total bytes / frame size)

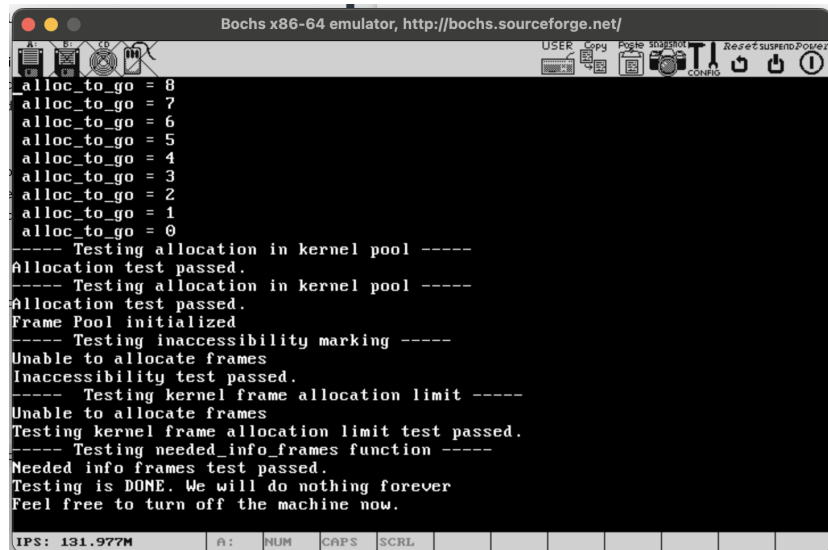
kernel.C

Kernel file was edited to add the custom tests.

Testing

Apart from the given test function, few additional test functions are added.

- Testing allocation in kernel pool: The test verifies whether the kernel pool successfully allocates a frame.
- Testing allocation in process pool: The test verifies whether the process pool successfully allocates a frame.
- Testing release frames: Create process frames and release those frames. The test will verify the release frame functionality and also tests the traversing of the pool lists.
- Testing inaccessibility marking: This test verifies that frames marked with Hos state can't be accessed again.
- Testing kernel frame allocation limit: This test verifies whether the pool manager refuses to allocate frames above the limit.
- Testing needed_info_frames function: This test verifies whether needed info frames works or not.

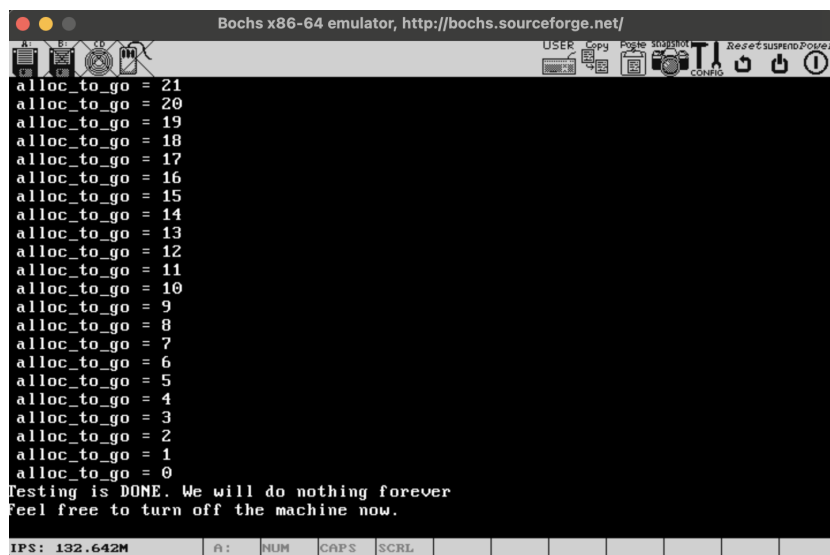


The image shows a screenshot of a Bochs x86-64 emulator window. The title bar reads "Bochs x86-64 emulator, http://bochs.sourceforge.net/". The window contains a black terminal area with white text. The text shows a series of "alloc_to_go" values from 8 down to 0, followed by several test results: "Testing allocation in kernel pool", "Allocation test passed.", "Testing allocation in kernel pool", "Allocation test passed.", "Frame Pool initialized", "Testing inaccessibility marking", "Unable to allocate frames", "Inaccessibility test passed.", "Testing kernel frame allocation limit", "Unable to allocate frames", "Testing kernel frame allocation limit test passed.", "Testing needed_info_frames function", "Needed info frames test passed.", and finally "Testing is DONE. We will do nothing forever" and "Feel free to turn off the machine now.". At the bottom of the window, there is a status bar with the text "IPS: 131.977M" and several checkboxes for "A:", "NUM", "CAPS", and "SCRL".

```
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 allocation in kernel pool -----
Allocation test passed.
----- Testing allocation in kernel pool -----
Allocation test passed.
Frame Pool initialized
----- Testing inaccessibility marking -----
Unable to allocate frames
Inaccessibility test passed.
----- Testing kernel frame allocation limit -----
Unable to allocate frames
Testing kernel frame allocation limit test passed.
----- Testing needed_info_frames function -----
Needed info frames test passed.
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.

IPS: 131.977M  A:  NUM  CAPS  SCRL
```

Figure 1: Results of Custom Tests



The image shows a screenshot of a Bochs x86-64 emulator window. The title bar reads "Bochs x86-64 emulator, http://bochs.sourceforge.net/". The window contains a black terminal area with white text. The text shows a series of "alloc_to_go" values from 21 down to 0, followed by the text "Testing is DONE. We will do nothing forever" and "Feel free to turn off the machine now.". At the bottom of the window, there is a status bar with the text "IPS: 132.642M" and several checkboxes for "A:", "NUM", "CAPS", and "SCRL".

```
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: 132.642M  A:  NUM  CAPS  SCRL
```

Figure 2: Results of Existing Tests