Design Document
Continuous frame pool manager

CSCE 611
Operating Systems
MP2 - Fall 2018
by
Himanshu Gupta

This machine problem needs a frame manager for allocation and management of continuous frame pool in kernel memory. The total system memory available is 32 MB, of which 4 MB is used by kernel and a 1MB hole at 15MB which is inaccessible. The frame size used is 4KB.

The implementation of this frame pool manager is divided into several modules as API's in the frame manager. Basically, we need to implement the following API's:

- Class Constructor – ContFramePool()
- get_frames
- mark_inaccessible
- release_frames
- needed_info_frames

From these, the release_frames() api is a static method and can be called without any object of the class. As we have 4 different mark to set on a frame, we will use 2 bits to represent one frame. Following represents how they are assigned in this implementation:

- 01 - Head of frame
- 11 - ALLOCATED
- 00 - FREE
- 10 – Inaccessible

Next we define the private variables to be used in the class ContFramePool. Apart from the regular variable needed for frame counts, I added a singly linked list to track the number of framepools being instantiated. These are static lists and can be used by the release_frames api to determine which frame pool manages the particular frame and then act accordingly.

I have used just one bitmap to track the allocated frames and their management information using the values mentioned above.

We now describe the implementation with each of the above API

1. **Constructor**: This takes all the arguments and assigns them to private variables. Sets the bitmap pointer based on info_frame_no. Next, it will update the list of objects that have been created from this class. So, we basically update the list and next pointer for same.
2. **get_frames:** This is the core API of this whole implementation. Probably the most complex one as well. This will determine if we have the number of continuous frames available in the frame pool. The search algorithm is optimized to run in O(n) time, i.e. the length of frame pools. It does a linear search on the frames and determines if we a continuous length of those frame available. If found, it will update the first frame as head of the sequence and then mark rest of them as allocated. It then returns the frame number at which we found the empty sequence.
3. **mark_inaccessible:** This API basically marks the set of frames as inaccessible (10). It also checks whether the range of frames given is within the number frames managed by this frame pool.

4. **release_frames**: This API frees the allocated frames and marks them as free(00) to be used by other process. The first part of this API is to determine the frame pool from which this frame will be released. To do that, we access the static list of frame pools to check their range and then use its bitmap pointer. The marking of free frames continues till we have allocated frames.
5. **needed_info_frames**: This API simply returns the number of management frame needed for a particular memory size.

TESTING

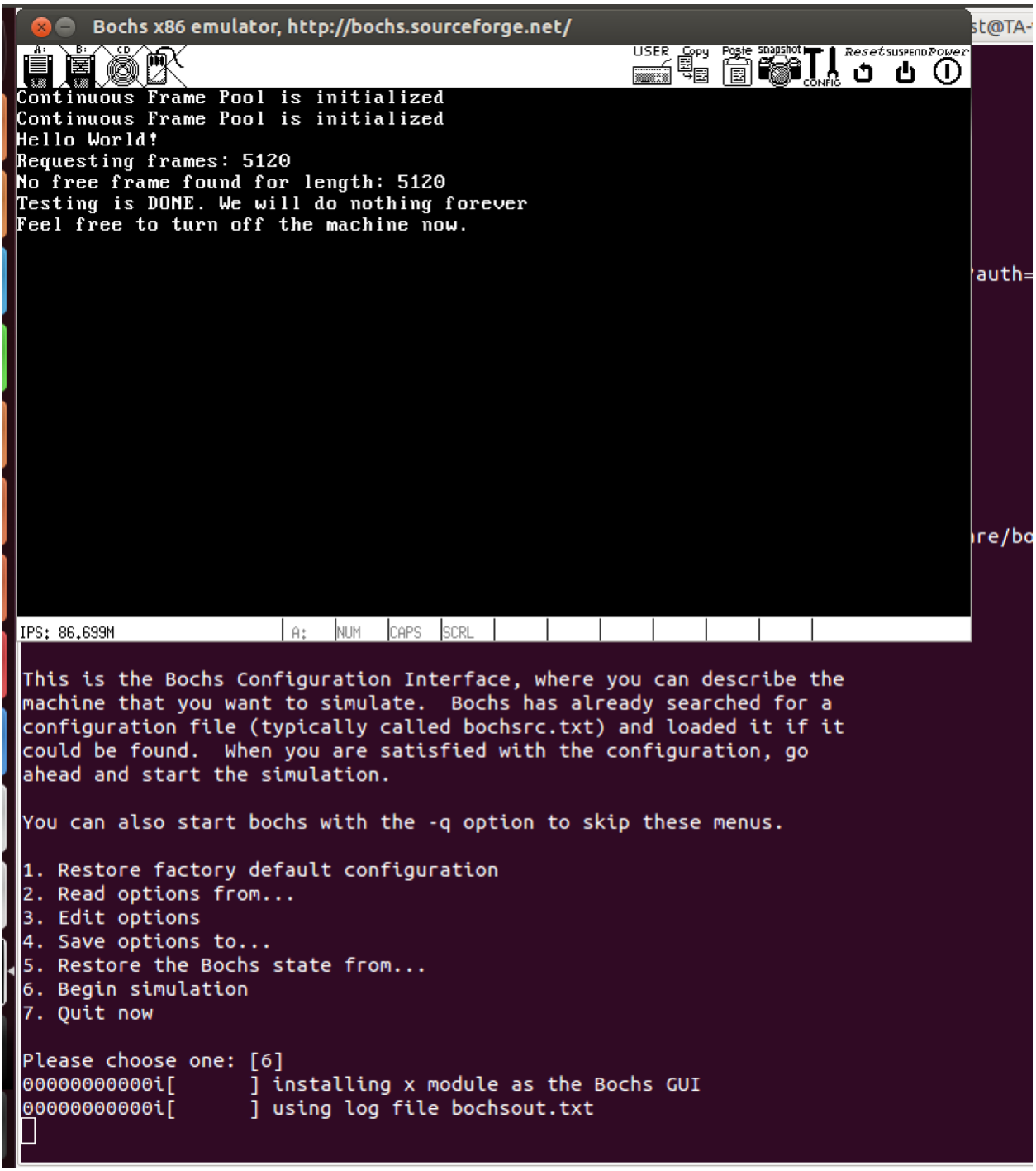
The tricky part of this implementation was the algorithm for accessing 2 bits at time and then the bitwise operations that were done to classify it in any of the category.

I tested each of the API written above individually by several debugs and calling them from kernel.C, the test_memory functionalilty was added for the process memory pool as well.

How the frame pool was tested: Dump of the frame map in the file cont_frame_pool.C indicate clearly at each step how the map is updated correctly and values are reset after the release frame function. I added an invalid get for the get_frame function, which queries the api for frames of 20 MB but is then successfully denied as there is a hole in the memory at 15MB. This also helped identify some issues with the get_frame function.

I don't see the need for more test api's as all of them got tested with the test_memory and detailed debugs. The commits from git will indicate several small bugs identified and fixed.

Screenshot of the successful run.


Bochs x86 emulator, http://bochs.sourceforge.net/
A:
B:
CD
USER
Copy
Paste
snapshot
CONFIG
Reset
suspend
Power
Continuous Frame Pool is initialized
Continuous Frame Pool is initialized
Hello World!
Requesting frames: 5120
No free frame found for length: 5120
Testing is DONE. We will do nothing forever
Feel free to turn off the machine now.
IPS: 86.699M
A:
NUM
CAPS
SCRL
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]
00000000000i[ ] installing x module as the Bochs GUI
00000000000i[ ] using log file bochsout.txt