

CSCE 611 Operating Systems
MP2 - Continuous frame pool manager
Design Document
18th September 2022
Ram Sankar Koripalli

This design document explains the implementation of continuous frame pool manager. The frame manager is responsible for allocation and deallocation of contiguous frames.

The frame pool manager consists of the following APIs:

- ContFramePool() - constructor
- get_frames()
- mark_inaccessible()
- release_frames()
- needed_info_frames()

Unlike the simple frame pool manager which has only two states (free and used) and can be represented in 1 bit, the continuous frame pool manager has 4 states and needs 2 bits [free (00), head of sequence (01), allotted (11), inaccessible (10)].

A linked list is used to maintain the frame pool objects. The release frames API uses this list to determine the mapping of frame to frame pool.

Bitmap is used to manage the frame allocation by maintaining the frame states according to the above values.

Detailed Implementation of APIs in **cont_frame_pool.C**:

Constructor: All the variables declared in the header file are initialized according to the arguments of the constructor. It marks all the frames after the info frame as free.

get frames : This API takes number of frames as input and returns the head of the frames as output. This API initially makes a linear search in the frame pool and determines whether there exists a contiguous frames of required length. After finding the frame sequence, the first frame of the sequence is marked as head of sequence and the rest of them as allocated. Finally, the starting frame is returned back to the caller.

mark inaccessible : This API is similar to get_frames. The function takes base frame and the no. of frames as input, then it masks the head of the sequence as inaccessible and the rest as allocated. The first frame should be marked as inaccessible instead of head of sequences because if release_frames function takes in the head of sequences as input, it would mark the inaccessible area as free.)

release frames: The release frame takes frame no. as input, it determines the corresponding frame pool using the list object. The bitmap of the frame pool is masked as free starting from the head of sequence until it finds free or new head of sequence

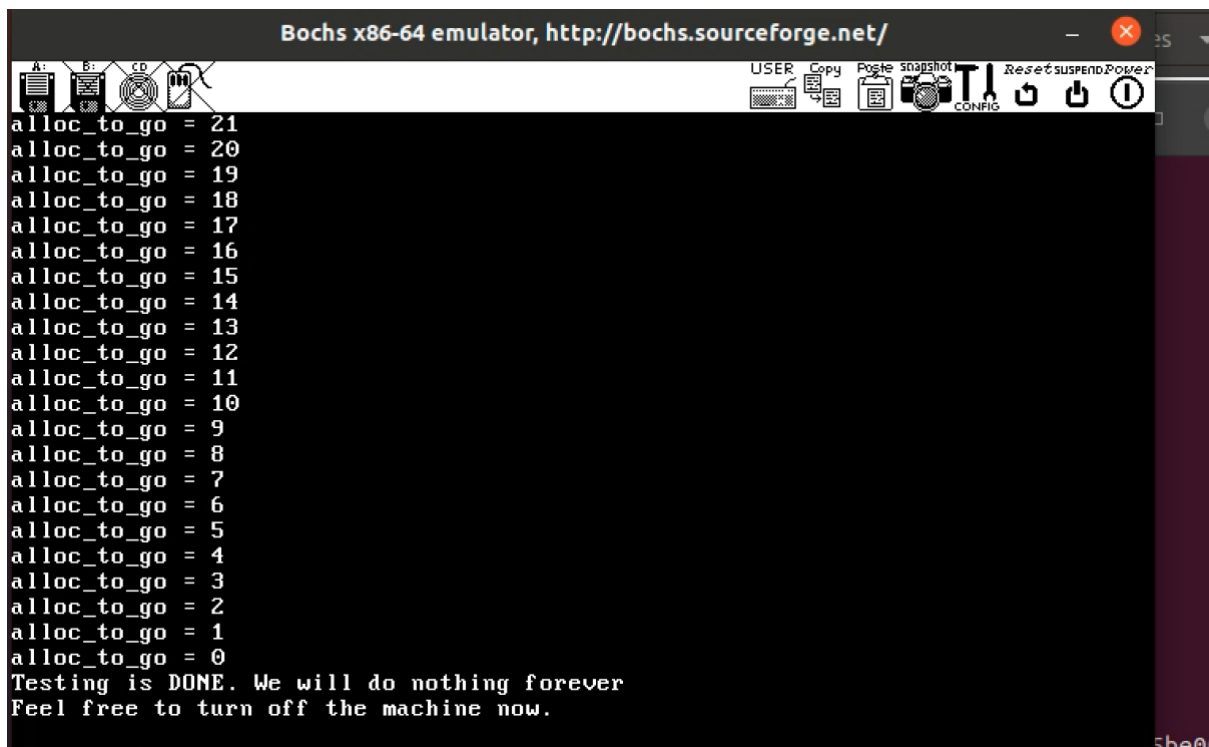
needed info frames: This API returns the number of frames required for a given memory size.

TESTING & RESULT

Made a change to the kernel.C to include the testing of process memory pool along with kernel memory pool.

The kernel was built successfully using *make* and copied to the disk image.

The kernel was tested using bochs emulator and it ran successfully with no error messages. Following is the screenshot of the kernel in action.

The image shows a screenshot of the 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 decreasing from 21 to 0, followed by a message: "Testing is DONE. We will do nothing forever. Feel free to turn off the machine now." The emulator's interface includes a toolbar at the top with icons for USER, Copy, Paste, snapshot, CONFIG, Reset, suspend, and Power. On the right side, there are additional icons for a floppy disk, a hard disk, and a CD-ROM. The bottom right corner of the window shows the text "5be0".

```
Bochs x86-64 emulator, http://bochs.sourceforge.net/
A: B: CD
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.
```