Machine Problem 2: Frame Manager

Gaurangi Sinha

UIN: 734004353

CSCE611: Operating System

The goal of this assignment is to initiate the development of a demand-paging virtual memory system for a kernel by focusing on creating a frame manager. The frame manager's primary role is to oversee the allocation and deallocation of physical memory pages, or "frames." It must accurately monitor the status of each frame, distinguishing between those that are currently in use and those that are available for allocation. Given that various segments of the operating system will utilize different portions of memory, the frame manager will control multiple "frame pools." These frame pools are designed to facilitate operations for acquiring and releasing frames, ensuring efficient memory management across different system components.

we need to implement the following API's:
• Class Constructor – ContFramePool()
• allocate_frames
• release_allocated_frames

I also made changes in the cont_frame_pool.H and cont_frame_pool.C

The purpose of the ContFramePool module, is to manage a pool of contiguous memory frames. This involves:

- Allocation and Deallocation: Functions within this file are used to handle the allocation of contiguous frames to various parts of the operating system and deallocate them when they are no longer needed.
- Tracking Frame Usage: Keeping track of which frames are free and which are in use, possibly using a bitmap or another data structure for efficient lookup and management.
- Support for Multiple Frame Pools: Implementing mechanisms to manage multiple pools of frames, allowing for different parts of the operating system to have dedicated memory resources.

Key Components for cont_frame_pool.H

- Preprocessor Guards: Ensures the header is included only once during compilation to prevent multiple definition errors.
- Includes: The file includes machine.H, which suggests that it relies on definitions or constants provided in this file, related to the underlying hardware or machine specifics like page size.
- Data Structures: Custom structures or classes defined within for managing the frame pool.
- Functionality: The description implies the file will declare classes or functions to:
    - ❖ Allocate and deallocate contiguous frames.
    - ❖ Keep track of free and used frames within the pool.
    - ❖ Possibly support operations like marking certain frames as inaccessible, reflecting in memory protection or access rights.

- Class Declaration: A ContFramePool class encapsulates the functionality for frame allocation, deallocation, and status tracking.
- Member Variables: To store the pool's state, such as the bitmap for frame allocation status, counters for free frames, and identifiers for the start and end of the managed memory region.
- Member Functions: Methods for operating on the frame pool, including constructors for initialization, methods for frame allocation/release, and utility functions for state inspection or debugging.
- Static Members: Static variables and methods to manage or access global properties of the frame pool system, such as a list of all frame pools.

I also made minor changes in kernel.C. That is I modified kernel.C file. I just changed the function names, from get_frames to allocate_frames.
And changed the function name of release_frames to release_allocated_frames.

cont_frame_pool.C (changes made)

Implementing New Methods: Coding the logic for newly declared methods in the header file, such as algorithms for finding contiguous free frames or handling frame state transitions.

Refactoring Existing Logic: Enhancing or optimizing existing methods, possibly by improving efficiency, fixing bugs, or updating the logic to reflect changes in the system's requirements or capabilities.
Updating Global or Static Variables: Adjustments to static or global control structures, such as lists or counters managing the overall state of frame pools.
Error Handling: Introducing or enhancing error checking and handling mechanisms to ensure robustness, especially in critical systems like memory management.
Comments and Documentation: Revising comments within the source code to better describe the logic, assumptions, or changes made during the update process.

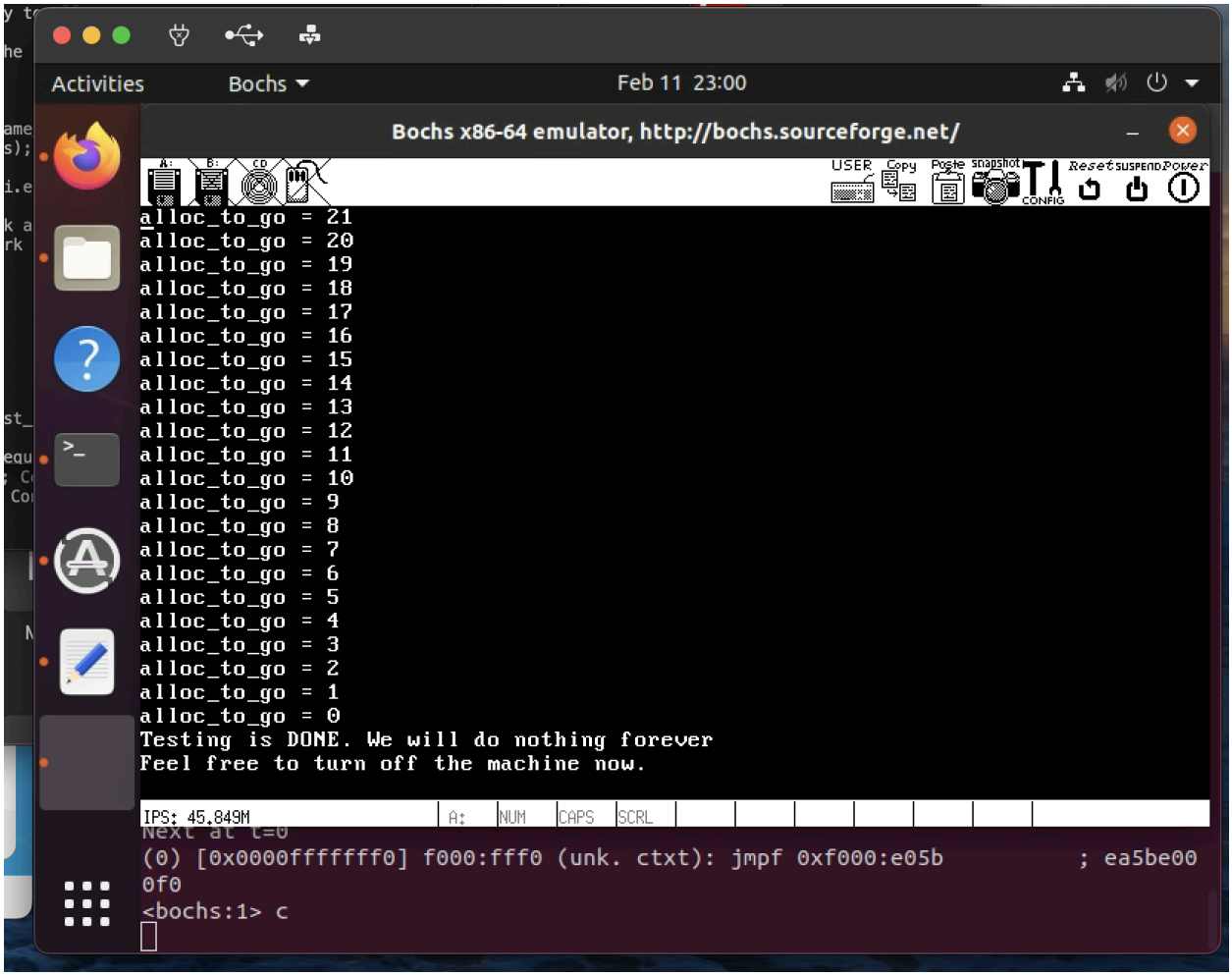
Compilation

To compile the code, we run the following commands

```
make clean
Make
./copykernel.sh
bochs –f bochsrc.bxrc
```

Testing

We get the following result. Screenshot of the successful run.


Activities
Bochs
Feb 11 23:00
Bochs x86-64 emulator, http://bochs.sourceforge.net/
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: 45.849M
(0) [0x0000fffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b ; ea5be00
0f0
<bochs:1> c