CSCE 611: OPERATING SYSTEMS
MACHINE PROBLEM 2 - FRAME MANAGER
DESIGN DOCUMENT

VISHNUVASAN RAGHURAMAN
UIN - 234009303

In the MP2, the below files have been modified:

- cont_frame_pool.C
- cont_frame_pool.H
- utils.h -> to define NULL

The handout mentions the following:

- The total memory available in the machine is 32MB.
- The first 4MB is reserved exclusively for the kernel, which includes both code and kernel data.
    - Within this 4MB range, the first 1MB contains all global data, while the actual kernel code begins at address 1MB.
- Memory within the initial 4MB range is directly mapped, meaning that each memory address corresponds directly to a physical frame. However, memory beyond the 4MB mark is freely mapped. This means that each page in this range can be mapped to any available physical frame, providing flexibility in memory allocation.

Approach used - Bitmap approach
A bitmap was utilised to manage the frames within the frame pool, with two bits allocated for each frame to denote its state as *Used, Free, or HoS*. Since we have 3 distinct or unique states, it was decided to use 2 bits.

The mapping of states was defined as follows:

- **Free**: Represented by the bit sequence "11".
- **Used**: Represented by the bit sequence "01".
- **HoS**: Represented by the bit sequence "00".

Additionally, a *singly linked list* was utilized to oversee and navigate the kernel pool. This design choice aimed to streamline the management of kernel frames within the pool.

- Frame Pool Construction:
    - The class *ConstFramePool* is constructed with parameters such as *Base Frame(base_frame_no), Frame Pool Size(n_frames), and Info Frame(info_frame_no).* These parameters are essential for initializing and managing the frame pool effectively.

- Upon initialization, all bits within the bitmap are set to the Free state. Subsequently, the first bit in the bitmap is marked as Used, signifying its allocation.

- To manage the kernel pool, a LinkedList structure(Singly Linked List) is utilized. The head of this LinkedList is assigned to the kernel pool, facilitating efficient navigation and management of kernel frames within the pool.

- The process of initializing the *ConstFramePool* class with these parameters, setting the bitmap states, and assigning the LinkedList head to the kernel pool ensures effective control and utilization of memory resources.

- *get_frames*
  - The allocation process in the *ConstFramePool* class ensures available and contiguous frame allocation:
    - **Frame Availability Check:** Frames to be allocated are checked for availability in the *Free* state and ensure the requested frame count doesn't exceed the pool's maximum.
    - **Skipping Used Frames:** *Used* frames are skipped while counting to determine the starting point for allocation.
    - **Contiguous Allocation:** The code checks for contiguous frame availability, throwing an error if insufficient contiguous space exists.
    - **Starting Frame Identification:** The starting frame for contiguous allocation is identified.
    - **Frame Allocation:** Frames are allocated from the starting frame onwards, marking the first bitmap as *Head* state and subsequent frames as *Used*.

- *release_frames & release_frame_range*
  - To release frames, the frame number parameter is used to identify the frame for release.
  - We verify whether the frame number corresponds to a HoS state and proceed to deallocate or free the subsequent frames until encountering the next frame with a HoS state.
  - We determine the frame pool where the frame to be released is located by utilizing the frame number, base frame number, and the number of frames. This enables us to locate the specific frame pool corresponding to the provided frame number, leveraging the linked list structure for efficient traversal.
  - Subsequently, if the identified frame has a HoS state, we proceed to release the frames and continue deallocating until encountering the next frame with a HoS state. However, if the frame is not the Head frame, we raise an error.

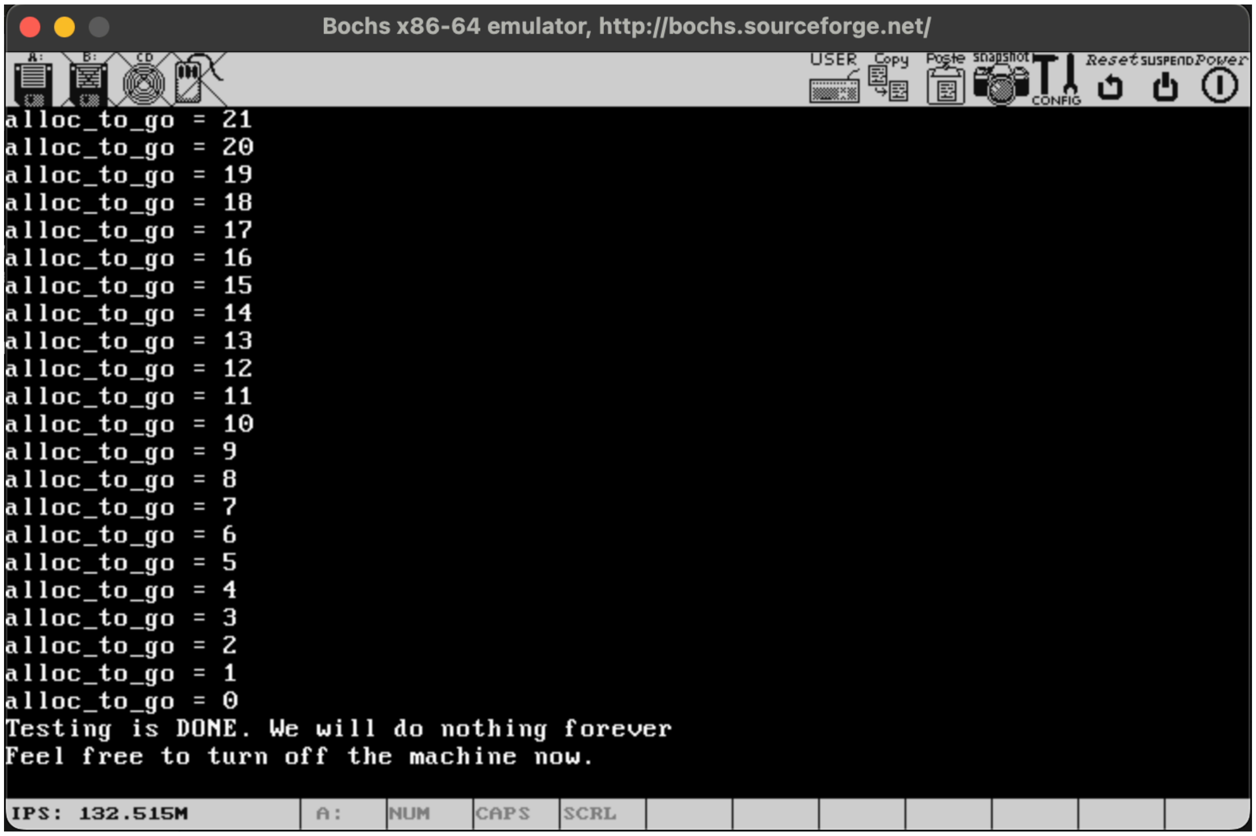
- *mark_inaccessible*
  - In the *mark_inaccessible* function implementation, we designate frames as either already in use or mark them as HoS. Beginning from the specified frame number

up to the number of frames, we assess whether they are already in use. If not, we designate them as HoS state.

- *needed_info_frames*
  - We return the number of information management frames needed for the allocated memory size of the pool.

The above mentioned steps and points are the implementation carried out for this Machine Problem. The Bochs Emulator screenshot is attached below.

OUTPUT:



```
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.
```