## Design Idea:

**Page Table Management (implemented in page_table.c):**

The entire memory (32 MB) is divided among two frame pools:

- **kernel_mem_pool:** Memory in the first 4MB is direct mapped to physical memory, i.e., physical and logical addresses are identical. The first 2 MB is reserved for system use. The kernel_mem_pool, between 2MB and 4MB, manages frames in the shared portion of memory, typically for use by the kernel for its data structures.
- **process_mem_pool:** Above 4MB, the process_mem_pool manages frames in the non-shared memory portion (28MB), i.e., the freely mapped memory, where logical addresses are not the same as physical addresses. Frames in the logical address space are mapped to their physical address space using 2-level paging. Every page in this address range is mapped to whatever physical frame was available when the page was allocated.

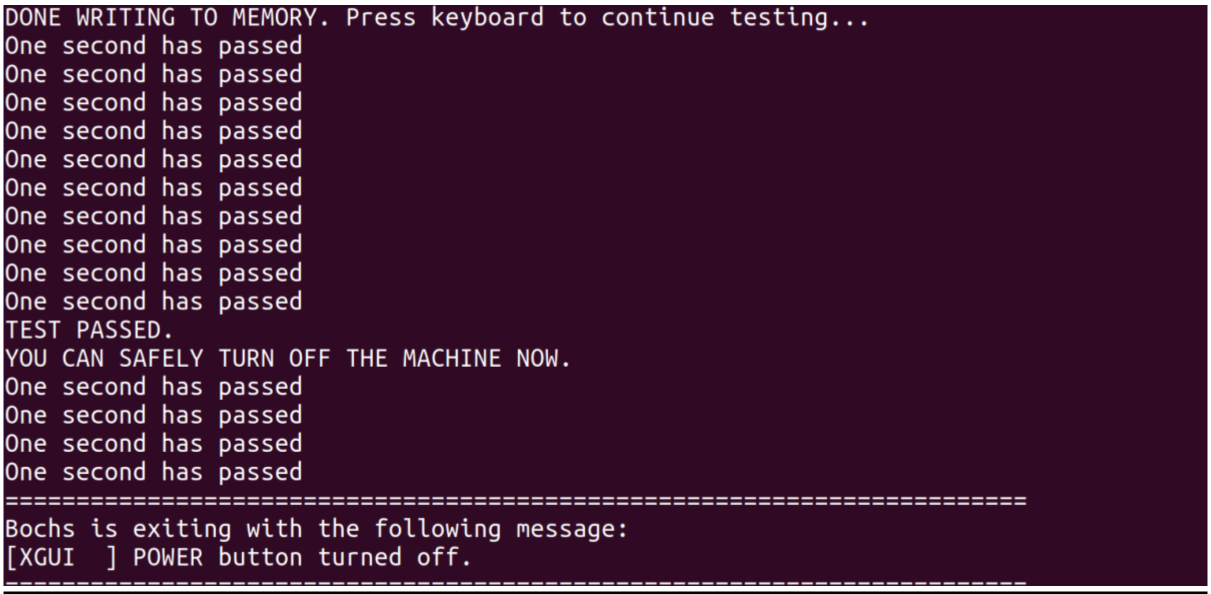
After the pools are initialized, the following functions help the kernel to set-up the overall paging experience.

- **init paging:** set the parameters for the paging subsystem.
- **PageTable constructor:** sets up the entries in the page directory and the page table for the shared portion of memory. The page directory points to each page table in the shared memory and is set as valid using the last bit of the entry. The page table entries for the shared portion of the memory (i.e., the first 4MB) are marked valid using the last bit in each entry.
- **load() function**: The page table is loaded into memory by storing the address of the page directory into the CR3 register (PTBR).
- **enable_paging() function:** It helps the kernel switch from physical addressing to logical addressing. The paging is enabled by setting the Most Significant Bit (MSB) in the CR0 register.
- **handle_fault():** This method first reads the logical_address from the CR2 register and the error_word from the error_code present in the REGS object. It then looks up the appropriate entry in the page directory and the page table for faulty entries using the valid bit ($0^{th}$ bit). If there is no physical memory (frame) associated with the page, an available frame is brought in, and the page-table entry is updated. If a new page-table page has to be initialized, a frame is brought in for that, and the new page table page and the directory are updated accordingly.

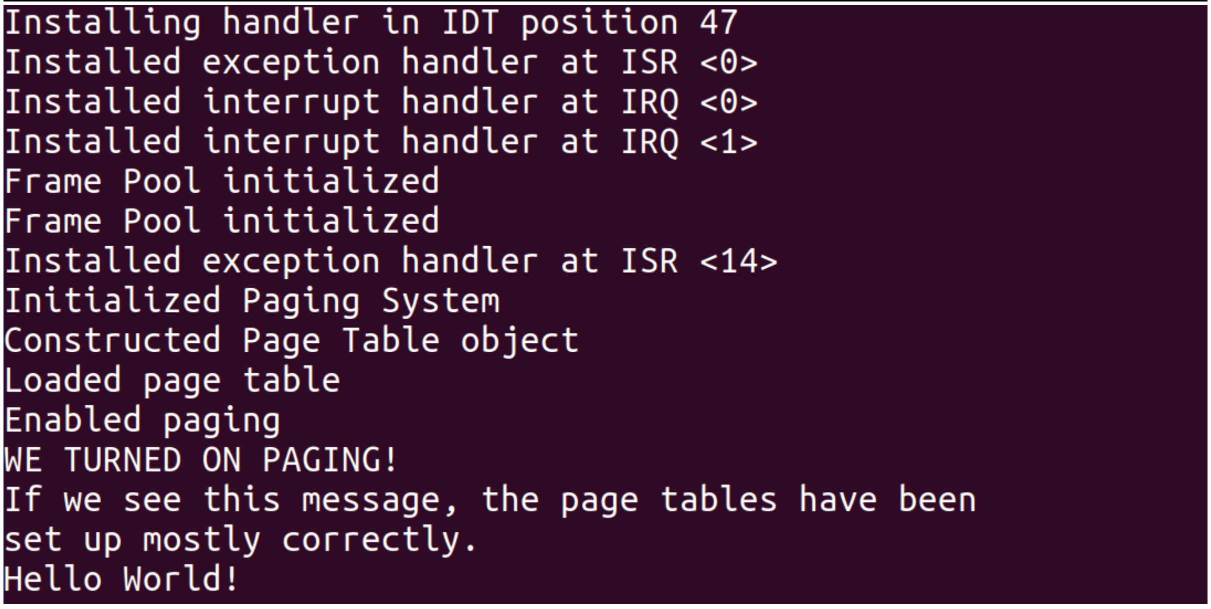
Every entry in a frame takes 32 bits, where:

- **Bits 31-12:** represent the address (of Page Table / Frame) stored in the entry.
- **Bits 11-9:** Available bits
- **Bits 8-3:** Reserved bits for various purposes.
- **Bit 2:** User/Supervisor Bit (0 -> Supervisor Bit, 1-> User Bit)
- **Bit 1**: Read/Write Bit (0-> Read-Only, 1-> can be written to)
- **Bit 0:** Valid Bit (0 -> Invalid entry, 1-> Valid Entry)

**Test Screenshots / Output:**



```
DONE WRITING TO MEMORY. Press keyboard to continue testing...
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
========================================================================
Bochs is exiting with the following message:
[XGUI  ] POWER button turned off.
------------------------------------------------------------------------
```



```
Installing handler in IDT position 47
Installed exception handler at ISR <0>
Installed interrupt handler at IRQ <0>
Installed interrupt handler at IRQ <1>
Frame Pool initialized
Frame Pool initialized
Installed exception handler at ISR <14>
Initialized Paging System
Constructed Page Table object
Loaded page table
Enabled paging
WE TURNED ON PAGING!
If we see this message, the page tables have been
set up mostly correctly.
Hello World!
```

## Frame Manager (implemented in cont_frame_pool.c):

I use a bitmap to store the three states of a frame – Free, Used, Head of Sequence. 2 bits are used to represent the state of a frame as below.

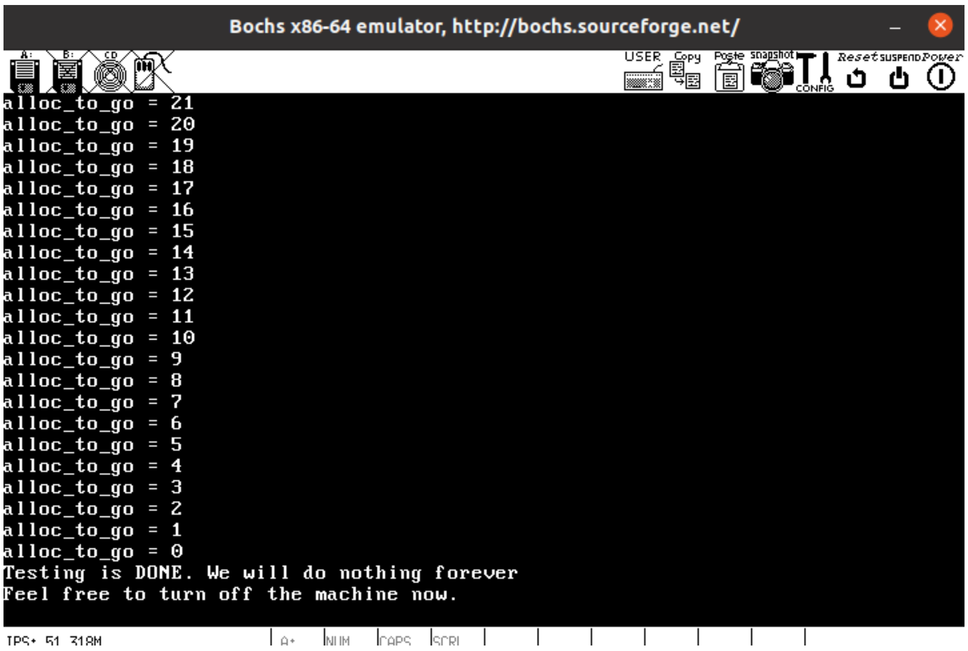
- 00 -> represents Free Frame
- 11 -> represents Used Frame
- 10 -> represents HoS (Head of Sequence) Frame

When a new frame pool is created, the pool gets appended to a static pool list, which is helpful for releasing any frame from any pool. New frames can be requested, and freed at any time, using two pools: kernel_mem_pool and process_mem_pool.

When requesting new frames, it is checked if contiguous space is available, otherwise an error is raised. If the space is available, status of the corresponding frames are updated to Used, and the first frame is marked as HoS. The address of the first frame number is returned.

When releasing frames, its pool, and base frame number are fetched first. If not found, an error is raised. Otherwise, the bitmap state of the frames in the pool are freed starting from the first frame until the state of a frame does matches Used. (This is because all the frames are allocated memory in a contiguous fashion.)

**Test Screenshots / 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.
```

## Documents changed:

I have only changed the following files, as mentioned in the instructions:

- page_table.c
- cont_frame_pool.c
- cont_frame_pool.h