MP-3 Design Document

Files Changed:
cont_frame_pool.c cont_frame_pool.H and page_table.c page_table.H files are edited.
Objective:
The objective of this MP is to implement demand-paging based virtual memory system. The frame manager of physical memory is used to map pages to frames.

Two Level Paging:

In the two level paging mechanism we have a Page Directory which addresses page tables in the next level. In our case we have 32 bits in the virtual address space, of these 32 bits first 10 bits are allocated to index into the page table directory. The directory addresses 1024 page tables. Each page table has 10 bits which are used to index into the page table. And lastly the 12 bits are used as offset used to index in the physical frame. The virtual address space can address 4GB with the 32 bits.

Before proceeding with paging, we need to note few things. Firstly, the first 4mb of the process is directly mapped to the kernel and the rest is mapped virtually. And secondly we need to initialise the paging system, which is done below in init_paging function.

The following control registers' state is used to implement the above functionality:
CR0: Indicates if paging is enabled or disables
CR2: Stores the fault address
CR3: It is the page table base register which stores the address of page directory.

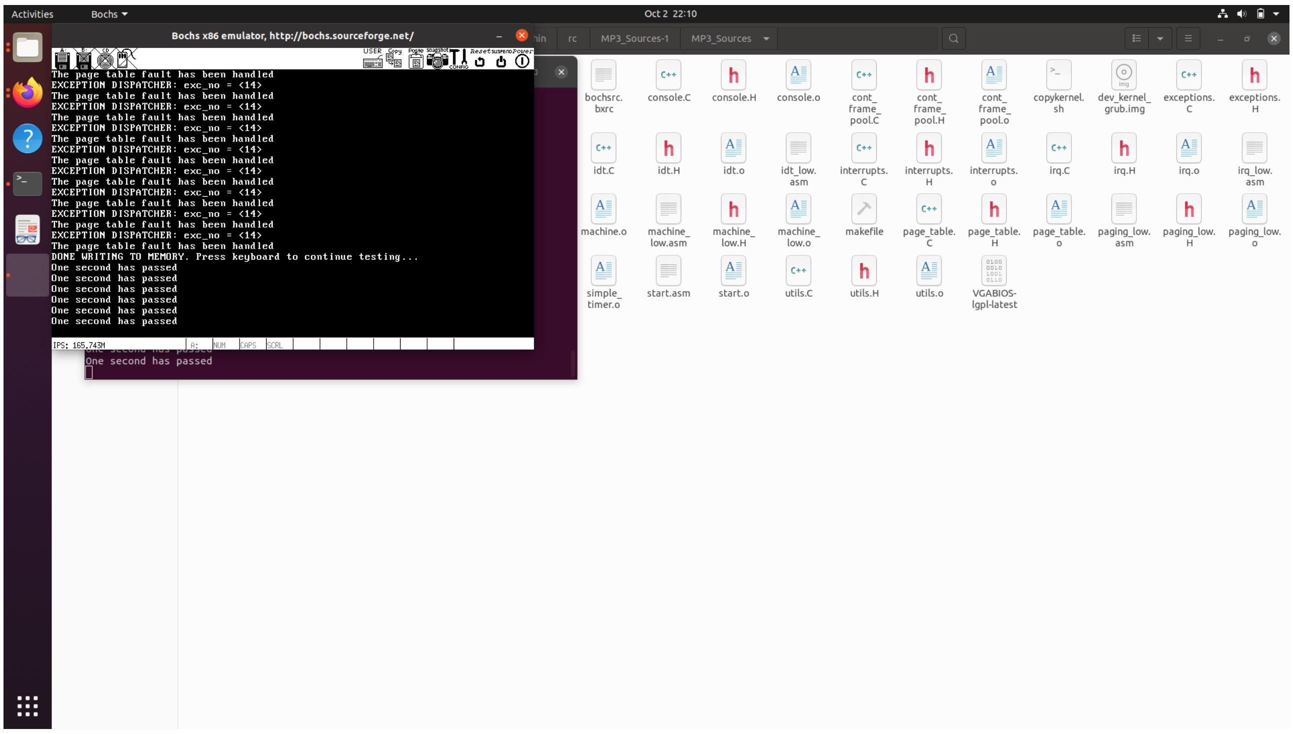
Functions Implementation:

A) **init_paging():** This initialises private variables kernel frame pool, process frame pool and the shared space value- 4mb in our case.
B) **PageTable():** This is the constructor of the class and it first maps the first 4mb of the process directly and marks the pages as present. Secondly, it marks the first entry of the page directory as present and the remaining as not present.
C) **load():** Loads the current page table into the CR3 register.
D) **enable_paging():** Marks the CR0 register appropriately and paging enabled variable is also set.

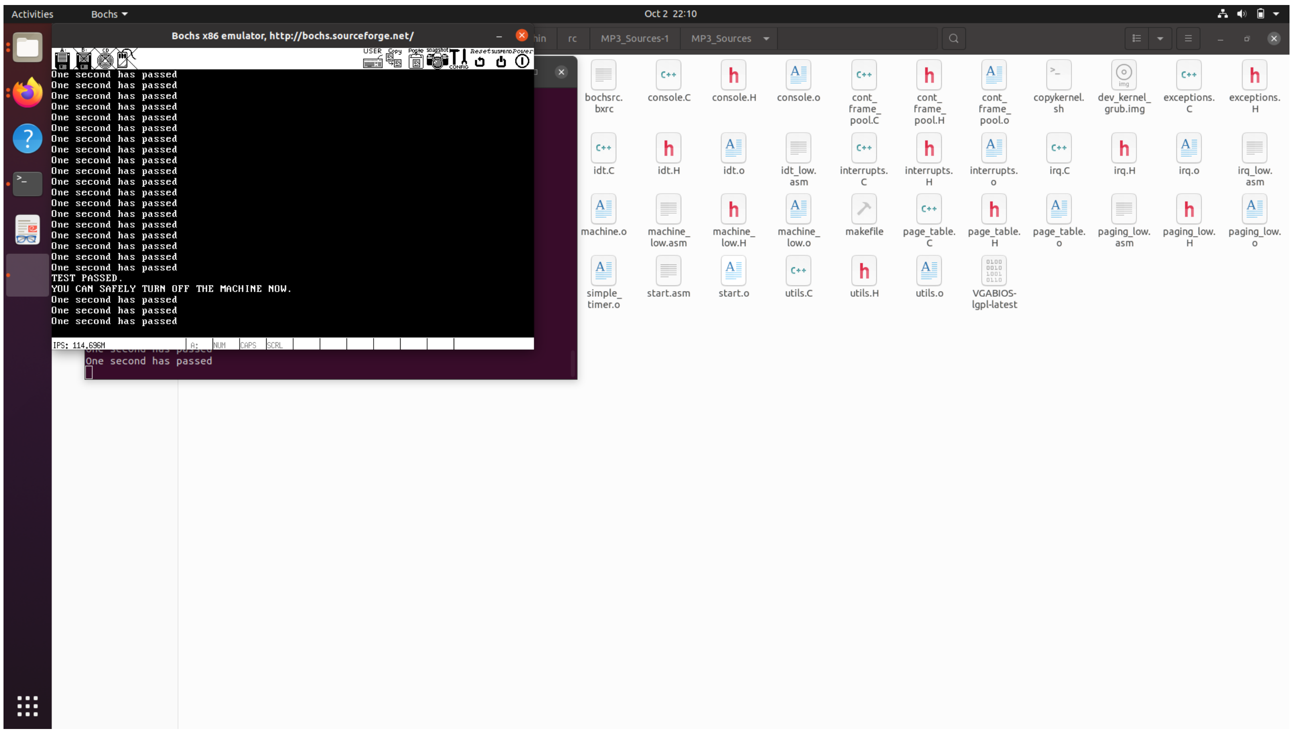
E) Handle fault(): This function is automatically called by the hardware when there is a fault. We check if the page directory is mapped. If not we allocate a frame and set the corresponding bit. Similarly the page table is also checked for the same and a frame is allocated and state is modified if there was no frame allocated from before.

Results:

Below are the screenshots of successful running:



```
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
EXCEPTION DISPATCHER: exc_no = <14>
The page table fault has been handled
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
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
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
```