CSCE 611: Operating Systems

Fall 2022

Machine Problem 3: Page Table Management

Name: Rajesh Satpathy

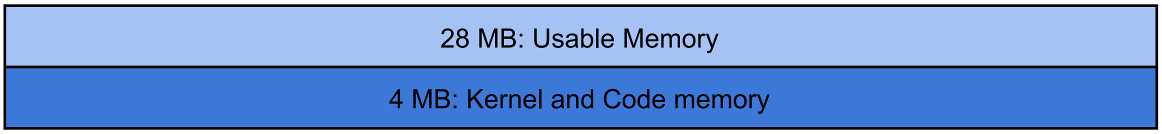
UIN: 931006158

Goal:

Initialize a paging system and page table infrastructure for a single address space, with an eye towards extending it to multiple processes, and therefore multiple address spaces, in the future for an on demand-paging based virtual memory operating system with a given kernel.

Assumptions shared about the given Kernel: (Same as mp2)

- Total machine memory: 32 MB
- Memory Layout:



28 MB: Usable Memory

4 MB: Kernel and Code memory

- About the first 4 MB:
  - Is reserved for the kernel.
  - The first 1 MB address space is reserved for IDTs, GDTs, video memory, etc.
  - The address range from 1MB to 2MB consists kernel code and stack space.
  - The kernel frame pool that is implemented in mp2 assignment is located between 2 MB an 4MB.

Scope

- This machine problem is limited to a single process → single address space.
- Multiple address spaces will be supported later

Implementation:

The paging system is implemented within page_table.C, containing the following functions:

- **Constructor:** Initializes a page table with a given location for the directory and the page table.
- **init_paging:** Set the global parameters for the paging subsystem.
- **load:** Makes the given page table the current table. This must be done once during system startup and whenever the address space is switched

- **enable paging:** Enable paging on the CPU. Typically, a CPU start with paging disabled, and memory is accessed by addressing physical memory directly. After paging is enabled, memory is addressed logically.
- **handle_fault:** Handles page faults when the page requested is not present in page table or if page is not present at index. Reads the cr2 register to get the fault.

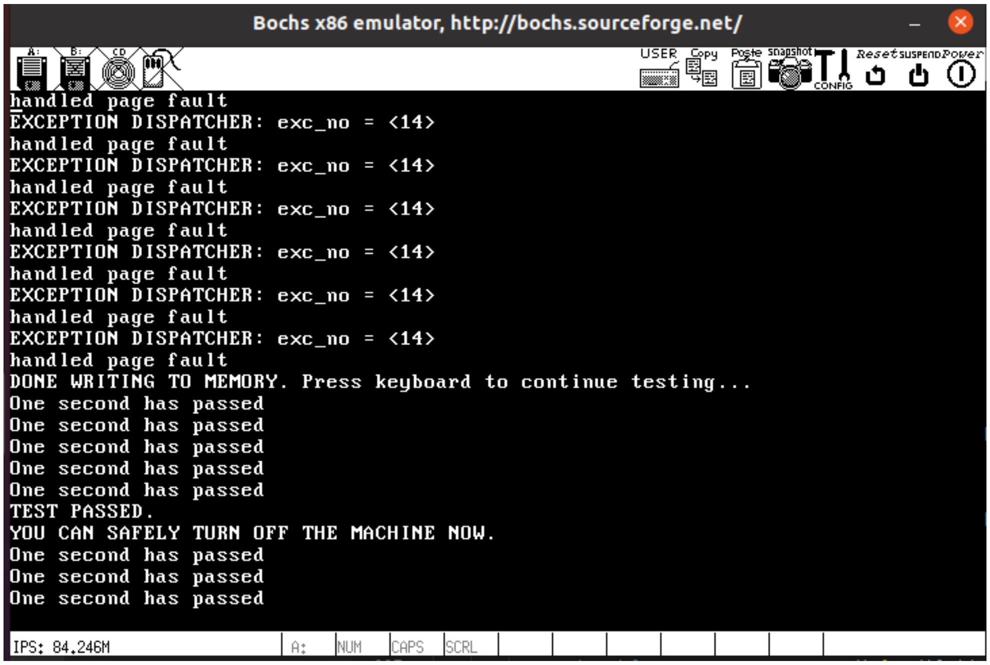
Design Decisions:

The design decisions are made based on as per the shared http://www.osdever.net/tutorials/view/implementing-basic-paging and http://www.osdever.net/tutorials/view/memory-management-1

Page fault handling is implemented based on the documentation provided in the question and page miss conditions. The steps to handle page fault are as follows:

1. Check if the page is present in memory, by checking the error code in the register, passed in the function. If the page is not present (err code = 000, 010, 100, 110), go to Step 2, else go to step 6 because there is a protection fault (err code = 001, 011, 101, 111).
2. Read the cr2 register and get the *page table index* and *page number* by bit operations.
3. Check if the page is present in the index. Go to Step 4 if the page entry at the page index is empty, else go to step 5.
4. Create a new page table entry and initialize the page table and its indices. Finish and return.
5. Page table is present at entry, but page table entry is missing. Set bits for the page entry. Finish and return.
6. Print the type of protection fault and return.

Output from testing:



```
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
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
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
```