

## CSCE 611

### Machine Problem 4: Page Table in Virtual Memory: Virtual Memory Allocator

Submitted By – Vaibhav Rawat

UIN: 626008171

Main task in this assignment was to implement page table in virtual memory. Basically, the hierarchical two - level page table including page directory and page table is managed in process memory pool which allows a larger number of page tables for processes. When page table and page directory were in kernel memory pool, since kernel memory pool was directly mapped we didn't had to handle virtual memory addresses for accessing page table and page directory. But once page directory and page table are shifted to process memory pool we need to issue logical addresses to access them. For this, a technique called as Recursive Page Table lookup is employed. Basically, in this technique the last entry in page directory and page table points to the head of page directory and page table respectively. So once a page fault occurs, this is leveraged by first extracting page directory entry number from fault address (first 10 bits) and page table entry no. (next 10 bits). To access the page directory, simply construct the virtual address as `1023|1023|page_dir_index`. Since last entry (1023), points to page directory itself so when MMU reads this address using the first 10 bits and next 10 bits it will refer to the page directory itself. And then `page_dir_index` is used to refer to the entry within the page directory and we can update/read it. Similarly for page table, we construct virtual address as `1023| page_dir_index| page_table_index`. MMU from first 10 bits points to `page_dir`, next 10 bits are used to go to respective page table and lastly using `page_table_index` we read/update the entry in page table.

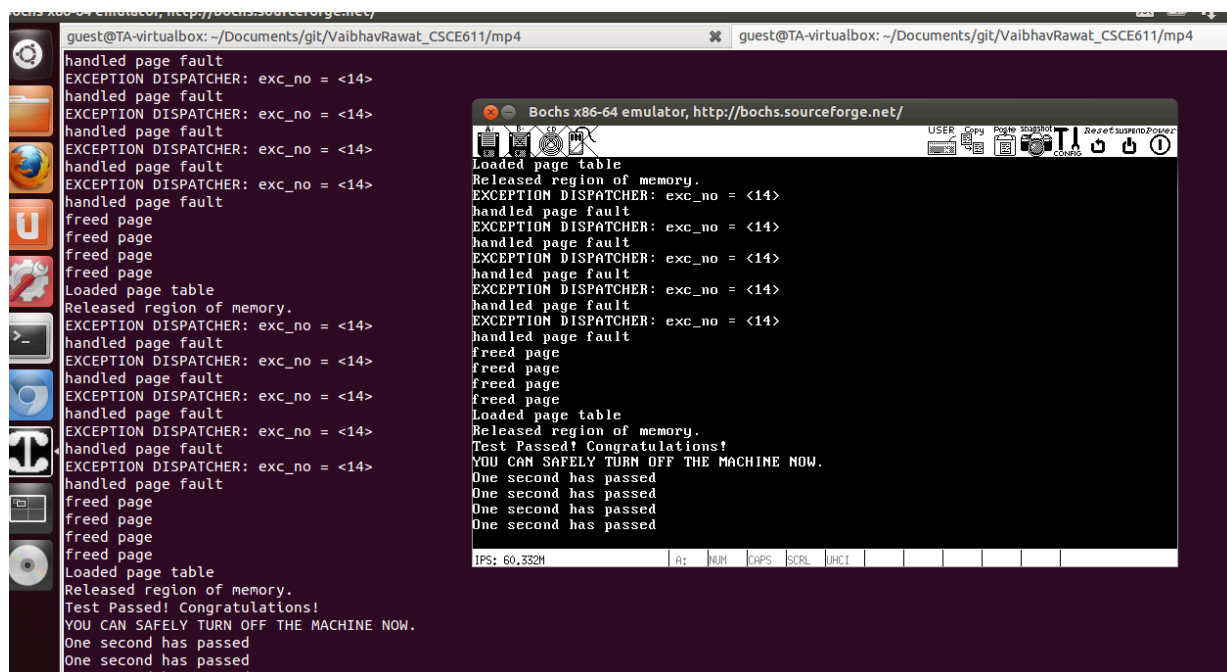
For supporting the virtual memory pools, I implemented a register pool method. Register pool method allows pools like for code and heap to be registered with process memory pool, I have put a max limit on number of pools that can be registered otherwise exception is thrown. Once registered requests for memory from these virtual pools using new operators gets frames from physical process pool updating page tables/page directories.

Virtual memory pool constructor takes in a base address, size and associated process memory pool and page table and saves local copies of each. Registers itself to the associated page table. It works by creating a regions table for all the requests for memory from this virtual pool. At this point it behaves in a lazy manner and does not allocate memory requested for the virtual pool.

Once some memory is requested, virtual memory pool's allocate method just creates a new region for this request in table storing start address and size.

If this memory is referenced, then a page fault happens, which checks if this address maps to same registered virtual memory pool and address is legitimate by checking memory boundary for the respective virtual pool. Then frame gets allocated for the request using Recursive Page Table Lookup mechanism.

Finally, when we call delete operation on a region, release method of virtual pool loops to find the respective region by checking start address, to which memory maps to. Then it loops to free all the pages associated with that regions using free\_page method from page table. Free\_page method in page table first finds the frame no. by looking up page no. in page table and then release that frame no. Release method of virtual pool also updates its region table to delete the corresponding region entry that has been released. Also TLB is flushed by reloading the CR3 register.



```
guest@TA-virtualbox: ~/Documents/git/VaibhavRawat_CSCE611/mp4
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
freed page
freed page
freed page
freed page
freed page
Loaded page table
Released region of memory.
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>
freed page
freed page
freed page
freed page
freed page
Loaded page table
Released region of memory.
Test Passed! Congratulations!
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
```