MP4: Page Manager 2

Dhanraj Murali
UIN: 734003894
CSCE611: Operating System

Assigned Tasks:

Main: Completed.

System Design:

1. Total memory size = 32MB
2. Memory reserved for kernel = 4MB
3. Memory partition to be used by kernel = 2MB to 4MB
4. Memory partition to be used by process = Above 4MB
5. All page directory and page tables exist in the process memory pool.

Code Description:

I. Recursive Page Table – Lookup, last entry of the page directory and page table pointing to one's own self triggering a recursive page lookup while handling page faults.

II. Generic virtual memory pool allocator, release functionality. The virtual memory pools/regions are followed by storing their starting address and corresponding size in 2 different arrays defined within a struct.

III. VM Pool is initialized with a free size indicator and free address pointer. Whenever a frame or region is allocated, the free size is updated accordingly in allocate and release functions by incrementing or decrementing the free size.

Files modified:

a. page_table.H
b. page_table.C
c. vm_pool.H
d. vm+pool.C
e. kernel.C – modified to add some debug points. Reverted to original file.

Note: cont_frame_pool solution provided in piazza is used here as I was getting issues.

All changes made as part of MP1 bonus to enable GDB integration in the following files are also included in this repository as well. GDB is disabled.

a. Makefile
b. bochsrc.bxrc

Function Description:

a. PageTable() - Constructor

Logic Used:

Constructor creates a page directory by getting a frame from the kernel memory pool using *get_frames* function in "cont_frame_pool.C". Once page directory is allocated a frame, all entries are initialized and write bit is enabled. For the shared memory space, a page table is created. A frame is allocated using *get_frame* function and then all entries of the page table are initiated with write and present enabled. The corresponding PDE is updated in page

directory with the first address of the page table and write/present is enabled.
Constructor is also updated to initialize the VM pool with null values upto the upper bound. Then the page directory address itself is stored as the last entry of the page directory enabling recursive page table lookup.

b. handle_faults

Logic Used:

Here we have 2 types of page faults.

I. Page table is present but page table entry is not present. So, page is faulting in page table. We just have to get a frame for the required page and update the page table with the address and enable the write/present/user bits.

II. Page table is faulting in page directory. Now we have to get a frame for a PDE and update the write bits. Then initialize the entire page table as user and write bit enabled. Update the page directory with the address of the new page table. Now get a page for the PTE corresponding to the faulting address and update the page table with it.

In this function we try to manipulate the faulting address in such a way that we are able to extract the 3 parts of the address.
The address is separated into 3, 10 bits for PDE, 10 bits for PTE and 12 bits for the offset.
Bit manipulation is done to extract the required form of addresses for creating page table by appending 1023 at the front to enable recursive page lookup.

c. **VMPool::allocate**

Logic used:

Initially, we are finding the exact number of frames required. Once it is identified, we need to assign the starting address by identifying the address where the last region ends. If the vm pool list is empty, just leave the first page for page table and then use from the 2nd available address. Then update the 2 arrays as part of the struct with the corresponding starting address and the size and update the pool count in the current pool pointer variable.

d. **VMPool::release**

Logic Used:

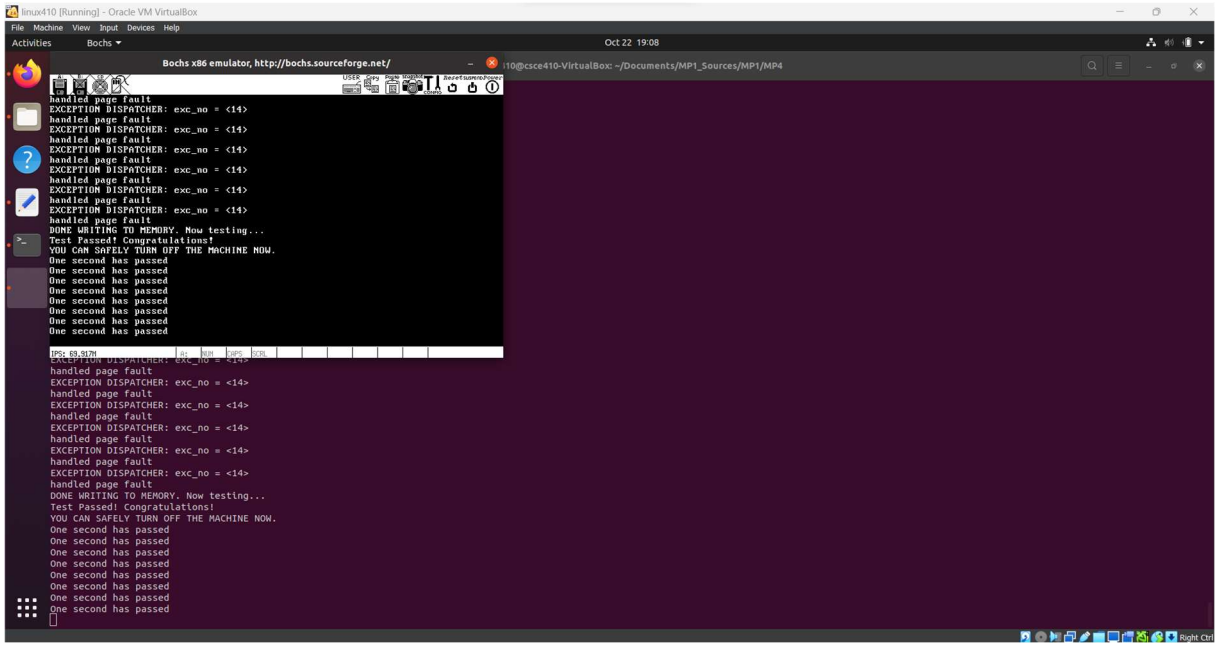
Using start address, find the index of the region by iterating over the 2 arrays. Once size is found, we will be able to identify the number of frames allocated for the region. Once that is found we can just iterate from starting address till the size and then call the free page function. Also, we need to remove the pool from the 2 arrays used to track the pools, this is done by left shifting all contents of array from the location of the target pool.

e. **VMPool::is_legitimate**

Logic Used:

Check if the address provided as input lies between the start and end of the region by using the start address and start address + size of the region

Testing:


linux410 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Bochs
Oct 22 19:08
Bochs x86 emulator, http://bochs.sourceforge.net/
10@csce410-VirtualBox: ~/Documents/MP1_Sources/MP1/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
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Now testing...
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
One second has passed
One second has passed
One second has passed
One second has passed
IPS: 69.917M
Right Ctrl