CSCE 611 Operating Systems Summer 2017

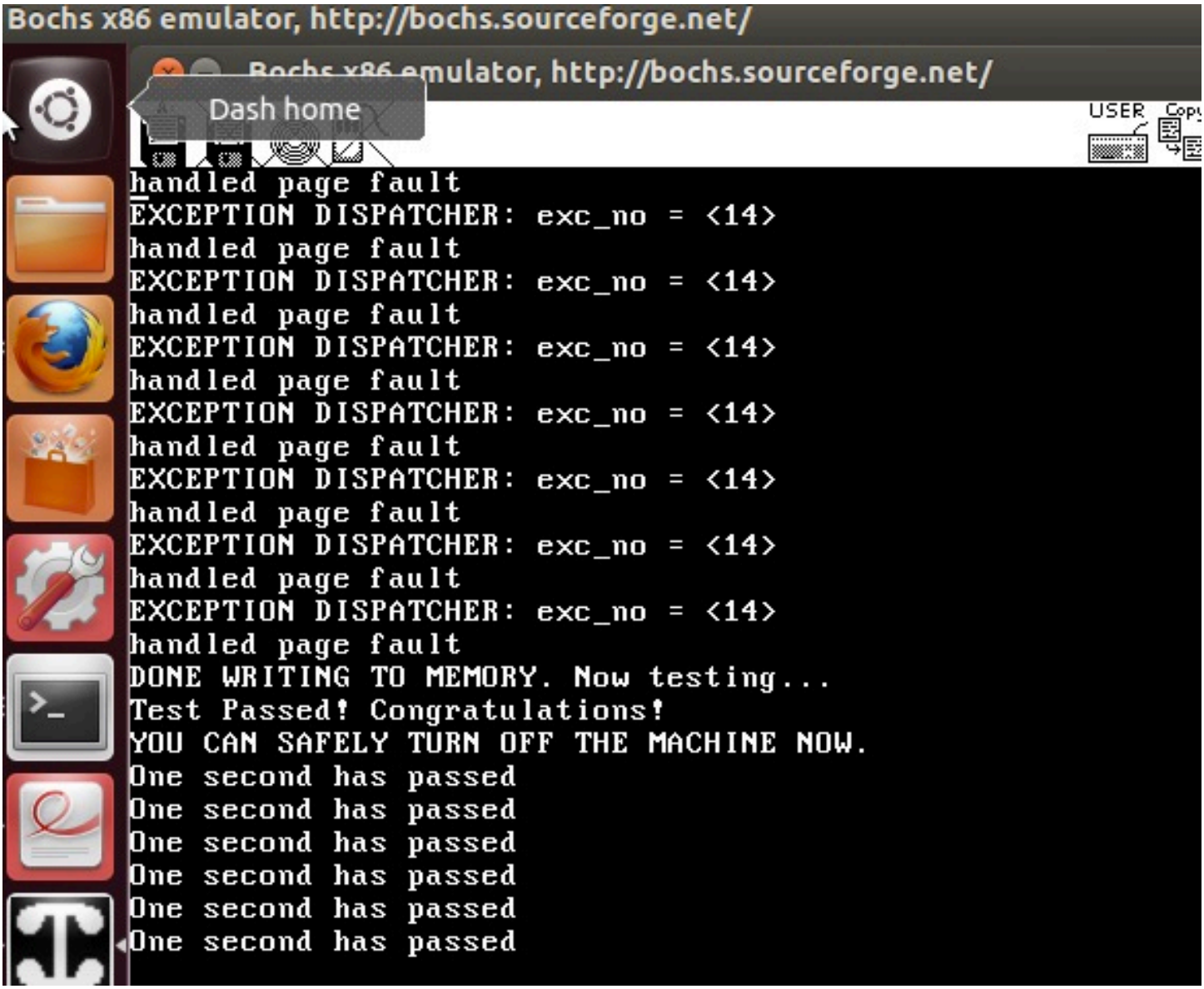
825008857 Chaiwei Chang

MP4

Design

For Part I of this MP, I extended my previous page table manager to handle pages in virtual memory. We use the trick of "recursive page table lookup" to manipulate the last frame of page directory content which will point back to the head of page directory entry itself again. This time instead of getting memory from kernel frame pool, we allocate directory frame and the page table page frames from the process frame pool and they are no longer directly map. I first implemented the register_vmpool method which takes a vm_pool object and puts it into a global array which I have declared the maximum size of 10. In this MP, we are dealing with real memory by new and delete the object, not in stack anymore. Since so far the register_vmpool array accepts at most 10 vm_pools, my program will console write out the error message if someone try to ask for more than that. And then I Implementing free_page method which simply deletes the corresponding frame and call the memory pool manager to release the page. Here comes the hardest part of part1 and part2 -- updating handle_fault. This method handles address virtually after enabling the page_table. It first checks if the address is valid as we do in the previous homework but adding additional checking of registered vm_pools array whether it is within the range of some vmpools or not. Followed by allocating a frame for the page or for the entry virtually, here we need to use different bit manipulate operator than previous. Because we will use the first 10 bit to 1023 directly to achieve the recursive mechanism and add two bit of (00) to the end, so you can take a look at my code for logical detail.

Finally, for the part 3, I tried to implement vmpool. The idea is to allocates regions of different sizes. Each region consists of two main information and I store them as a structure: (1) base address; (2) the exact size of the region. We are asked to ignore the fact that the function allocate allows for the allocation of arbitrary-sized regions and always allocate multiples of pages. Whenever a virtual memory pool releases a region, I will call the release_frames method to notify the page table that the pages can be released. Overall, my first version tries to keep the implementation of the allocator simple. While constructor initializes the variables, the allocator creates region. Regions in this case are similar to a linked list but in the form of array. The end of each region is the beginning of another. Referring to Is_legitimate method, it just simply checks if the address is within the boundaries for all pools. Note that this method will be called in page_fault handling process make it really easy to identify the address satus. When a request of allocating (new) is made, allocates try to find the nearest big-enough region. On the other hand, release change the availability variable of a region and reconstruct the array. In the future, maybe we can design a more complicate system that will combines regions if possible. Last but not least, I wrote some command on my code, but if grader of TA have any question about my logic feel free to contact me at hunkywei@tamu.edu


Bochs x86 emulator, http://bochs.sourceforge.net/
Bochs x86 emulator, http://bochs.sourceforge.net/
Dash home
USER
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
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