Assignment 1: Interrupt handlers

January 28, 2018

The goal of this assignment is to get familiar with interrupt handlers. An interrupt descriptor table (IDT) is a sequential array of descriptors (max 256). The size of each descriptor is 64-bits. The first 32 descriptors are reserved for exception handling and the others are reserved for interrupts. A descriptor corresponding to a vector `x` contains the address of the target handler(routine), which is called on every occurrence of interrupt vector `x`. Apart from containing the address of target interrupt handler (32-bits), the rest 32-bits of the descriptor contain the `cs` segment selector and other flags. Interested readers may refer to page-197, fig 6-2 of Intel Software developer manual (volume 3). For this assignment, we are only interested in the address of the target interrupt handler (In Linux `cs` segment base is always set to zero).

`struct idt_entry` in main.c represents an IDT entry. Here, `lower16` and `higher16` correspond to lower 16-bits and higher 16-bits address of target interrupt handler. `sidt` instruction on X86 hardware takes a 6-byte memory location as input and stores the size and base address of the current IDT table in it. Routine `imp_store_idt` (main.c) takes a pointer to a 6-byte memory location and returns the address and size of the current IDT. Similarly, `imp_load_idt` loads a new IDT whose size and base address are passed to this routine (For this assignment you might not need this routine).

In this assignment, you need to modify the kernel module to count the number of divbyzero exception. First entry (0th index) in IDT table corresponds to divbyzero exception. To count the number of divbyzero exceptions you need to overwrite the original `divbyzero` handler with `__wrapper_divbyzero` (wrapper.S). You might also want to look at indirect jmp instruction (`jmp *MEM`), which takes a 4-byte memory location as input and jmp to the program counter stored in that location. For example: `jmp *orig_divbyzero`, will jump to an address which is stored in `orig_divbyzero` (main.c) variable.

0.1 Compilation and running

The source code contains two folders kernel and user. As the name suggests, kernel runs as a part of kernel address space and user runs as a part of user address space.

To compile:
cd kernel && make
cd user && make

To run:
cd kernel && sudo ./load loads the kernel module.

`start_module` (main.c) is called automatically when the module is loaded. If you run ``dmesg'' at this point, you should see "aos module loaded successfully". `dmesg` prints the kernel log (i.e., log generated using printk, etc.) on terminal. `make clean` removes the executables and temporary files.

User programs can make a call to the kernel module for different services. In this example, if a user program does the ioctl system call, `device_ioctl` (main.c) routine is called. `Ioctl` takes an identifier and a 32-bit argument (user-mode addresses can also be passed as an argument). Depending on the identifier the driver may do different tasks (see switch case in `device_ioctl`). The user folder contains three programs. You need to run them after loading the kernel module. These programs do different kind of ioctls. e.g., start does `START_DIVBYZERO` (see `device_ioctl`), stop does `STOP_DIVBYZERO`, and stats does `STATS_DIVBYZERO`. STATS_DIVBYZERO copies (using `copy_to_user`) the value of `total_count` into the user-space address passed by stats.

0.2 Turn in:

In this assignment, you need to implement `start_tracking_divbyzero` and `stop_tracking_divbyzero` routines. `start_tracking_divbyzero` routine modifies the interrupt handler to increment the `total_count` variable before calling the original `divbyzero` handler. `stop_tracking_divbyzero` routine restores the original `divbyzero` handler.

0.3 Environment:

You need to do this assignment on a uniprocessor 32-bit Linux virtual machine. To get the virtual machine (LinuxVM.vhd) raw disk, run:
`scp aos@192.168.1.161:/home/iiitd/LinuxVM.vhd .`

Use "aos" as the password to proceed.

To boot from this virtual disk follow the steps below.

- Create a new Virtual Machine in VirtualBox.
- Choose type Linux and Version Linux 2.6/3.x/4.x (32-bit).
- Click next until it asks you about the Hard Disk.
- Select the option to use an existing virtual hard disk file.

- Give the path to LinuxVM.vhd.
- Start the virtual machine.
- Use scaled mode for larger display.
- Use "aos" when you are prompted for the password.

VirtualBox is installed in the L23 (library 2nd floor, 3rd lab) on Linux hosts.

0.4 How to submit.

A copy of the assignment is already present in the /home/aos folder. Before implementing anything first make a local branch (`git checkout -b [your_branch_name]`). For submission send a diff to the master branch. Run ``` ``git diff master > submit.txt'' ``` to redirect your changes to the submit.txt file. Submit the submit.txt file (please do not submit the entire folder).