CS 0449 – Device Driver Lab

Background

Standard UNIX and Linux systems come with special files like /dev/zero, which returns nothing but zeros when it is read, and /dev/urandom, which returns random bytes. These are not actual files in your hard disk. They are devices that are exposed by the Operating System to the end user and applications using the file interface, comprised of open(), read(), write(), lseek(), and close() system calls, as we learned in class. The file interface allows applications to interact with any hardware device uniformly, without having to know the details of each device, be it a keyboard, a sound device, or a camera. All files located under the /dev/ directory work in this way and are called device files.

Of course, someone must know how to interact with the device hardware and that someone is the Operating System, which is charge of managing all hardware devices. However, the OS cannot be expected to know how to interact with all devices in the world – new devices are born daily! Hence, device manufacturers write the code to interact with their own devices (which only makes sense) called device drivers, and these device drivers are distributed with the devices. Device drivers come in the form of plug-and-play modules that can be inserted and removed from the OS on-the-fly. These modules are also called kernel objects since they are object files dynamically linked with the kernel.

The purpose of this lab is to write a device driver of your own, insert it into an OS, and test it. Now since device drivers become part of the OS, there is a danger that a badly written device driver may compromise the entire OS. Hence, we are not going to install the drivers to the thoth machine. Instead, we are going to install a virtual machine monitor on your individual PCs and run a Debian Linux OS on top of it inside a virtual machine. We are going to use this as our test bed.

Step 1: Try Reading From Existing Devices on Thoth

1. On `thoth.cs.pitt.edu`, try reading from the two device files explained above by running the following commands, in sequence:

```
cat /dev/zero | xxd | head
cat /dev/urandom | xxd | head
```

   Here is an explanation of the different components of the command line:

   - `cat`: A Unix utility that reads and dumps the contents of a file to stdout.
   - `xxd`: A Unix utility that reads binary from stdin and converts it to human readable hex format, which is printed to stdout.
   - `head`: A Unix utility that reads text input from stdin and outputs just the first 10 lines.

These three programs are linked together in the command line using the ‘|’ character. The ‘|’ is a Unix pipe and it ‘pipes’ the stdout of the left command to the stdin of the right command. It is similar to file redirection in many respects except now you are redirecting to another program. The output of each device is piped to xxd to convert binary numbers to human readable format and then is piped to head to truncate it to just 10 lines (which otherwise would have been an infinite stream of numbers). Notice that the output of `/dev/urandom/` is random every time you run it. You are going to use a similar methodology to test the output of the device driver you are going to be writing for your project.

Step 2: Install the Virtual Machine on your PC

2. Download and install VirtualBox: http://www.virtualbox.org. This would be the virtual machine monitor, the program that actually runs the virtual machine.

3. Download and install Vagrant: https://www.vagrantup.com/. This would be used to download the image of the virtual machine OS.

4. Next open a terminal (cmd for Windows users) and change to the directory where the configuration file for your image will be stored. Best to create a new directory for this purpose.

5. Run the following commands (will download a Debian Linux image and install it to VirtualBox):

```
vagrant init puphpet/debian75-x32
vagrant up
vagrant halt
```

   * Some Macs have a problem with running vagrant. If you have a problem, try the following:

```
sudo mv /opt/vagrant/embedded/bin/curl /opt/vagrant/embedded/bin/curl_
```

   The problem is that the curl distributed with vagrant is incompatible with some Mac systems. Once you do the above, the Mac machine will revert to its original curl, under /usr/bin/.

6. Open VirtualBox and you should see a virtual machine in the left panel. It should be labeled as ‘Powered Off’ at this time since we halted it. Double click to start the VM.

7. Login with 'vagrant' and password 'vagrant' after virtual machine boots up.

8. Next, change the root password to allow easy login (type root in the password prompt):

```
sudo su
passwd
```

9. Restart the virtual machine using the following command:

```
reboot
```

   And login with 'root' and password 'root'.

Step 3: Build Hello World Device Driver

10. We now need to download the hello world device driver package from thoth using scp:

```
scp USERNAME@thoth.cs.pitt.edu:/u/SysLab/shared/hello_dev.tar.gz ./
```

11. Unpack the file:

```
tar xvfz ./hello_dev.tar.gz
```

12. Change into the directory:

```
cd hello_dev
```

13. Build the kernel object. This should produce the kernel object hello_dev.ko in the directory.

```
make
```

14. Load the driver into the OS using insmod:

```
insmod hello_dev.ko
```

15. We now need to make the device file in /dev. First, we need to find the MAJOR and MINOR numbers that identify the new device:

```
cat /sys/class/misc/hello/dev
```

16. The output should be a number like 10:57. The 10 is the MAJOR and the 57 is the MINOR.

17. We use `mknod` to make a special file. The name will be `hello` and the 'c' says it will be a character device (we will learn what that is later). The 10 and the 57 correspond to the MAJOR and MINOR numbers we discovered above (if different, use the ones you saw.)

```
mknod /dev/hello c 10 57
```

18. We can now read the data out of `/dev/hello` with a simple call to cat:

```
cat /dev/hello
```

    You should see "Hello, world!" which came from the driver. **Show the TA.**

19. Clean up by removing the device file and unloading the module:

```
rm /dev/hello
rmmod hello_dev.ko
```