Lab 11: Kernel Building (Time Device Driver)

In this lab, you are going to create a time device driver almost from scratch. The process will be almost the same as previous lab where the kernel module will be build in `thoth.cs.pitt.edu` (`cs449.cs.pitt.edu`) and then transferred and install into the virtual machine (`tty.qcow2`). You also going to write a user program to read the time from your time device driver.

1 Time Device Driver

Imagine that your computer is connected to a physical clock (a piece of hardware). You are writing a device driver that allows user to read the current time from this clock. What user needs to do is simply open this device file (`/dev/mytime`) using the system call `open()` and call `read()` to read at least 8 bytes. What user gets will be a string **not null terminated** in the format of `"hh:mm:ss"` (hours, minutes, and second) that represents the current time. For example, if user use the following series of statement:

```
int fd;
char buffer[10];
fd = open("/dev/mytime", O_RDWR);
read(fp, buffer, 8);
```

The buffer will be filled with 8 characters represents current time. Too bad, we do not have the actual physical clock hardware, so we are going to use the system clock. The start code (`mytime_dev.c`) already have a series of statements that obtain the number of second since January 1, 1900 at 00:00:00 GMT for you. You simply have to convert it into hour, minute, and second.

2 Starter Program and `Makefile`

First, let's get the starter file (`mytime_dev.c` and the `Makefile` using the following steps:

1. Login to `thoth.cs.pitt.edu` (`cs449.cs.pitt.edu`), and go to your `/u/SysLab/USERNAME` directory using the following commmand:

   ```
   cd /u/SysLab/USERNAME
   ```

   and replace `USERNAME` by your username.

2. Create a directory named `lab11` and go there

   ```
   mkdir lab11
   cd lab11
   ```

3. Copy the file `mytime_dev.c` to your directory using the following command:

```
cp /afs/cs.pitt.edu/usr0/tkosiyat/public/cs0449/lab11/mytime_dev.c .
```

4. Copy the file `Makefile` to your directory using the following command:

   ```
   cp /afs/cs.pitt.edu/usr0/tkosiyat/public/cs0449/lab11/Makefile .
   ```

   and make sure it looks like the following:

   ```
   obj-m := mytime_dev.o

   KDIR  := /u/SysLab/shared/linux-2.6.23.1
   PWD   := $(shell pwd)

   default:
           $(MAKE) -C $(KDIR) M=$(PWD) modules
   ```

3 What to do (Part I)?

Use an editor of your choice to open the file `mytime_dev.c`. For this lab, you are going to modify the function `mytime_read()` only. **Do not modify any other part of the program**. When this device is read, it will write an array of character in the form of `"hh:mm:ss"` to the buffer passed by the system call `read()`. Inside the function `mytime_read`, you will see the following:

```
static ssize_t mytime_read(struct file * file, char * buf, size_t count,
                           loff_t *ppos)
{
    struct timespec ts;        // Time structure

    getnstimeofday(&ts);        // Get time

    return 0;
}
```

This function will be called when user call `read()`. Note that the system call `read()` needs three arguments as shown in its signature below:

```
ssize_t read(int fd, void *buf, size_t count);
```

The first argument is the file descriptor. This is the file descriptor that is returned by the system call `open()` which is used to open the device driver file. For this lab, the file that you will open is `/dev/mytime`. The second argument is the buffer to store values. The third argument is the number of bytes to read.

When a program call `read()`, the value of the argument `count` of the function `mytime_read()` will be initialized to the same value of the argument `count` of `read()`. This indicates how my bytes a program trying to read from the device driver. The function `mytime_read()` can use this value to decide what to do. For example, some device driver require a user program to read exact a number of bytes or at least a number of bytes. In our case, user must read **at least** 8 bytes.

The argument `ppos` is a **pointer** to an integer represents the current position. **Note** that it is a pointer variable. If you want to know the value stored at that address, use dereference (*). When `mytime_read()` is called for the first time `*ppos` will be 0. The value stored at `ppos` can be updated by the function `mytime_read()` if it is needed to be updated. The next called to the function `mytime_read()`, the value stored at `ppos` will be the new value if it was updated by the previous call. The function `mytime_read()` can use this value to keep track of the current position of the data stream (index) if it has a long stream of data and required to be read multiple time.

The argument `buf` is a **pointer** to user's buffer. **Note** that it is located in user address space, not kernel address space. To copy data from kernal address space to user address space, you need to use the function `copy_to_user()`. which has the signature as shown below:

```
unsigned long copy_to_user(void __user * to, const void * from, unsigned long n);
```

The argument `to` is the address in the user address space. The argument `from` is the address in the kernel address space. The argument `n` is the number of bytes to copy from kernel address space to user address space. For example, if you have an array of character named `time_string` and you want to copy to user address space point by `buf` for 8 bytes, use the following command:

```
copy_to_user(buf, time_string, 8);
```

Lastly, when user call `read()`, it returns the number of bytes read, this will be the same number that the function `mytime_read()` returns at the end. This return value generally used by user to see how many bytes of data it gets.

From the given code, after the function `getnstimeofday()` is called, the structure variable named `ts` will contain information about the current time. The only component of this structure that you are going to use is called `tv_sec`. That is `ts.tv_sec` is the number of second since January 1, 1900 at 00:00:00 GMT. Note that we are in eastern time zone which is 5 hours slower than GMT. So, your job is to convert `ts.tv_sec` into hour, minute, and second. We do not care about month, day, or year. So, the mathematics for calculating hour, minute, and second should be straightforward. Then create a string in the form of `"hh:mm:ss"` which will be copy to user address space when user call `read()`. Note that `"hh"` can be between `"00"` to `"23"`, `"mm"` can be between 00 to `"59"`, and `"ss"` can be between `"00"` to `"59"`. Lastly, the requirement for this device is as follows:

- User must read at least 8 bytes
- Once it is read, user cannot read any more. In other words, the successive call to `read()` after the first read results in return value -1 (`-EINVAL`).

Lab 11: Kernel Building (Time Device Driver)

Note that you should look at the code from previous lab `hello_dev.c` and use it as a guideline to create your program.

3.1 Build your Kernel Module

These steps are almost identical to the previous lab.

1. Build the kernel object using the following command:

   ```
   make ARCH=i386
   ```

   The `ARCH=i386` is important because we are building a 32-bit kernel on a 64-bit machine. If all went well, you will get a new file named `mytime_dev.ko` and this is your kernel module.

2. Go to your Windows, Mac, or Ubuntu machine and run `qemu` to open the virtual machine `tty.qcow2`. This is the same virtual machine used in previous lab.

3. When Linux virtual machine boots under `qemu`, login using the `root` as your username and `root` as your password.

4. Now, you need to download the kernel module that you just build in step 8 into your virtual machine using the following command:

   ```
   scp USERNAME@thoth.cs.pitt.edu:/u/SysLab/USERNAME/lab11/mytime_dev.ko .
   ```

   Replacing `USERNAME` by your username. **Note** that this is a space and a dot at the end of the above command.

5. To load the kernel module into your kernel, use the following command:

   ```
   insmod mytime_dev.ko
   ```

6. The next step is to make the device file in the `/dev` directory. First, we need to know the MAJOR and MINOR numbers that identify the new device using the following commands:

   ```
   cd /sys/class/misc/mytime
   cat dev
   ```

   The output should look like the following:

   ```
   10:63
   ```

   The 10 indicates the MAJOR number and the 63 indicates the MINOR number.

7. We use the command `mknod` to make a special file. The name will be `mytime` and it is a character device. The 10 and the 63 correspond to the MAJOR and MINOR numbers we discovered above (if different, use the ones you saw.) To create the device file, use the following commands:

   ```
   cd /dev
   mknod hello c 10 63
   ```

8. We can now read the data out of `/dev/mytime` with a command below:

   ```
   cat /dev/mytime
   ```

   and the output should be

   ```
   22:21:36
   ```

   which came from the driver.

9. To unload the module from the kernel, use the following commands:

   ```
   rm /dev/mytime
   rmmod mytime_dev.ko
   ```

   Note that if you want to load this module again, you do not need to compile it. Simply follow step 5 to 7.

4 What to Do (Part II)?

For this part, go back to `thoth.cs.pitt.edu` in the directory `/u/SysLab/USERNAME/lab11` and create a program named `readTime.c` that open the device file `/dev/mytime` and use the system call `read()` to read at least 8 bytes. Then display the current time on the console screen. **Note** that you should use the program `readHello.c` from previous lab as a guideline for this part. Note that you should compile your program using the following command:

```
gcc -m32 -o readTime readTime.c
```

and transfer to your virtual machine by using the following command **on the virtual machine**:

```
scp USERNAME@thoth.cs.pitt.edu:/u/SysLab/USERNAME/lab11/readTime .
```

Try your program on your virtual machine. If it works properly, it should display the current time string in the form of `"hh:mm:ss"`.

What to Hand In

First, on thoth.cs.pitt.edu, go back up to our /u/SysLab/USERNAME directory:

```
cd ..
```

Now, let us first make the archive. Type your username for the USERNAME part of the filename:

```
tar cvf USERNAME_lab11.tar lab11
```

And then we can compress it:

```
gzip USERNAME_lab11.tar
```

Which will produce a USERNAME_lab11.tar.gz file.

If you work on cs449.cs.pitt.edu (thoth) you can skip to the next section. **If you use your own machine, you need to transfer the file to cs449.cs.pitt.edu first**. This can simply be done by a command line. For example, assume that your username is abc123 and you are in the same directory as the file abc123_lab11.tar.gz. To transfer the file to cs449.cs.pitt.edu use the following command:

```
scp abc123_lab11.tar.gz abc123@cs449.cs.pitt.edu:.
```

The above command will copy the file to your home directory in cs449.cs.pitt.edu. If you want to copy it to your private directory, use the following command:

```
scp abc123_lab11.tar.gz abc123@cs449.cs.pitt.edu:./private/.
```

Copy File to Submission Directory

We will then submit that file to the submission directory:

```
cp USERNAME_lab11.tar.gz /afs/cs.pitt.edu/public/incoming/CS0449/tkosiyat/sec1
```

Once a file is copied into that directory, you cannot change it, rename it, or delete it. If you make a mistake, resubmit a new file with slightly different name, being sure to include your username. For example USERNAME_lab11_2.tar.gz. **Check the due date of this lab in our CourseWeb under Labs/Recitations**.