CS 0449 – Project 4: /dev/pi
Due: Sunday, August 6, 2017 at 11:59pm

Project Description
Standard UNIX and Linux systems come with a few special files like `/dev/zero`, which returns nothing but zeros when it is read, and `/dev/random`, which returns random bytes. In this project, you will write a device driver to create a new device, `/dev/pi` which returns the digits of everyone's favorite irrational number: π.

How It Will Work
For this project we will need to create three parts: the device driver, the special file `/dev/pi`, and a test program to convince ourselves it works. The good news is that I have provided a function that will generate the first N digits of pi for you. Your job is then to make a device driver, and a simple program named pi_digits that takes two command-line arguments of which digits of pi to return (start to end, zero indexed):

```
./pi_digits 0 3
3141
./pi_digits 4 7
5926
```

Implementation
Our device driver will be a character device (although it could be a block device since we can access individual bytes of pi by address) that will implement a read function and return the appropriate substring of pi back to the user.

Installation and Example
1. On `thoth.cs.pitt.edu`, login and `cd` to your `/u/SysLab/USERNAME` directory.
2. `tar xvfz ../shared/hello_dev.tar.gz`
3. `cd hello_dev`
4. Open the Makefile with the editor of your choice. (E.g., `pico Makefile`)
5. We need to setup the path to compile against the proper version of the kernel. To do this, change the line:

   `KDIR  := /lib/modules/$(shell uname -r)/build`

   to

   `KDIR  := /u/SysLab/shared/linux-2.6.23.1`

6. Build the kernel object. The ARCH=i386 is important because we are building a 32-bit kernel on a 64-bit machine.

   ```
   make ARCH=i386
   ```

7. Download and launch QEMU. For Windows users, you can just double click the `qemu-win.bat` that is supplied in the zipfile available on my website. Mac users should download and install `Q.app` and use the `tty.qcow2` disk image provided in the main zipfile. Linux users are left to install qemu as appropriate.
8. When Linux boots under QEMU login using the root/root account (username/password).

9. We now need to download the kernel module you just built into the kernel using scp:

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

10. Load the driver using insmod:

```
insmod hello_dev.ko
```

11. 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:

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

12. The output should be a number like 10:63. The 10 is the MAJOR and the 63 is the MINOR.

13. We use `mknod` to make a special file. The name will be `hello` 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.)

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

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

```
cat /dev/hello
```

15. You should see "Hello, world!" which came from the driver. We can clean up by removing the device and unloading the module:

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

What to Do Next

The code for the example we went through comes from:

http://www.linuxdevcenter.com/pub/a/linux/2007/07/05/devhelloworld-a-simple-introduction-to-device-drivers-under-linux.html?page=2

Read that while going through the source to get an idea of what the Module is doing. Start with the third section entitled "Hello, World! Using /dev/hello_world" and read up until the author starts describing udev rules; we are not using udev under QEMU.

When you have an idea of what is going on, make a new directory under your `/u/SysLab/USERNAME` directory called `pi_driver`:

```
mkdir pi_driver
```

Copy and rename the `hello_dev.c` from the example, and copy over the `Makefile`. Edit the `Makefile` to build your new file. Change all the references of "hello" to "pi_driver".

The pi generating function can be copied from the `/u/SysLab/shared` directory:

```
cp /u/SysLab/shared/pi.h .
```

Your job is to make a read function that returns the appropriate substring of pi back (starting at *ppos and stretching out `count` digits. Notice that the `pi.h` code will give you all the digits from 3141 until some maximum you specify. You'll need to make this work.

Building the Driver

To build any changes you have made, on `thoth` in your `pi_driver` directory, simply:

```
make ARCH=i386
```

If you want to force a rebuild of everything you may want to remove the object files first:

```
rm *.o
```

Copying the Files to QEMU

From QEMU, you will need to download the driver that you just built. Use `scp` to download the driver to a home directory (`/root/` if root):

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

Loading the Driver into the Kernel in QEMU

As root (either by logging in or via su):

```
insmod pi_driver.ko
```

Making the /dev/pi Device

Like in the example, we'll need to determine the MAJOR and MINOR numbers for this particular driver.

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

and use those numbers to make the `/dev/pi` file:

```
cd /dev
mknod pi c MAJOR MINOR
```

Unloading the Driver from the Kernel in QEMU

As root (either by logging in or via su):

```
rmmod pi_driver.ko
```

Then you can remove the `/dev/pi` file:

```
rm /dev/pi
```

Implementing and Building the pi_digit Program

Since thoth is a 64-bit machine and QEMU emulates a 32-bit machine, we should build with the –m32 flag:

```
gcc –m32 –o pi_digit pi_digit.c
```

Running pi_digit

We cannot run our `pi_digit` program on `thoth.cs.pitt.edu` because its kernel does not have the device driver loaded. However, we can test the program under QEMU once we have installed the new driver. We first need to download `pi_digit` using `scp` as we did for the driver. However, we can just run it from our home directory without any installation necessary.

File Backups

One of the major contributions the university provides for the AFS filesystem is nightly backups. However, the `/u/SysLab/` partition is **not** part of AFS space. Thus, any files you modify under your personal directory in `/u/SysLab/` are not backed up. If there is a catastrophic disk failure, all of your work will be irrecoverably lost. As such, it is my recommendation that you:

Backup all the files you change under `/u/SysLab` to your `~/private/` directory frequently!

Loss of work not backed up is not grounds for an extension. YOU HAVE BEEN WARNED.

Hints and Notes

- `printk()` is the version of `printf()` you can use for debugging messages from the kernel.
- In the driver, you can use some standard C functions, but not all. They must be part of the kernel to work.
- In the `pi_digit` program, you may use any of the C standard library functions.
- As root, typing `poweroff` in QEMU will shut it down cleanly.
- If the module crashes, it may become impossible to delete the file you created with mkdev in /dev. If that happens, just grab a new disk image and start over. It's why we're developing in a virtual machine as opposed to a real one.

Requirements and Submission

You need to submit:

- Your `pi_driver.c` file and the `Makefile`
- Your well-commented `pi_digit` program's source

Make a tar.gz file named `USERNAME-project4.tar.gz`

Copy it to `~jrmst106/submit/449` by the deadline for credit.