CS 0449 – Project 4: /dev/dice
Due: Tuesday, November 29, 2016, 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/dice which returns a randomly selected roll of a 6-sided die.

How It Will Work
For this project we will need to create three parts: the device driver, the special file /dev/dice, and a test program to convince ourselves it works

The test program will be a solitaire implementation of the game of yahtzee.

Driver Implementation
Our device driver will be a character device that will implement a read function (which is the implementation of the read() syscall) and returns an appropriate die roll (a 1 byte value from 1 to 6). As we discussed in class, the kernel does not have the full C Standard library available to it and so we need to get use a different function to get random numbers. By including <linux/random.h> we can use the function get_random_bytes(), which you can turn into a helper function to get a single byte:

```
unsigned char get_random_byte(int max) {
        unsigned char c;
        get_random_bytes(&c, 1);
        return c%max;
}
```

Yahtzee Implementation
Yahtzee (also known as Poker Dice) is a multiplayer dice game, with fairly simple rules. For this assignment, you will be implementing a solitaire version where a user plays by herself. I highly suggest playing the game or at least reading the rules http://en.wikipedia.org/wiki/Yahtzee

5 dice are rolled. The player can choose to keep any of the dice and is allowed to reroll any they do not want to keep. They may reroll up to twice, but at the end of the third roll, the five dice are finalized.

After the user has finalized their five dice, they may place them into one of N categories:

Event	Score
Upper Section	
Ones, Twos, Threes, Fours, Fives, Sixes	The sum of the dice with the appropriate value
Lower Section	
Three of a Kind, Four of a Kind	The total of all 5 dice
Full House	25
Small straight	30
Large straight	40
Yahtzee	50
Chance	The sum of the dice

We will not do anything special for a second Yahtzee. Note that the player can score 0 points in a category.

If the total points of the Ones, Twos, Threes, Fours, Fives, and Sixes is 63 or more, the user gets a bonus of 35 points.

```
Your roll:

3 4 1 4 5

Which dice to reroll? 1 2

Your second roll:

5 6 1 4 5

Which dice to reroll? 0

Place dice into:
1) Upper Section
2) Lower Section

Selection? 1

Place dice into:
1) Ones
2) Twos
3) Threes
4) Fours
5) Fives
6) Sixes

Selection? 5

Your score so far is: 10

Ones:                   Fours:
Twos:                   Fives: 10
Threes:                 Sixes:
```

```
Upper Section Bonus: 0
Three of a Kind:           Four of a Kind:
Small Straight:            Large Strait:
Full House:                Yahtzee:
Chance:

…
```

The game ends when all categories have been assigned a point (13 turns).

For this assignment you need to do the following:

- Write a program that allows a player to play Yahtzee
- Gets each die to display by reading one byte from the `/dev/dice` file.

Hints and Suggestions

- Use the qsort function to sort the dice for scoring. It makes finding multiples and straights much easier.
- You may write the yahtzee program using stdlib's rand() for testing, but make sure you eventually replace it with your read of `/dev/dice.`

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 and also use the `tty.qcow2` disk image provided in the main zipfile.

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 `dice_driver`:

```
mkdir dice_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 “dice_driver”.

Building the Driver

To build any changes you have made, on `thoth` in your `dice_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/dice_driver/dice_driver.ko .
```

Loading the Driver into the Kernel in QEMU

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

```
insmod dice_driver.ko
```

Making the /dev/dice Device

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

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

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

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

Unloading the Driver from the Kernel in QEMU

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

```
rmmod dice_driver.ko
```

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

```
rm /dev/dice
```

Implementing and Building the yahtzee 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 yahtzee yahtzee.c -static
```

Running yahtzee

We cannot run our `yahtzee` 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 `yahtzee` 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 `yahtzee` 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 `dice_driver.c` file and the `Makefile`
- Your well-commented `yahtzee` program's source

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

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