CS 0449 – Project 4: /dev/dice

Due: Friday, December 7, 2018, at 11:59pm

Project Description

Standard UNIX and Linux systems come with special device 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 file, /dev/dice. Reading /dev/dice returns an endless stream of random rolls of an N-sided die, where N can be designated by the user by writing to /dev/dice. We will also experiment with signals that we learned in class.

How It Will Work

For this project we will need to create two parts: the device file /dev/dice, and a program that uses the device driver to generate random die rolls. For the latter we are going to use a modified version of the RPG emulator we did for Project 1.

Driver Implementation

The skeleton code for /dev/dice has already been implemented for you. Change into your preferred directory and unzip the provided code as follows:

```
thoth $ cp ~wahn/public/cs449/dice_dev.tar.gz ./
thoth $ tar -zxvf dice_dev.tar.gz
```

/dev/dice is a character device just like the /dev/hello device we tested for our Device Driver Lab. This time, it has a write function as well as a read function. As we discussed in class, the dice_read and dice_write functions are called by the kernel in response to read() and write() system calls to the /dev/dice file. Try doing 'man 2 read' and 'man 2 write' if you are unsure of the system call interface. The dice_read function should fill the provided buf with count number of bytes, where each byte is a randomly generated die roll. Refer back to the /dev/hello lab on how to do this. 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;
}
```

By default, /dev/dice rolls a 6-sided die when initialized. The user can later change the number of sides in the die by writing to /dev/dice. The dice_write function should set the number of sides to the last byte written to it.

To build and install the device driver on your virtual machine, follow the steps outlined in the Device Driver Lab. You can unit test the /dev/dice before going on to the next step. If all goes well, your /dev/dice should output the following:

```
thoth $ insmod dice_dev.ko
thoth $ cat /dev/dice | hexdump | head
0000000 0401 0202 0301 0502 0506 0501 0404 0105
0000010 0604 0306 0101 0403 0506 0401 0501 0104
0000020 0305 0402 0101 0301 0305 0604 0103 0301
…
thoth $ echo -ne \\x8\\x4 > /dev/dice
thoth $ cat /dev/dice | hexdump | head
0000000 0204 0302 0301 0102 0301 0301 0203 0202
0000010 0304 0304 0304 0203 0103 0102 0103 0203
0000020 0403 0101 0103 0204 0401 0201 0401 0301
…
```

We have used piping to view the hexdump of the device output, as we did for the Device Driver Lab. Note that the first time we read from /dev/dice it generates random rolls from a 6-sided die, whereas on the second time, it generates rolls from a 4-sided die. This is because the previous echo wrote the two bytes 8 and 4 to /dev/dice (try doing 'man echo' to understand the syntax). Since 4 is the last byte written, /dev/dice is at that point a 4-sided die.

Hints and Suggestions

- `printk()` is the version of `printf()` you can use for debugging messages from the kernel.
- In the device driver, you can only use functions in the Linux kernel.
- As root, typing `poweroff` in VirtualBox will shut it down cleanly.
- If your device driver module crashes in a bad way, the virtual machine may start to behave strangely. Seemingly unrelated things like scp may stop working. If this happens, you will need to remove the virtual machine from VirtualBox by clicking 'remove' and then start over again from 'vagrant init' and 'vagrant up'. Comparatively less painful than reinstalling your laptop OS.

RPG Implementation

Building upon the RPG fight emulator we wrote for Project 1, we are going to write a simple single-player RPG game. We are later going to expand it into a multi-player game in Project 5. At below is an example output when we first startup the game:

```
thoth $ ./rpg
What is your name?
Frodo

List of available armors:
0: cloth (AC=10)
1: studded leather (AC=12)
2: ring mail (AC=14)
3: chain mail (AC=16)
4: plate (AC=18)

Choose Frodo's Armor (0~4): 3

List of available weapons:
0: dagger (damage=1d4)
1: short sword (damage=1d6)
2: long sword (damage=1d8)
3: great sword (damage=2d6)
4: great axe (damage=1d12)

Choose Frodo's Weapon(0~4): 1

Player setting complete:
[Frodo: hp=20, armor=chain mail, weapon=short sword, level=1, xp=2000]

All is peaceful in the land of Mordor.
Sauron and his minions are blissfully going about their business:
0: [Sauron: hp=115, armor=plate, weapon=great axe, level=20, xp=1048576000]
1: [Orc 1: hp=20, armor=cloth, weapon=great sword, level=1, xp=2000]
2: [Orc 2: hp=20, armor=studded leather, weapon=long sword, level=1, xp=2000]
3: [Orc 3: hp=20, armor=cloth, weapon=short sword, level=1, xp=2000]
4: [Orc 4: hp=20, armor=cloth, weapon=great axe, level=1, xp=2000]
5: [Orc 5: hp=20, armor=ring mail, weapon=long sword, level=1, xp=2000]
6: [Orc 6: hp=20, armor=chain mail, weapon=great sword, level=1, xp=2000]
7: [Orc 7: hp=20, armor=cloth, weapon=great sword, level=1, xp=2000]
8: [Orc 8: hp=20, armor=chain mail, weapon=great sword, level=1, xp=2000]
9: [Gollum: hp=10, armor=cloth, weapon=dagger, level=1, xp=2000]
Also at the scene are some adventurers looking for trouble:
0: [Frodo: hp=20, armor=chain mail, weapon=short sword, level=1, xp=2000]
command >>
```

We set up the player name, armor and weapon as before. There are two more properties we didn't have before: level and xp. Xp is experience points. The player starts at level 1 and levels up every time that she reaches the 2^N mark where N is the level. So, xp 4000 is equivalent to level 2, xp 8000 equivalent to level 3, and so forth. The level determines how many hit points (hp) a character has and the formula is: hp = 20 + (level – 1) * 5.

After setting up the player, we populate the world with 10 non-player characters (NPCs). NPCs 1 ~ 8 (the Orcs) are all level 1 with randomly generated equipment. NPC 9 (the Gollum) always has 10 hp and is always equipped with cloth armor and a dagger, to make it and easy target for beginning players. NPC 0 (Sauron) is always level 20 and is always equipped with plate armor and great axe.

Now we are ready to enter commands after the prompt. Here are the commands you have to parse and implement: quit, stats, look, and fight.

- **Quit**: exits the program and saves the current state to a file (more on that later).

```
command >> quit
thoth $
```

- **Stats**: displays current state of the player character

```
command >> stats
[Frodo: hp=20, armor=chain mail, weapon=short sword, level=1, xp=2000]
command >>
```

- **Look**: displays current state of the world

```
command >> look
All is peaceful in the land of Mordor.
Sauron and his minions are blissfully going about their business:
0: [Sauron: hp=115, armor=plate, weapon=great axe, level=20, xp=1048576000]
1: [Orc 1: hp=20, armor=cloth, weapon=great sword, level=1, xp=2000]
2: [Orc 2: hp=20, armor=studded leather, weapon=long sword, level=1, xp=2000]
3: [Orc 3: hp=20, armor=cloth, weapon=short sword, level=1, xp=2000]
4: [Orc 4: hp=20, armor=cloth, weapon=great axe, level=1, xp=2000]
5: [Orc 5: hp=20, armor=ring mail, weapon=long sword, level=1, xp=2000]
6: [Orc 6: hp=20, armor=chain mail, weapon=great sword, level=1, xp=2000]
7: [Orc 7: hp=20, armor=cloth, weapon=great sword, level=1, xp=2000]
8: [Orc 8: hp=20, armor=chain mail, weapon=great sword, level=1, xp=2000]
9: [Gollum: hp=10, armor=cloth, weapon=dagger, level=1, xp=2000]
Also at the scene are some adventurers looking for trouble:
0: [Frodo: hp=20, armor=chain mail, weapon=short sword, level=1, xp=2000]
command >>
```

- **Fight**: fights with the chosen NPC

```
command >> fight 9
Frodo hits Gollum for 1 damage (attack roll 19)
Frodo misses Gollum (attack roll 5)
Gollum misses Frodo (attack roll 6)
Frodo hits Gollum for 6 damage (attack roll 18)
Gollum hits Frodo for 2 damage (attack roll 18)
Frodo hits Gollum for 2 damage (attack roll 11)
Gollum misses Frodo (attack roll 10)
Frodo misses Gollum (attack roll 1)
Gollum misses Frodo (attack roll 11)
Frodo hits Gollum for 3 damage (attack roll 10)
Gollum hits Frodo for 3 damage (attack roll 17)

Gollum is killed by Frodo.
Get Gollum's cloth, exchanging Frodo's current armor chain mail (y/n)? n
Get Gollum's dagger, exchanging Frodo's current weapon short sword (y/n)? y
Frodo levels up to level 2!
[Frodo: hp=25, armor=chain mail, weapon=dagger, level=2, xp=4000]
Respawning Gollum ...
[Gollum: hp=10, armor=cloth, weapon=dagger, level=1, xp=2000]
```

The fight system is identical to Project 1, except that the output is trimmed for brevity. The outcome of a fight is either: 1) the NPC dies or 2) the player dies or 3) they both die.
1) NPC dies: Player has an opportunity to change her current armor and weapon to that of the NPC. Also, player gains 2000 * (level of NPC) xp and levels up if appropriate. Also, the player's hp is fully recharged. The NPC is respawned. If NPC is an Orc, he gets a new set of randomly generated equipment, as well as a randomly generated level between 1 and the player's level. If the NPC is Gollum or Sauron, he gets the same equipment and level.
2) Player dies: Player is respawned with full hp. But player loses all xp except for the bare minimum required for that level (e.g. if xp was 6000 when killed, it would go down to 4000 on respawn).

```
command >> fight 0
…
Frodo misses Sauron (attack roll 9)
Sauron hits Frodo for 6 damage (attack roll 17)

Frodo is killed by Sauron.
Respawning Frodo ...
[Frodo: hp=25, armor=chain mail, weapon=dagger, level=2, xp=4000]
```

3) Both die: Player and NPC are both respawned.

Saving and Restoring

When the player exits from the game, the current state (all NPC and player information) is save to a file named rpg.save. Next time when the game starts up, if it finds the file rpg.save in its local directory, it will give the user a chance to start where she left off:

```
thoth $ ./rpg
Found save file.  Continue where you left off (y/n)? y

All is peaceful in the land of Mordor.
Sauron and his minions are blissfully going about their business:
0: [Sauron: hp=115, armor=plate, weapon=great axe, level=20, xp=1048576000]
1: [Orc 1: hp=20, armor=cloth, weapon=great sword, level=1, xp=2000]
2: [Orc 2: hp=20, armor=studded leather, weapon=long sword, level=1, xp=2000]
3: [Orc 3: hp=20, armor=cloth, weapon=short sword, level=1, xp=2000]
4: [Orc 4: hp=20, armor=cloth, weapon=great axe, level=1, xp=2000]
5: [Orc 5: hp=20, armor=ring mail, weapon=long sword, level=1, xp=2000]
6: [Orc 6: hp=20, armor=chain mail, weapon=great sword, level=1, xp=2000]
7: [Orc 7: hp=20, armor=cloth, weapon=great sword, level=1, xp=2000]
8: [Orc 8: hp=20, armor=chain mail, weapon=great sword, level=1, xp=2000]
9: [Gollum: hp=10, armor=cloth, weapon=dagger, level=1, xp=2000]
Also at the scene are some adventurers looking for trouble:
0: [Frodo: hp=20, armor=chain mail, weapon=short sword, level=1, xp=2000]
command >>
```

Signals

- **SIGTERM**, **SIGINT**: Write a signal handler for these signals such that the game saves the current state to the file rpg.save before terminating, even when it is terminated using Ctrl+C or kill.
- **SIGRTMIN**: Write a signal handler for this user signal such that an earthquake is caused every time this signal is raised:

```
command >>
EARTH QUAKE!!!

Sauron suffers -20 damage but survives.
Orc 1 suffers -20 damage and dies.  Respawning ...
[Orc 1: hp=25, armor=studded leather, weapon=short sword, level=2, xp=4000]
Orc 2 suffers -20 damage and dies.  Respawning ...
[Orc 2: hp=25, armor=ring mail, weapon=great sword, level=2, xp=4000]
Orc 3 suffers -20 damage and dies.  Respawning ...
[Orc 3: hp=25, armor=studded leather, weapon=short sword, level=2, xp=4000]
Orc 4 suffers -20 damage and dies.  Respawning ...
[Orc 4: hp=25, armor=chain mail, weapon=short sword, level=2, xp=4000]
Orc 5 suffers -20 damage and dies.  Respawning ...
[Orc 5: hp=25, armor=studded leather, weapon=great sword, level=2, xp=4000]
Orc 6 suffers -20 damage and dies.  Respawning ...
[Orc 6: hp=20, armor=cloth, weapon=great axe, level=1, xp=2000]
Orc 7 suffers -20 damage and dies.  Respawning ...
[Orc 7: hp=25, armor=plate, weapon=long sword, level=2, xp=4000]
Orc 8 suffers -20 damage and dies.  Respawning ...
[Orc 8: hp=20, armor=cloth, weapon=long sword, level=1, xp=2000]
Gollum suffers -20 damage and dies.  Respawning ...
[Gollum: hp=10, armor=cloth, weapon=dagger, level=1, xp=2000]
Frodo suffers -20 damage but survives.
command >>
```

Hints and Suggestions

- There is a **reference implementation** for the RPG game at: `~wahn/public/cs449/rpg`. To observe behavior described in the worksheet, please try running the above binary. It does not use /dev/dice to generate dice rolls to make it easier for you to test.

- You may write the RPG game using the C Library rand() for testing, but make sure you eventually replace it with read / write to `/dev/dice`.

- When reading / writing to /dev/dice make sure you use read() and write() system calls, instead of the C Library fread() and fwrite(). The C Library calls does buffering so the actual read() and write() system calls may not happen immediately after the fread() and fwrite(). They will only happen after draining (or filling) the buffer. This can cause problems when you want fine-grained control over a device like the case here.

- When you save the current state to rpg.save, you will be saving player and NPC structs. Make sure you don't have any pointers in those structs. Saving a pointer to a file (or sending it across a network) is always a big no-no. Because when you load the file back in, the memory layout may have changed, in which case the pointer will not point to the object that it intends.

References

If you are still unsure about what a device driver does, try reading the /dev/hello device driver tutorial written by Valerie Henson linked below the project worksheet. Also, use the LDD Chapter 3: Char Drivers link beside the Device Driver lecture slides as a reference source.

Running RPG

We cannot run our `rpg` program on `thoth.cs.pitt.edu` because its kernel does not have the device driver loaded. However, we can test the program on our virtual machine in VirtualBox once we have installed the new device driver. We need to download both `rpg` and `dice_dev.ko` using `scp` as we did in the Device Driver Lab. Install `dice_dev.ko` using `insmod` before trying `rpg`.

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.

Requirements and Submission

You need to submit:

- Your `dice_dev.c` file and the `Makefile`
- Your `rpg.c` file which implements the RPG game

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

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