EI338 Computer Systems Engineering

Project 1 Introduction to Linux Kernel Modules

November 29, 2019

Zhihui Xie, 517030910356

The environment used in this project is **Deepin 15.11**, the latest version of an open source operating system based on Debian's stable branch. The kernel version is **Linux version 4.15.0**.

Exercise 1

Question

Design a kernel module that creates a */proc* file named */proc/jiffies* that reports the current value of **jiffies** when the */proc/jiffies* file is read, such as with the command

```
cat /proc/jiffies
```

Be sure to remove */proc/jiffies* when the module is removed.

Thinking

It would not be a problem reporting the value of **jiffies** since it is directly declared in *<linux/jiffies.h>*. The difficulty may lies in the understanding of each part of kernel structure in our first attempt of kernel designing.

Answer

In this exercise, we design a kernel module named *jiffies*. There are mainly three parts or three functions to realize our purpose.

- **proc_init()**, the module entry point
- **proc_exit()**, the module exit point
- **proc_read()**, the function called when the specific */proc* file is read

proc_init()

In **proc_init()**, we create the new */proc/jiffies* entry using the **proc_create()** function. This function is passed **proc_ops** which contains a reference to a struct **file_operations**. It serves as a simple interface to create the */proc* file system. This struct initializes the **.owner** and **.read** members. The value of **.read** is the name of the function **proc_read()**. An message is sent to a kernel log buffer using **printk()** to notice that the module is successfully inserted.

```
/**
* Function prototypes
*/
static ssize_t proc_read(struct file *file, char *buf, size_t count, loff_t *pos);

static struct file_operations proc_ops = {
```

```
        .owner = THIS_MODULE,
        .read = proc_read,
};

/* This function is called when the module is loaded. */
static int proc_init(void)
{

        // creates the /proc/jiffies entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC_NAME, 0, NULL, &proc_ops);

        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);

    return 0;
}
```

Listing 1: **proc_init()**

proc_exit()

In **proc_exit()**, we need to remove the */proc/jiffies* using the function **remove_proc_entry()**. An message is sent to a kernel log buffer using **printk()** to notice that the module is successfully removed.

```
static void proc_exit(void) {

        // removes the /proc/jiffies entry
        remove_proc_entry(PROC_NAME, NULL);

        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}
```

Listing 2: **proc_exit()**

proc_read()

In **proc_read()**, we are dealing with how to report the current value of **jiffies** when the */proc/jiffies* file is read. It is not a big deal to get the value of **jiffies** since it is directly declared in the file *<linux/jiffies.h>*. The real problem is about how can we display it in the command line.

The read handler **proc_read()** receives 4 parameters:

- File object ***file**, per process structure with the opened file details (permission , position, etc.)
- User space buffer ***usr_buf**
- Buffer size **count**
- Requested position ***pos**

To implement the read callback, we need to:

- Check the requested position
- Fill the user buffer with a data (max size $\leq$ buffer size) from the requested position
- Return the number of bytes we filled

We first check if it is the first time we read the file and the user buffer size is bigger than **BUFFER_SIZE**. If not, we return 0 to indicate that there is nothing to read. Finally, we build the returned buffer, copy it to the user, update the position and return the number of bytes we wrote. And it is worth mentioning why we use the function **copy_to_user()**. We use it to memcpy the data from the user space to the kernel space since it is not accessible for us to directly deal with the buffer in kernel space.

```
static ssize_t proc_read
(struct file *file, char __user *usr_buf, size_t count, loff_t *pos)
{
        char buffer[BUFFER_SIZE];
        int len = 0;

        if(*pos > 0 || count < BUFFER_SIZE) return 0;

        len += sprintf(buffer, "The total number of interrupts is %lu\n", jiffies);

        // copies the contents of buffer to userspace usr_buf
        copy_to_user(usr_buf, buffer, len);

        // updates the position and returns the number of bytes we received
        *pos = len;
        return len;
}
```

Listing 3: **proc_read()**

Each time the */proc/jiffies* file is read, the **proc_read()** function is called repeatedly until it returns 0.

Experiment

We directly present the commands in shell.

Figure 1: Experiment Result in Exercise 1

Exercise 2

Question

Design a kernel module that creates a *proc* file named */proc/seconds* that reports the number of elapsed seconds since the kernel module was loaded. This will involve using the value of **jiffies** as well as the **HZ** rate. When a user enters the command

```
cat /proc/seconds
```

your kernel module will report the number of seconds that have elapsed since the kernel module was first loaded. Be sure to remove **/proc/seconds** when the module is removed.

Thinking

This time, we need to keep some value. We may use a global variable to address this. Nothing new.

Answer

All three main functions are realized in a way similar to that in Exercise 1. The main difference lies on the output: the output changes depending on when we call the function. Besides this, everything is the same.

To address this, we declare a global variable named **interrupt_start** to record the value of **jiffies** when the module is loaded.

And in **proc_init()**, we assign the current value of **jiffies** to it.

```
static int proc_init(void)
{

        // ...

        interrupt_start = jiffies;
        return 0;
}
```

Listing 4: **proc_init()**

The function **proc_exit()** keeps the same. In **proc_read()**, we need to simply calculate the elapsed seconds using **HZ** and the difference between current **jiffies** and **interrupt_start**.

```
static ssize_t proc_read
(struct file *file, char __user *usr_buf, size_t count, loff_t *pos)
{
        // ...

        unsigned long int interrupts = jiffies - interrupt_start;

        // ...

        len += sprintf(buffer, "The number of elapsed seconds
        since the kernel module was loaded is %lus\n", interrupts/HZ);

        // ...

        return len;
}
```

Listing 5: **proc_read()**

Experiment

We directly present the commands in shell. To show that our output is correct, we check the system time as reference.

Figure 2: Experiment Result in Exercise 2