EI 338: Project #1

Due on Sunday, Octuber 20, 2019

Wu Chentao 12:55pm

Ma Jiaqi

Contents

Environment

Ubuntu 18.04.2

Description

The Introduction of the Linux's kernel. And help give some guidance on how to add and remove modules in the kernel. Besides, it teaches us some basic function in the module, such as the initial part and the exit part, the basic working process of the module.

Module

Introduction

Followed is a simple module, only within the init part and the exit part. Whenever this module is inserted into the kernel, the program will run the init part, so does the exit part. So, we can track the module's action by printing a message on the system's log.

Listing 1: A sample Module

```c
/**
 * simple.c
 *
 * A simple kernel module.
 *
 * To compile, run makefile by entering "make"
 *
 * Operating System Concepts - 10th Edition
 * Copyright John Wiley & Sons - 2018
 */

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

/* This function is called when the module is loaded. */
static int simple_init(void)
{
       printk(KERN_INFO "Loading Module\n");

       return 0;
}

/* This function is called when the module is removed. */
static void simple_exit(void) {
     printk(KERN_INFO "Removing Module\n");
}

/* Macros for registering module entry and exit points. */
module_init( simple_init );
module_exit( simple_exit );

MODULE_LICENSE("GPL");
```

```
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("SGG");
```

Insert & Remove

Before the Insert and Remove operation, we need to compile the code first.

Listing 2: Makefile

```
obj-m += #FileName.o
all:
     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Then we can get a series file about our module, and we only need to concern on the *.ko* file.

Listing 3: Corresponding Instruction

```
sudo insmod sample.ko

sudo rmmod sample
```

The above instruction is to insert a module into the kernel and the bottom one is to remove a module from the kernel. Notice that the insert instruction has *.ko* postfix and the remove one doesn't. Because to insert a module, system need to find the file firstly to compile it, but when removing it, the module has existed in the module list, so only the module's name is enough.

Cat

cat instruction can help us call the module.

Listing 4: Jiffies.c

```
/**
 * jiffies.c
 *
 * Kernel module that communicates with /proc file system.
 *
 * The makefile must be modified to compile this program.
 * Change the line "simple.o" to "hello.o"
 *
 * Operating System Concepts - 10th Edition
 * Copyright John Wiley & Sons - 2018
 */

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/jiffies.h>
#include <asm/uaccess.h>
```

Module [Cat] continued on next page...

```c
#define BUFFER_SIZE 128

#define PROC_NAME "jiffies"
#define MESSAGE "Hello World\n"

/**
 * 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/hello 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;
}

/* This function is called when the module is removed. */
static void proc_exit(void) {

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

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

/**
 * This function is called each time the /proc/hello is read.
 *
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 *
 * params:
 *
 * file:
 * buf: buffer in user space
 * count:
 * pos:
```

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

        if (completed) {
                completed = 0;
                return 0;
        }

        completed = 1;

        rv = sprintf(buffer, "%lu\n", jiffies);

        // copies the contents of buffer to userspace usr_buf
        raw_copy_to_user(usr_buf, buffer, rv);

        return rv;
}


/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Jiffies Module");
MODULE_AUTHOR("SGG");
```

Above is the *jiffies.c*'s code. Let's focus on the *proc_read* function. When this module is called by user, it will active this this function. Here I let the code print the value of the variable jiffies to verify that this module is called successfully.

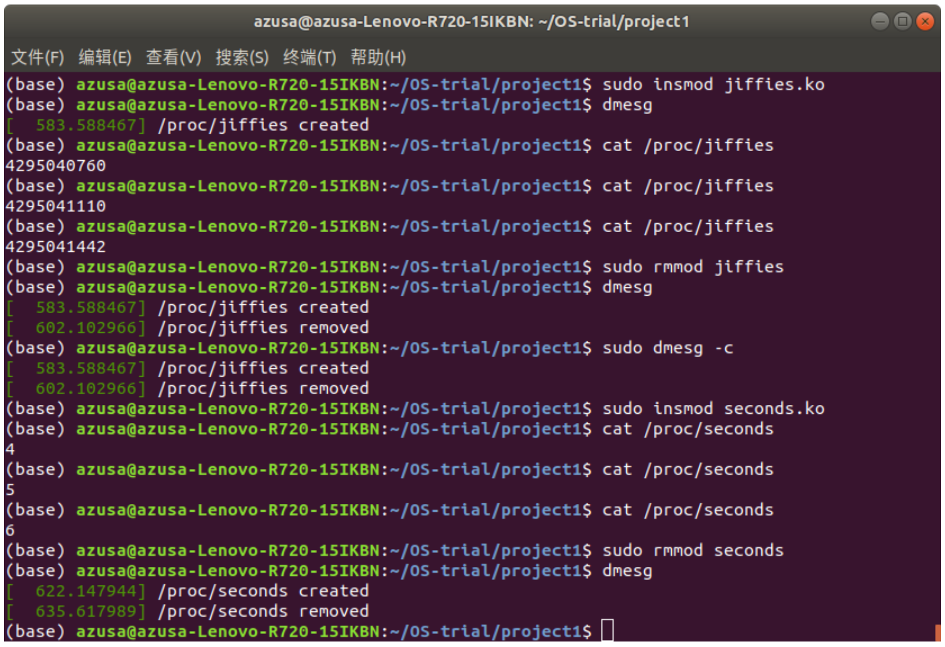
Exercise

1. 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.

2. 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.

The *jiffies.c* has been presented before. Followed is the *seconds.c* and the screenshot of the actual situation.

Exercise continued on next page…


azusa@azusa-Lenovo-R720-15IKBN: ~/OS-trial/project1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ sudo insmod jiffies.ko
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ dmesg
[  583.588467] /proc/jiffies created
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ cat /proc/jiffies
4295040760
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ cat /proc/jiffies
4295041110
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ cat /proc/jiffies
4295041442
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ sudo rmmod jiffies
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ dmesg
[  583.588467] /proc/jiffies created
[  602.102966] /proc/jiffies removed
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ sudo dmesg -c
[  583.588467] /proc/jiffies created
[  602.102966] /proc/jiffies removed
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ sudo insmod seconds.ko
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ cat /proc/seconds
4
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ cat /proc/seconds
5
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ cat /proc/seconds
6
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ sudo rmmod seconds
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ dmesg
[  622.147944] /proc/seconds created
[  635.617989] /proc/seconds removed
(base) azusa@azusa-Lenovo-R720-15IKBN:~/OS-trial/project1$ 
```

Listing 5: Seconds.c

```c
/**
 * seconds.c
 *
 * Kernel module that communicates with /proc file system.
 *
 * The makefile must be modified to compile this program.
 * Change the line "simple.o" to "hello.o"
 *
 * Operating System Concepts - 10th Edition
 * Copyright John Wiley & Sons - 2018
 */

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/jiffies.h>
#include <asm/param.h>
#include <asm/uaccess.h>

#define BUFFER_SIZE 128

#define PROC_NAME "seconds"
#define MESSAGE "Hello World\n"

/**
 * Function prototypes
 */
long jif=0;
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/hello 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);
     jif=jiffies;
        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);

     return 0;
}

/* This function is called when the module is removed. */
static void proc_exit(void) {

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

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

/**
 * This function is called each time the /proc/hello is read.
 *
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 *
 * params:
 *
 * file:
 * buf: buffer in user space
 * count:
 * pos:
 */
static ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *pos)
{
        int rv = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;

        if (completed) {
                completed = 0;
                return 0;
```

Exercise continued on next page. . .

```
        }

        completed = 1;

        rv = sprintf(buffer, "%lu\n", (jiffies-jif)/HZ);

        // copies the contents of buffer to userspace usr_buf
        raw_copy_to_user(usr_buf, buffer, rv);

        return rv;
}


/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("seconds Module");
MODULE_AUTHOR("SGG");
```