Project 1: Introduction to Linux Kernel Modules

连炎杰　519021910706

1 Introduction

Project1作为操作系统课程设计的第一个project，向我们初步介绍了Linux内核模块的运作，并将教会我们如何自己创建一个内核模块并将其加载进 Linux kernel，以及动手实现一些简单的内核模块。

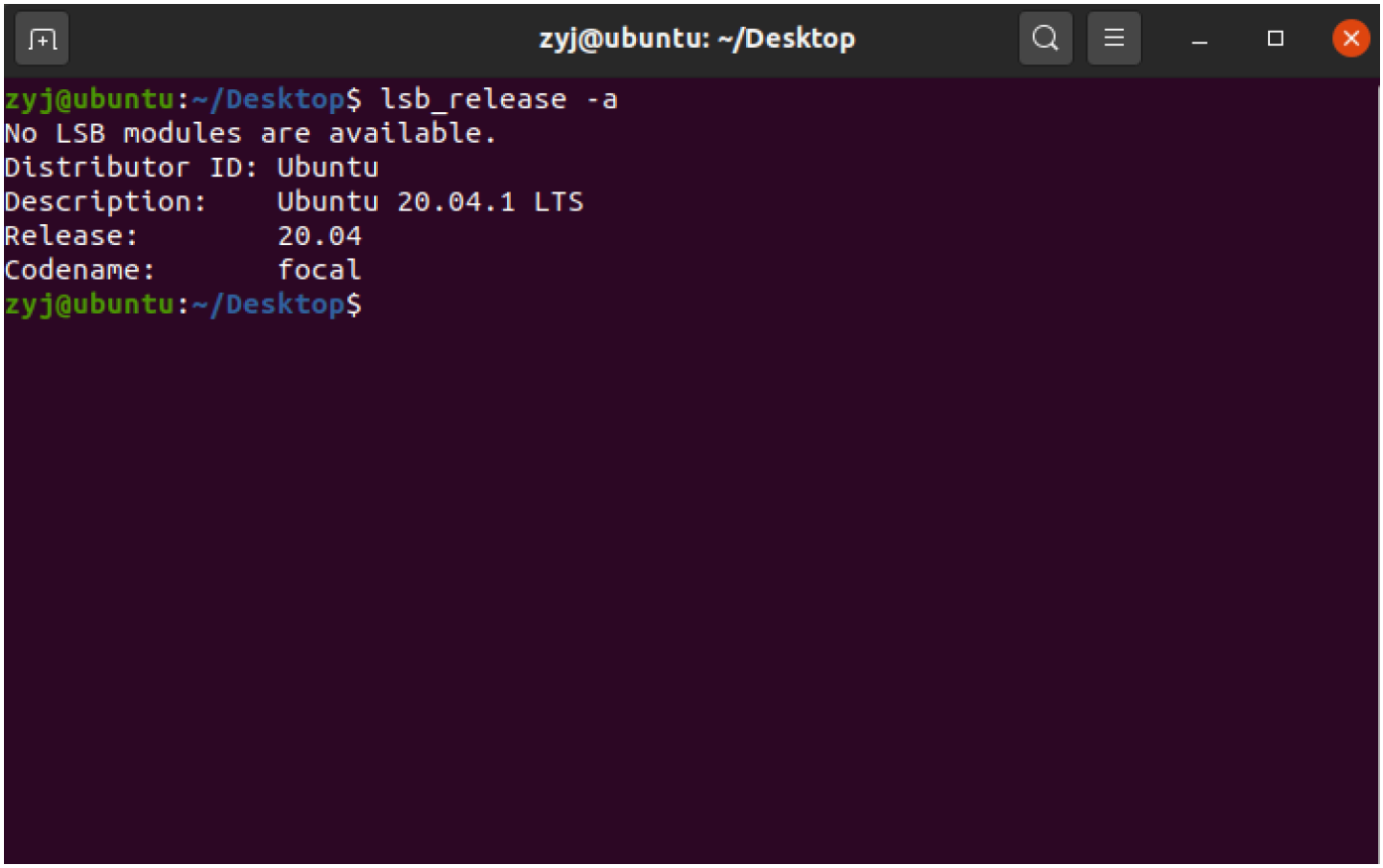
1.1 运行环境

我完成project1的环境为：在**MacOS**上的**VMware Fusion**虚拟环境中的**Ubuntu20.04**。

查看macOS版本：点击左上角苹果标志即可查看。

查看ubuntu版本号：在ubuntu中打开命令行，输入：

```
lsb_release -a
```



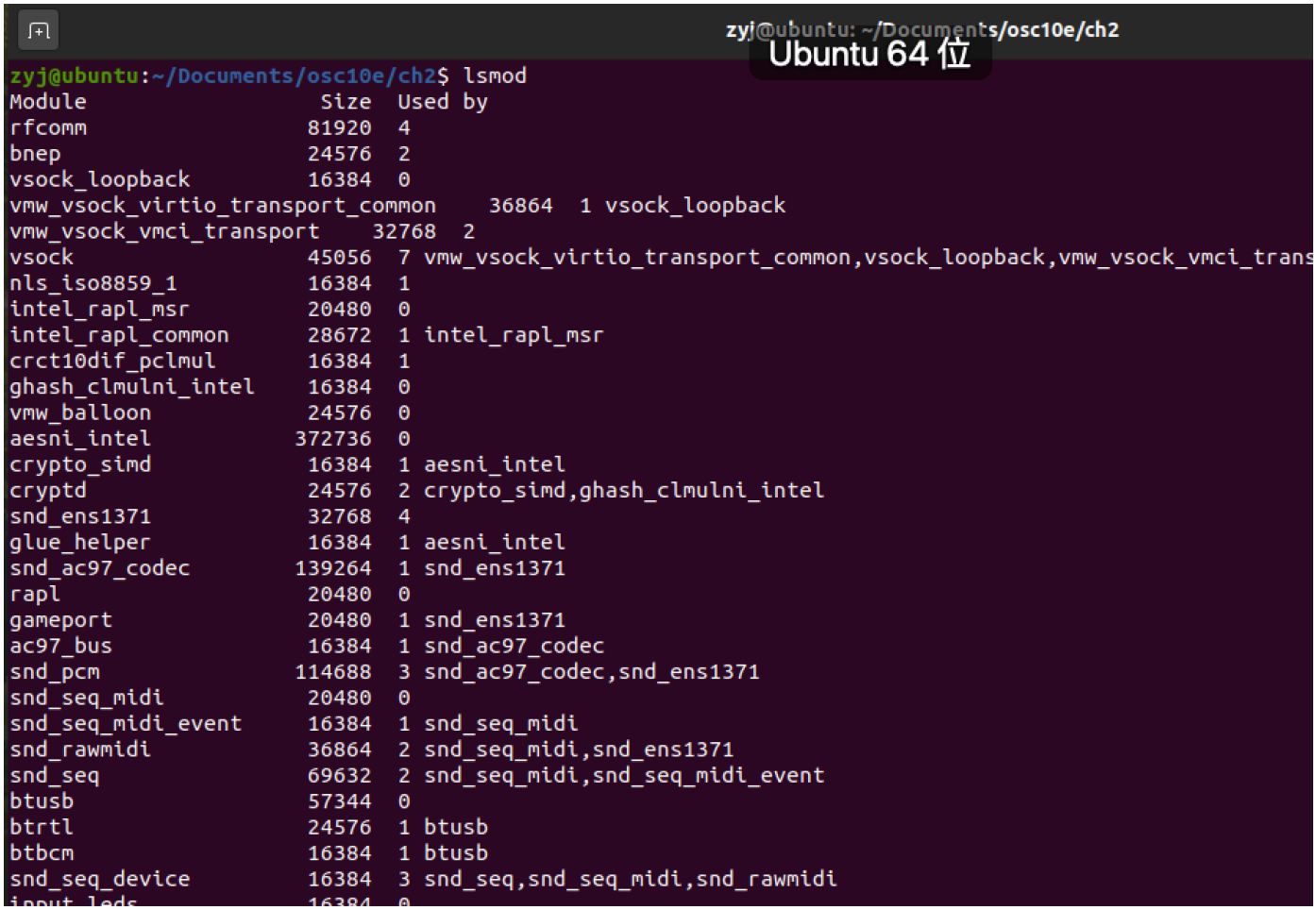
2 Linux Kernel初步探索

通过该部分的引导教程和实验我们可以初步掌握Linux 内核模块的相关操作。

2.1 列出已加载的内核模块

```
lsmod
```

通过在命令行输入该命令，可以获得当前已加载入内核的模块，以 **名字，大小，使用处** 的形式展现在命令行中，如下图所示。



2.2 内核模块的程序模版

以下程序向我们展示了一个简单的内核程序的组织形式。

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
/* This function is called when the module is loaded. */
int simple_init(void)
{
printk(KERN_INFO "Loading Kernel Module\n");
return 0;
}
/* This function is called when the module is removed. */
void simple_exit(void)
{
printk(KERN_INFO "Removing Kernel 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("Yanjie Ze");
```

函数simple_init()作为 **module entry point**，在模块加载进内核的时候被调用。这类函数必须返回一个整型值，0代表调用成功，其他值代表调用失败。

函数simple_exit()作为 **module exit point**，在模块被移出内核的时候被调用。

这两类函数都不需要传递任何参数。

为了将模块出入口作为模块的一部分加载入内核，需要使用

```
module_init(simple_init)
module_exit(simple_exit)
```

需要注意的是在

```
printk(KERN_INFO "Removing Kernel Module\n");
```

中调用的printk函数。该函数相当于内核模块中的printf。printk函数允许我们声明一个priority flag，即代码中的 **KERN_INFO**。

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("Yanjie Ze");
```

这三行表示更多的模块信息，比如license，description以及author。

2.3 加载和移除内核模块

对于程序simple.c，我们需要首先进行编译，将其转化为.ko文件。

2.3.1 编译

在命令行中执行：

```
make
```

将运行该文件夹下的makefile文件：

```
obj-m += simple.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
```

看到以下输出我们可以知道编译成功了：

```
zyj@ubuntu:~/Documents/osc10e/ch2$ make
make -C /lib/modules/5.8.0-50-generic/build M=/home/zyj/Documents/osc10e/ch2 modules
make[1]: 进入目录“/usr/src/linux-headers-5.8.0-50-generic”
  CC [M]  /home/zyj/Documents/osc10e/ch2/simple.o
  MODPOST /home/zyj/Documents/osc10e/ch2/Module.symvers
  CC [M]  /home/zyj/Documents/osc10e/ch2/simple.mod.o
  LD [M]  /home/zyj/Documents/osc10e/ch2/simple.ko
make[1]: 离开目录“/usr/src/linux-headers-5.8.0-50-generic”
zyj@ubuntu:~/Documents/osc10e/ch2$
```

2.3.2 加载内核模块

执行：

```
sudo insmod simple.ko
```

我们将simple模块加载入内核。

为了验证是否加载成功，执行：

```
dmesg
```

加载成功后，应该输出如下结果：

```
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo insmod simple.ko
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[29422.602582] Loading Kernel Module\n
```

2.3.3 移除内核模块

执行：

```
sudo rmmod simple
```

将刚才载入的simple模块移除。

为了验证是否移除成功，执行：

```
dmesg
```

移除成功后，有如下结果：

```
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo rmmod simple.ko
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[29650.122849] Removing Kernel Module\n
[29713.180714] Loading Kernel Module\n
```

由于kernel log buffer会很快被填满信息，执行：

```
sudo dmesg -c
```

来清除buffer。其中-c表示clear。

2.4 小测试：输出GOLDEN_RATIO_PRIME和最大公约数

project中要求我们对simple模块做一点修改：

1. 在simple_init中输出GOLDEN_RATIO_PRIME。
2. 在simple_exit中输出3300和24的最大公约数。

修改后完整代码如下：

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gcd.h>
#include <linux/hash.h>
#include <asm/param.h>
#include <linux/jiffies.h>

/* This function is called when the module is loaded. */
static int simple_init(void)
{
    printk(KERN_INFO "%llu\n", GOLDEN_RATIO_PRIME);
       return 0;
}

/* This function is called when the module is removed. */
static void simple_exit(void) {
    unsigned long result = gcd(3300, 24);
  printk(KERN_INFO "%lu\n", result);
}

/* 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("Yanjie Ze");
```

结果展示:

```
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[29720.076615] Removing Kernel Module\n
[30194.000872] 7046029254386353131\n
[30225.655203] 12
```

2.5 小测试：输出jiffies和HZ

根据project中的任务，我们需要再对simple模块做一点修改：

1. 在simple_init中输出jiffies和HZ的值。
2. 在simple_exit中输出jiffies的值。

其中，jiffies是timer interrupt自从系统启动以来的数量，HZ表示timer interrupt的频率。

修改后的simple.c的完整程序如下：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gcd.h>
#include <linux/hash.h>
#include <asm/param.h>
#include <linux/jiffies.h>

/* This function is called when the module is loaded. */
static int simple_init(void)
{
       printk(KERN_INFO "%u\n", HZ);// 250
       printk(KERN_INFO "%lu\n", jiffies);// 4303130727
       return 0;
}

/* This function is called when the module is removed. */
static void simple_exit(void) {
  printk(KERN_INFO "%lu\n", jiffies);// 4303216083
}

/* 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("Yanjie Ze");
```

结果展示：

```
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo insmod simple.ko
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[30573.270005] 250
[30573.270006] 4302535739
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo rmmod simple
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[30573.270005] 250
[30573.270006] 4302535739
[30776.039365] 4302586428
zyj@ubuntu:~/Documents/osc10e/ch2$
```

3 /proc 文件系统

/proc文件系统是一个仅存在在内核内存中的“伪”文件系统，只用来查询内核和进程的统计信息。

3.1 /proc文件系统的hello world

由于版本问题，project给出的hello.c中存在一些旧版本的函数已经被更新，因此我们使用更新后的函数。

更新后的hello.c完整程序如下：

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

#define BUFFER_SIZE 128

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

/**
 * Function prototypes
```

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

// static struct proc_ops proc_ops = {
//         .owner = THIS_MODULE,
//         .read = proc_read,
// };
static struct proc_ops my_fops={
    .proc_read = seq_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, &my_fops);

        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 seq_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, "Hello World\n");//rv is the number of
string

        // copies the contents of buffer to userspace usr_buf
        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("Hello Module");
MODULE_AUTHOR("Yanjie Ze");
```

上述程序中，函数seq_read：

```
static ssize_t seq_read(struct file *file, char __user *usr_buf, size_t
count, loff_t *pos)
```

在每次我们调用**cat /proc/hello**时都会执行。在该程序中，我们让seq_read函数执行向buffer中输出hello world的功能。

载入内核后，我们可以执行：

```
cat /proc/hello
```

来判断/proc/hello模块是否加载成功。若是加载成功，每次运行上述命令都会输出“Hello World”。

完整的结果展示如下：

```
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo insmod hello.ko
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[30573.270005] 250
[30573.270006] 4302535739
[30776.039365] 4302586428
[31708.161547] /proc/hello created
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/hello
Hello World
```

4 Assignment

在初步学习Linux内核模块的设计方法后，我们需要完成两个简单内核模块的设计。

4.1 任务1

4.1.1 任务描述

我们需要设计一个内核模块实现以下功能：

1. 创建一个文件名为/proc/jiffies
2. 每当调用 **cat /proc/jiffies** 时，汇报当前jiffies的值。

4.1.2 实现思路

我们需要利用函数sprintf将长无符号整数jiffies的值以字符串的形式保存进message：

```
sprintf(message, "%lu\n", jiffies);
```

再将message存入一个buffer，buffer用来向user-buffer写入数据。

```
rv = sprintf(buffer, message);//rv is the number of string
```

最后写入usr-buffer即可：

```
copy_to_user(usr_buf, buffer, rv);
```

4.1.3 完整实现程序

借助前面对于jiffies的应用，以及对于proc文件系统操作的认知，我们可以完成这个比较简单的模块。

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/jiffies.h>
#include <asm/param.h>
#define BUFFER_SIZE 128

#define PROC_NAME "jiffies"

static ssize_t seq_read(struct file *file, char *buf, size_t count,
loff_t *pos);

static struct proc_ops my_fops={
    .proc_read = seq_read,
};


static int proc_init(void)
{

        proc_create(PROC_NAME, 0, NULL, &my_fops);
```

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

  return 0;
}

static void proc_exit(void) {

        remove_proc_entry(PROC_NAME, NULL);

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

static ssize_t seq_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;
        char message[25];

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

        completed = 1;

        sprintf(message, "%lu\n", jiffies);
        rv = sprintf(buffer, message);//rv is the number of string

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

        return rv;
}


module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Jiffies Module");
MODULE_AUTHOR("Yanjie Ze");
```

4.1.4 程序结果展示

我们通过dmesg展示jif模块的成功加载与删除，通过执行 **cat /proc/jiffies** 获得jiffies值。

```
zyj@ubuntu:~/Documents/osc10e/ch2$ make
make -C /lib/modules/5.8.0-50-generic/build M=/home/zyj/Documents/osc10e/ch2 modules
make[1]: 进入目录“/usr/src/linux-headers-5.8.0-50-generic”
make[1]: 离开目录“/usr/src/linux-headers-5.8.0-50-generic”
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo insmod jif.ko
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[32447.037835] /proc/jiffies created
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/jiffies
4303008442
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/jiffies
4303009192
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/jiffies
4303009914
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo rmmod jif
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo dmesg -c
[32447.037835] /proc/jiffies created
[32477.014523] /proc/jiffies removed
```

4.2 任务2

4.2.1 任务描述

我们需要设计一个内核模块实现以下功能：

1. 创建一个proc文件名为/proc/seconds
2. 通过使用jiffies和HZ，计算自从模块加载后过去的秒数。当执行 **cat /proc/seconds** 时获得seconds的值。

4.2.2 实现思路

我们通过以下公式来计算自模块被加载到现在已经过去的时间：

$$second\_pass = (current\_jif - init\_jif)/HZ;$$

在代码中用一行实现：

```
second_pass = (current_jif - init_jif)/HZ;
```

然后与实现jiffies的方法相同，首先将整数传入字符串：

```
sprintf(message, "second pass: %lu\n", second_pass);
```

再用buffer保存：

```
        rv = sprintf(buffer, message);//rv is the number of string
```

最后传入usr-buffer：

```
copy_to_user(usr_buf, buffer, rv);
```

4.2.3 完整实现程序

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/jiffies.h>
#include <asm/param.h>
#define BUFFER_SIZE 128

#define PROC_NAME "seconds"

static ssize_t seq_read(struct file *file, char *buf, size_t count,
loff_t *pos);

static struct proc_ops my_fops={
    .proc_read = seq_read,
};

unsigned long init_jif;

static int proc_init(void)
{
        init_jif = jiffies;
        proc_create(PROC_NAME, 0, NULL, &my_fops);

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

  return 0;
```

```
}

static void proc_exit(void) {


        remove_proc_entry(PROC_NAME, NULL);

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

static ssize_t seq_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;
        char message[25];
        unsigned long current_jif = jiffies;
        unsigned long second_pass;

        second_pass = (current_jif - init_jif)/HZ;

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

        completed = 1;

        sprintf(message, "second pass: %lu\n", second_pass);
        rv = sprintf(buffer, message);//rv is the number of string

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

        return rv;
}


module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Jiffies Module");
MODULE_AUTHOR("Yanjie Ze");
```

4.2.4 程序结果展示

通过不断输入 **cat /proc/seconds**，我们可以发现秒数不断增加，表示着时间的流逝。

```
zyj@ubuntu:~/Documents/osc10e/ch2$ make
make -C /lib/modules/5.8.0-50-generic/build M=/home/zyj/Documents/osc10e/ch2 modules
make[1]: 进入目录“/usr/src/linux-headers-5.8.0-50-generic”
make[1]: 离开目录“/usr/src/linux-headers-5.8.0-50-generic”
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo insmod second.ko
zyj@ubuntu:~/Documents/osc10e/ch2$ dmesg
[32742.373514] /proc/seconds created
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/second
cat: /proc/second: 没有那个文件或目录
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/seconds
second pass: 14
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/seconds
second pass: 23
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/seconds
second pass: 25
zyj@ubuntu:~/Documents/osc10e/ch2$ cat /proc/seconds
second pass: 26
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo rmmod second
zyj@ubuntu:~/Documents/osc10e/ch2$ sudo dmesg -c
[32742.373514] /proc/seconds created
[32779.286467] /proc/seconds removed
```

5 总结与感想

Project 1带领我们从一些简单的小程序入手，了解到关于Linux 内核模块的基本操作，让我们学会了如何加载一个模块进入内核和如何从内核移除一个模块，以及让我们了解了Linux的proc文件系统。

虽然我对于内核模块还了解不深，但是这个project加深了我对Linux内核的兴趣，并且让我在没有很多专业知识的基础上能动手实践和体验，说明该project设计是十分巧妙的。