

Project1: Introduction To Linux Kernel Module

518021911193 刘昊林

1 使用到的功能、函数和命令

1.1 内核模块编程的基本结构

在为 Linux 内核模块编写.c 文件时，首先引入一系列的 Linux 头文件，例如 `#include <linux/init.h>` 等。然后设置模块的入口和出口，体现在.c 文件中为 `module_init(simple_init)` 和 `module_exit(simple_exit)` 两句。具体的入口和出口功能则需要函数 `int simple_init(void)` 和 `void simple_exit(void)` 来实现。这里需要注意，入口函数 `simple_init(void)` 必须返回一个整数值，其中 0 表示成功，其他任何值表示失败；但是出口函数 `simple_exit(void)` 则不需要返回任何值，并且这两个函数均不需要传递任何参数。在.c 文件的最后还需要加上 `MODULE_LICENSE()`、`MODULE_DESCRIPTION()`、`MODULE_AUTHOR()` 来注明模块的许可信息，说明信息和作者信息。

一个简单的内核模块.c 文件的结构如下所示：

```
1      #include <linux/init.h>
2      #include <linux/module.h>
3      #include <linux/kernel.h>
4
5      int simple_init(void) {
6          printk(KERN_INFO "Loading Module\n");
7          return 0;
8      }
9
10     void simple_exit(void) {
11         printk(KERN_INFO "Removing Module\n");
12     }
13
14     module_init( simple_init );
15     module_exit( simple_exit );
16
17     MODULE_LICENSE( "GPL" );
18     MODULE_DESCRIPTION( "Simple Module" );
19     MODULE_AUTHOR( "LHL" );
```

1.2 编写/proc 文件系统的基本结构

在上述结构的基础上，首先需要在.c 文件中设置一个全局变量 PROC_NAME 和一个结构体 static struct file_operations proc_ops，并设置好其中的.owner 和.read 等成员的值，特别.read 的值是每次读取对应文件时调用的函数 proc_read。然后为了创建/proc 文件系统的入口和出口，需要分别在 simple_init 模块入口函数中加入 proc_create(PROC_NAME, 0, NULL, &proc_ops) 文件系统创建函数；在 simple_exit 模块出口函数中加入 remove_proc_entry(PROC_NAME, NULL) 文件系统移除函数。最后，需要自己编写一个 proc_read 函数，每当接收到读取文件的指令时均会调用这个函数，直至返回值为 0。需要注意的是，这个函数必须有返回值，值为 0 表示读取结束，否则将会继续调用这个函数，所以引入了一个变量 completed 来控制函数的结束。并且这个函数还需要将内核空间中的内容复制到用户空间中，以便进行内容的读取与输出。所以将函数原型设置为 ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *pos)。

这里仅给出/proc 文件系统需要在上面的基本结构中添加的内容：

```
1      #define PROC_NAME "name"
2
3      ssize_t proc_read(struct file *file , char *buf
4      , size_t count , loff_t *pos);
5
6      static struct file_operations proc_ops = {
7          .owner = THIS_MODULE,
8          .read = proc_read ,
9      };
10
11      int proc_init(void) {
12          proc_create(PROC_NAME, 0, NULL, &proc_ops);
13          printk(KERN_INFO "/proc/%s created\n", PROC_NAME);
14          return 0;
15      }
16
17      void proc_exit(void) {
18          remove_proc_entry(PROC_NAME, NULL);
19          printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
20      }
21
22      ssize_t proc_read(struct file *file , char __user *usr_buf
23      , size_t count , loff_t *pos) {
24          static int completed = 0;
25
```

```

26         if (completed) {
27             completed = 0;
28             return 0;
29         }
30
31         completed = 1;
32         Do something here .....
33     }

```

1.3 使用到的函数

1. printk() 函数，用来在内核中打印信息，与 printf 类似，但是 printk 运行在内核态，printf 运行在用户态。
2. proc_create() 函数，用来创建一个 proc 文件系统。
3. remove_proc_entry() 函数，用来移除一个 proc 文件系统。
4. sprintf() 函数，原型为 int sprintf(char *str, const char *format, ...)，发送字符串 format 到 str 所指向的字符串。如果输出成功，则返回写入 str 的字符总数。如果失败，则返回一个负数。
5. copy_to_user() 函数，将内核缓冲区中的信息复制到用户空间中。

1.4 使用到的命令

1. sudo insmod simple.ko，加载 simple 内核模块。
2. sudo rmmod simple，删除 simple 内核模块
3. dmesg，在内核日志缓冲区中检查输出的消息。
4. dmesg -c，清空缓冲区。
5. cat /proc/hello，连接/proc/hello 文件并打印到输出设备上。

2 实现思路

2.1 Part 1-准备阶段

添加一系列的 Linux 头文件，在 simple_init(), simple_exit() 这两个函数中利用 printk() 等函数，输出书中要求的内容。最终通过 module_init(simple_init), module_exit(simple_exit) 这两个模块的进出口，实现书中提到的以下 4 个功能：

1. Print out the value of GOLDEN_RATIO_PRIME in the simple_init() function.
2. Print out the greatest common divisor of 3,300 and 24 in the simple_exit() function.
3. Print out the values of jiffies and HZ in the simple_init() function.
4. Print out the value of jiffies in the simple_exit() function.

具体代码在 simple.c 文件中。

2.2 Part 2-assignment 1

同样添加一系列重要的 Linux 头文件，在 `simple_init()` 函数利用 `proc_create()` 创建文件，并输出已经加载并且进入该模块的提示信息，在 `simple_exit()` 函数中利用 `remove_proc_entry()` 删除文件，并输出已经卸载并且离开该模块的提示信息。此外，需要定义一个结构体，其中的 `owner` 元素赋值为模块名，`read` 元素赋值为函数 `proc_read()` 的名称，在读取 `/proc/jiffies` 时将调用该函数。在 `proc_read()` 函数中，获得 `jiffies` 的值，并通过内核函数 `copy_to_user()`，将缓冲区的内容复制到用户空间，最后利用 `cat` 命令将文件中的内容输出到 shell 中；同时设置一个 `completed` 变量，控制函数完成任务后返回 0，而不是陷入死循环。

具体代码在 `jiffies.c` 文件中。

2.3 Part 3-assignment 2

在这个部分中，大部分内容与上一个部分相同。区别在于在 `simple_init()` 中记录下开始的 `jiffies` 值，在 `proc_read()` 中记录下结束时的 `jiffies` 值，并且通过公式

$$T = \frac{jiffies_end - jiffies_start}{HZ}$$

计算出经过的时间间隔，并且输出时间间隔。

具体代码在 `seconds.c` 文件中。

3 步骤截图

3.1 编译阶段

```
psyduckliu@ubuntu:~/Desktop/Project1$ make
make -C /lib/modules/4.19.108/build M=/home/psyduckliu/Desktop/Project1 modules
make[1]: Entering directory '/usr/src/linux-4.19.108'
CC [M] /home/psyduckliu/Desktop/Project1/simple.o
CC [M] /home/psyduckliu/Desktop/Project1/jiffies.o
CC [M] /home/psyduckliu/Desktop/Project1/seconds.o
Building modules, stage 2.
MODPOST 3 modules
CC /home/psyduckliu/Desktop/Project1/jiffies.mod.o
LD [M] /home/psyduckliu/Desktop/Project1/jiffies.ko
CC /home/psyduckliu/Desktop/Project1/seconds.mod.o
LD [M] /home/psyduckliu/Desktop/Project1/seconds.ko
CC /home/psyduckliu/Desktop/Project1/simple.mod.o
LD [M] /home/psyduckliu/Desktop/Project1/simple.ko
make[1]: Leaving directory '/usr/src/linux-4.19.108'
```

图 1: 编译

3.2 Part 1-准备阶段

```
psyduckliu@ubuntu:~/Desktop/Project1$ sudo insmod simple.ko
psyduckliu@ubuntu:~/Desktop/Project1$ dmesg
[ 941.643243] Loading Kernel Module
[ 941.643245] The value of GOLDEN_RATIO_PRIME is:7046029254386353131
[ 941.643245] The value of start jiffies is: 160025
[ 941.643245] The value of HZ is: 250
psyduckliu@ubuntu:~/Desktop/Project1$ sudo rmmod simple.ko
psyduckliu@ubuntu:~/Desktop/Project1$ dmesg
[ 941.643243] Loading Kernel Module
[ 941.643245] The value of GOLDEN_RATIO_PRIME is:7046029254386353131
[ 941.643245] The value of start jiffies is: 160025
[ 941.643245] The value of HZ is: 250
[ 1001.714648] The greatest common divisor of 3,300 and 24: 12
[ 1001.714648] The value of end jiffies is: 175042
[ 1001.714649] Removing Kernel Module
```

图 2: simple.ko

3.3 Part 2-assignment 1

```
psyduckliu@ubuntu:~/Desktop/Project1$ sudo insmod jiffies.ko
psyduckliu@ubuntu:~/Desktop/Project1$ dmesg
[ 1061.694036] Loading Kernel Module
[ 1061.694039] /proc/jiffies created
psyduckliu@ubuntu:~/Desktop/Project1$ cat /proc/jiffies
The current value of jiffies is:4295162624
psyduckliu@ubuntu:~/Desktop/Project1$ sudo rmmod jiffies.ko
psyduckliu@ubuntu:~/Desktop/Project1$ dmesg
[ 1061.694036] Loading Kernel Module
[ 1061.694039] /proc/jiffies created
[ 1096.327361] /proc/jiffies removed
[ 1096.327362] Removing Kernel Module
```

图 3: jiffies.ko

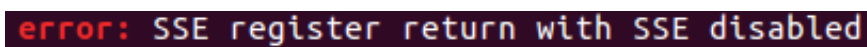
3.4 Part 3-assignment 2

```
psyduckliu@ubuntu:~/Desktop/Project1$ sudo insmod seconds.ko
psyduckliu@ubuntu:~/Desktop/Project1$ dmesg
[ 1259.203870] Loading Kernel Module
[ 1259.203879] /proc/seconds created
psyduckliu@ubuntu:~/Desktop/Project1$ cat /proc/seconds
The number of eplased seconds is 11.
psyduckliu@ubuntu:~/Desktop/Project1$ sudo rmmod seconds.ko
psyduckliu@ubuntu:~/Desktop/Project1$ dmesg
[ 1259.203870] Loading Kernel Module
[ 1259.203879] /proc/seconds created
[ 1287.375378] /proc/seconds removed
[ 1287.375379] Removing Kernel Module
```

图 4: seconds.ko

4 实验中遇到的问题

在最初编写 seconds 这个模块的时候，我将经过的时间 time 设置为 double 类型的数据，但是编译之后报错 (如下图所示)：



```
error: SSE register return with SSE disabled
```

图 5: 报错

经过我在 CSDN 上的搜索发现，内核进程与用户进程不同，内核并不能完美地支持浮点操作。在内核中使用浮点数时，除了要人工保存和恢复浮点寄存器，还有其他一些琐碎的事情要做，所以在 Linux 内核中最好不要使用浮点数表示数据。所以我被迫把 time 的数据类型改为了 int 型，问题虽然是迎刃而解，但是数据的精度却下降了。