

## 0.1 Project 1 Linux 内核模块简介

下面的内容摘自教材的介绍

在这个项目中，您将学习如何创建内核模块并将其加载到 Linux 内核中。然后，您将修改内核模块，以便它在 `/proc` 文件系统中创建一个条目。该项目可以使用随本文提供的 Linux 虚拟机完成。尽管您可以使用任何文本编辑器来编写这些 C 程序，但您必须使用终端应用程序来编译程序，并且您必须在命令行上输入命令来管理内核中的模块。

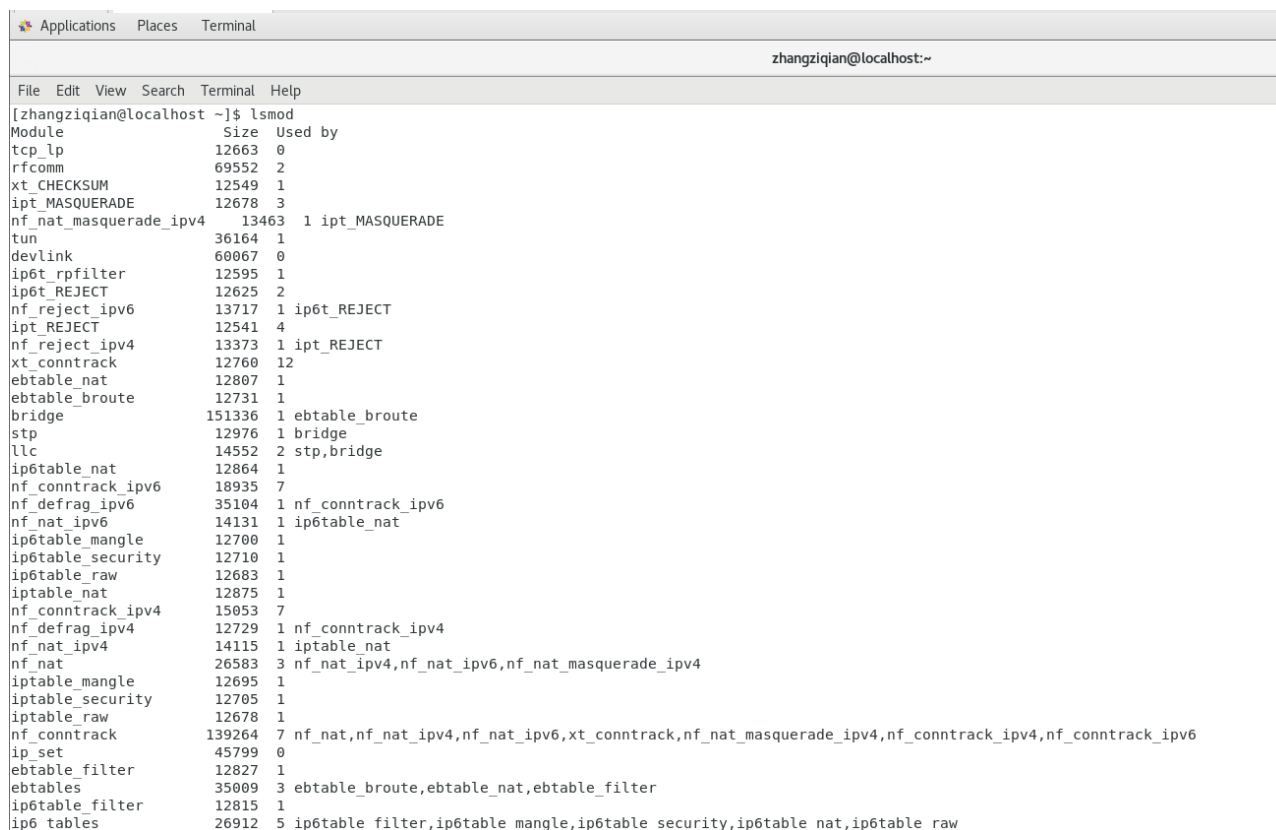
正如您将发现的那样，开发内核模块的优势在于它是一种与内核交互的相对简单的方法，因此您可以编写直接调用内核函数的程序。请务必记住，您确实在编写与内核直接交互的内核代码。这通常意味着代码中的任何错误都可能导致系统崩溃！但是，由于您将使用虚拟机，因此任何故障最多只需要重新启动系统即可。

### 0.1.1 第一部分 安装虚拟机

- 我这里使用的是 `CentOS-7` 的操作系统
- 安装方法在另外的一门课程中有详细的实验报告，可以参阅这个链接 [如何安装Linux在虚拟机](#)。
- 安装后可能还需要安装一些必备的软件，例如 `yum install yum install emacs` 以及 `gcc` 等，视情况而定。

### 0.1.2 第二部分 内核模块概述

- 该项目的第一部分涉及到创建模块并将其插入 Linux 内核的一系列步骤。您可以通过输入命令 `lsmod` 列出当前加载的所有内核模块。此命令将在三列中列出当前内核模块：名称、大小和模块正在使用的位置。
- 具体可以如下图所示：



```
[zhangziquan@localhost ~]$ lsmod
Module                  Size  Used by
tcp_lp                  12663  0
rfcomm                  69552  2
xt_CHECKSUM             12549  1
ipt_MASQUERADE          12678  3
nf_nat_masquerade_ipv4 13463  1 ipt_MASQUERADE
tun                     36164  1
devlink                 60067  0
ip6t_rpfilter           12595  1
ip6t_REJECT             12625  2
nf_reject_ipv6          13717  1 ip6t_REJECT
ipt_REJECT              12541  4
nf_reject_ipv4          13373  1 ipt_REJECT
xt_conntrack            12760  12
ebtable_nat             12807  1
ebtable_broute          12731  1
bridge                 151336 1 ebtable_broute
stp                     12976  1 bridge
llc                     14552  2 stp,bridge
ip6table_nat            12864  1
nf_conntrack_ipv6       18935  7
nf_defrag_ipv6          35104  1 nf_conntrack_ipv6
nf_nat_ipv6             14131  1 ip6table_nat
ip6table_mangle          12700  1
ip6table_security       12710  1
ip6table_raw            12683  1
iptables_nat            12875  1
nf_conntrack_ipv4       15053  7
nf_defrag_ipv4          12729  1 nf_conntrack_ipv4
nf_nat_ipv4             14115  1 iptable_nat
nf_nat                  26583  3 nf_nat_ipv4,nf_nat_ipv6,nf_nat_masquerade_ipv4
iptables_mangle         12695  1
iptables_security       12705  1
iptables_raw            12678  1
nf_conntrack            139264 7 nf_nat,nf_nat_ipv4,nf_nat_ipv6,xt_conntrack,nf_nat_masquerade_ipv4,nf_conntrack_ipv4,nf_conntrack_ipv6
ip_set                  45799  0
ebtable_filter          12827  1
ebtables                35009  3 ebtable_broute,ebtable_nat,ebtable_filter
ip6table_filter          12815  1
ip6_tables              26912  5 ip6table_filter,ip6table_mangle,ip6table_security,ip6table_nat,ip6table_raw
```

- 然后创建一个 `simple.c` 的文件夹，写入代码：

```
1  #include <linux/init.h>
2  #include <linux/module.h>
3  #include <linux/kernel.h>
4  #include <linux/hash.h>
5  #include <linux/gcd.h>
6  #include <linux/jiffies.h>
7  #include <asm/param.h>
8
9  /* This function is called when the module is loaded. */
10 int simple_init(void)
11 {
12     printk(KERN_INFO "GOLDEN_RATIO_PRIM value is:
13     %lu\n", GOLDEN_RATIO_PRIME);
14     //直接输出变量值
15     printk(KERN_INFO "gcd(3300,24) value is: %lu\n", gcd(3300,24));
16     //调用gcd函数 返回一个整数 然后打印输出
17     printk(KERN_INFO "jiffies value is: %lu\n", jiffies);
18     //在启动的时候打印输出信息
19
20     printk(KERN_INFO "Loading Module\n");
21     return 0;
22 }
23
24 /* This function is called when the module is removed. */
25 void simple_exit(void) {
26
27     printk(KERN_INFO "GOLDEN_RATIO_PRIM value is: %lu\n", gcd(3300,24));
28     printk(KERN_INFO "jiffies value is: %lu\n", jiffies);
29     printk(KERN_INFO "HZ value is: %d\n", HZ);
30     //在终止的时候打印输出信息
31     printk(KERN_INFO "Removing Module\n");
32 }
33
34 /* Macros for registering module entry and exit points. */
35 module_init( simple_init );
36 module_exit( simple_exit );
37
38 MODULE_LICENSE("GPL");
39 MODULE_DESCRIPTION("Simple Module");
40 MODULE_AUTHOR("SGG");
41
42
```

- 当然还有必不可少的 `makefile` 文件，如果没有这个文件的话，在执行 `make` 的时候程序会报错，就是说找不到 `include` 对应的头文件，所以推测这个文件是有关编译过程的配置文件，定义了使用到的一些库文件，这样编译的时候才能定位到相应的位置。

```

1 obj-m += simple.o
2 all:
3     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
4 clean:
5     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
6

```

- 创建好了这两个文件之后（第一个文件的后缀是 `.c`，第二个文件没有后缀），进入到这两个文件的父目录下，用终端命令行打开，然后输入命令 `make` 执行编译，编译后会生成一些列的文件：文件 `simple.ko` 代表编译后的内核模块。（编译后的命令行如下所示）

```

zhangziqian@localhost: ~/Desktop/assignment1
File Edit View Search Terminal Help
[zhangziqian@localhost assignment1]$ make
make -C /lib/modules/3.10.0-1160.53.1.el7.x86_64/build M=/home/zhangziqian/Desktop/assignment1 modules
make[1]: Entering directory `/usr/src/kernels/3.10.0-1160.53.1.el7.x86_64'
CC [M] /home/zhangziqian/Desktop/assignment1/simple.o
Building modules, stage 2.
MODPOST 1 modules
CC      /home/zhangziqian/Desktop/assignment1/simple.mod.o
LD [M] /home/zhangziqian/Desktop/assignment1/simple.ko
make[1]: Leaving directory `/usr/src/kernels/3.10.0-1160.53.1.el7.x86_64'
[zhangziqian@localhost assignment1]$

```

- 当然在正式开始前，我们前执行 `sudo dmesg -c` 命令，清除日志中的缓存区，通常来说，日志中的缓存区会很容易被填满，所以在开始前及时清除很有必要，清除的时候会打印已有的日志。

```

zhangziqian is not in the sudoers file.  This incident will be reported.
[zhangziqian@localhost assignment1]$ su
Password:
[root@localhost assignment1]# sudo dmesg -c
sudo: dmesg -c: command not found
[root@localhost assignment1]# sudo dmesg -c
[ 567.067987] simple: loading out-of-tree module taints kernel.
[ 567.068059] simple: module verification failed: signature and/or required key missing - tainting kernel
[ 567.069180] The /proc/seconds loaded!
[ 609.073748] The /proc/seconds unloaded!
[ 1733.621615] IPv6: ADDRCONF(NETDEV_UP): virbr0: link is not ready
[ 1733.688947] usb 2-2.1: USB disconnect, device number 4
[ 1733.852233] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 1733.853969] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 1733.855803] IPv6: ADDRCONF(NETDEV_UP): virbr0: link is not ready
[ 1733.855888] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 1733.856500] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 1733.857345] IPv6: ADDRCONF(NETDEV_UP): virbr0: link is not ready
[ 1733.982856] usb 2-2.1: new full-speed USB device number 5 using uhci_hcd
[ 1734.119003] usb 2-2.1: New USB device found, idVendor=0e0f, idProduct=0008, bcdDevice= 1.00
[ 1734.119010] usb 2-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1734.119016] usb 2-2.1: Product: Virtual Bluetooth Adapter
[ 1734.119032] usb 2-2.1: Manufacturer: VMware
[ 1734.119037] usb 2-2.1: SerialNumber: 000650268328
[ 1783.965491] e1000: ens33 NIC Link is Down
[ 1789.979016] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 1806.790364] perf: interrupt took too long (3663 > 3585), lowering kernel.perf_event_max_sample_rate to 54000
[ 2325.171075] e1000: ens33 NIC Link is Down
[ 2337.195750] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 3429.323887] perf: interrupt took too long (4617 > 4578), lowering kernel.perf_event_max_sample_rate to 43000
[root@localhost assignment1]# sudo dmesg -c
[root@localhost assignment1]# dmesg

```

- 加载这个内核模块只需要执行下面的命令即可：

```
1 | sudo insmod simple.ko
```

- 要检查模块是否已加载，只需要输入 `lsmod` 命令并在列表中搜索模块 `simple`。显然就在第一个，如下图所示。

```
[root@localhost assignment1]# dmesg
[root@localhost assignment1]# sudo insmod simple.ko
[root@localhost assignment1]# lsmod
Module                Size  Used by
simple                 12498  0
tcp_lp                12663  0
rfcomm                69552  2
xt_CHECKSUM           12549  1
ipt_MASQUERADE        12678  3
nf_nat_masquerade_ipv4 13463  1 ipt_MASQUERADE
tun                   36164  1
```

- 加载后，检查加载成功，然后再卸载模块，执行命令 `sudo rmmod simple`，即可卸载模块。
- 卸载后，输入命令 `dmesg` 检测日志，即可发现打印输出的一些数值，如下图所示。

```
~~~~~
[root@localhost assignment1]# sudo rmmod simple
[root@localhost assignment1]# dmesg
[ 3796.141398] GOLDEN_RATIO_PRIM value is: 11400862456688148481
[ 3796.141404] gcd(3300,24)value is: 12
[ 3796.141408] jiffies value is: 4298480089
[ 3796.141412] Loading Module
[ 3860.815891] GOLDEN_RATIO_PRIM value is: 12
[ 3860.815897] jiffies value is: 4298544762
[ 3860.815902] HZ value is: 1000
[ 3860.815907] Removing Module
[root@localhost assignment1]# █
```

- **注释：** 在 1.4.3 节中，我们学习了定时器的作用以及定时器中断处理程序。在 Linux 中，计时器滴答的速率（滴答速率）是在 `<asm/param.h>` 中定义的值 `HZ`。`HZ` 的值决定了定时器中断的频率，它的值因机器类型和架构而异。例如，如果 `HZ` 的值为 100，则定时器中断每秒发生 100 次，或每 10 毫秒发生一次。此外，内核跟踪全局变量 `jiffies`，它维护自系统启动以来发生的定时器中断的数量。`jiffies` 变量在文件 `<linux/jiffies.h>` 中声明。

### 0.1.3 第三部分 proc 文件系统

- 先看课本给的样例

```
1 | #include <linux/init.h>
2 | #include <linux/kernel.h>
3 | #include <linux/module.h>
4 | #include <linux/proc_fs.h>
5 | #include <asm/uaccess.h>
6 |
7 | #define BUFFER_SIZE 128                // 缓冲区大小128
```

```

8  #define PROC_NAME "hello"          // 定义这个进程名字为hello
9
10 /*forward Function declartion*/
11 ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t
    *pos);
12 // 声明了一个proc_read读进程函数
13
14 /*Declare the ower and read function*/
15 static struct file_operations proc_ops ={
16     .owner = THIS_MODULE,
17     .read = proc_read,
18 };
19
20 /*Begin call this function when the module loaded*/
21 int proc_init(void)
22 {
23     proc_create(PROC_NAME, 0666, NULL, &proc_ops);
24     return 0;
25 }
26
27 void proc_exit(void)
28 {
29     remove_proc_entry(PROC_NAME, NULL);
30     printk(KERN_INFO "The /proc/hello unloaded!\n");
31 }
32
33 /*Implementation of proc_read*/
34 ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count,
    loff_t *pos)
35 {
36     int rv=0;
37     char buffer[BUFFER_SIZE];
38     static int completed=0;
39     if (completed){
40         completed=0;
41         return 0;
42         /*proc_read逢0才停止，因此这个return 0保证只输出一值，否则将无限循环输出*/
43     }
44     completed=1;
45     rv=sprintf(buffer, "Hello World\n", );
46     copy_to_user(usr_buf, buffer, rv);
47     return rv;
48 }
49
50
51 module_init(proc_init);
52 module_exit(proc_exit);

```

```

53
54 MODULE_LICENSE("GPL");
55 MODULE_DESCRIPTION("None");
56 MODULE_AUTHOR("Book");
57

```

- 在模块入口点 `proc_init()` 中，我们使用 `proc_create()` 函数创建新的 `/proc/hello` 条目。这个函数被传递给 `proc_ops`，其中包含对结构文件操作的引用。这个结构初始化了 `.owner` 和 `.read` 成员。`.read` 的值是函数 `proc_read()` 的名称，每当读取 `/proc/hello` 时都会调用该函数。
- 在看 `proc_read()` 函数，字符串 `"Hello World\n"` 被写入变量缓冲区，其中缓冲区存在于内核内存中。由于 `/proc/hello` 可以从用户空间访问，我们必须使用内核函数 `copy to user()` 将缓冲区的内容复制到用户空间。该函数将内核内存缓冲区的内容复制到存在于用户空间中的变量 `usr_buf`。每次读取 `/proc/hello` 文件时，`proc_read()` 函数都会被重复调用，直到它返回 0，因此必须有逻辑确保该函数在收集到数据后返回 0（在这种情况下，字符串 `"Hello World\n"` 将进入相应的 `/proc/hello` 文件。最后，注意 `/proc/hello` 文件在模块退出点 `proc_exit()` 中使用函数 `remove_proc_entry()` 被删除。

- 基于以上的提示文件，我们适当修改，即可得到第一题的代码：

- 执行代码还是要经过一下步骤

- `su` 获取 `root` 权限
- `make` 编译，注意这里的是 `sample1` 所以对应的 `makedfile` 文件要改一下。
- 加载内核模块，`sudo insmod sample1`
- `cat /proc/jiffies` 执行，即可获得相应的 `jiffies` 数值。

```

1  /*Create a proc named seconds*/
2  #include <linux/init.h>
3  #include <linux/kernel.h>
4  #include <linux/module.h>
5  #include <linux/proc_fs.h>
6  #include <asm/uaccess.h>
7  #include <linux/jiffies.h>
8
9  #define BUFFER_SIZE 128
10 #define PROC_NAME "jiffies"
11
12 unsigned long int volatile intime_jiffies;
13
14
15 /*forward Function declartion*/
16 ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t
    *pos);
17

```

```

18  /*Declare the owner and read function*/
19  static struct file_operations proc_ops ={
20      .owner = THIS_MODULE,
21      .read = proc_read,
22      /*.read is the name of the function proc_read() that is to be called whenever
        /proc/hello is read.*/
23  };
24
25  /*Begin call this function when the module loaded*/
26  int proc_init(void)
27  {
28      proc_create(PROC_NAME, 0666, NULL, &proc_ops);
29      printk(KERN_INFO "The module has been loaded!\n");
30      return 0;
31  }
32
33  void proc_exit(void)
34  {
35      remove_proc_entry(PROC_NAME, NULL);
36      printk(KERN_INFO "The module has been unloaded!\n");
37  }
38
39  /*Implementation of proc_read*/
40  ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count,
        loff_t *pos)
41  {
42      int time_jiffies = jiffies;
43      int rv=0;
44      char buffer[BUFFER_SIZE];
45      static int completed=0;
46      if (completed){
47          completed=0;
48          return 0;
49          /*proc_read逢0才停止，因此这个return 0保证只输出一数值，否则将无限循环输出*/
50      }
51      completed=1;
52      rv=sprintf(buffer, "jiffies value is %d s\n", time_jiffies);
53      /*将jiffies的值存入buffer, rv为写入的字符总数，不包括字符串追加在字符串末尾的空字符
        */
54      copy_to_user(usr_buf, buffer, rv);
55      /*从内核的buffer拷贝到用户的usr_buf*/
56      return rv;
57  }
58
59  module_init(proc_init);
60  module_exit(proc_exit);
61

```

```

62 MODULE_LICENSE("GPL");
63 MODULE_DESCRIPTION("jiffies");
64 MODULE_AUTHOR("ZhangZiqian");

```

- 运行的参考图如下，可以看到，数值是实时的数值，这个数值会不断地增大。

```

[root@localhost assignment2]# cat /proc/jiffies
jiffies value is 5943835 s
[root@localhost assignment2]# cat /proc/jiffies
jiffies value is 5951295 s
[root@localhost assignment2]# cat /proc/jiffies
jiffies value is 5959649 s
[root@localhost assignment2]# cat /proc/jiffies
jiffies value is 5960845 s
[root@localhost assignment2]# cat /proc/jiffies
jiffies value is 5963232 s
[root@localhost assignment2]# cat /proc/jiffies
jiffies value is 5965511 s
[root@localhost assignment2]#

```

- 同样的道理，第二题只需要记录开始和最后的 `jiffies` 的数值：

$$\text{时间} = \frac{jiffies_{\text{终止}} - jiffies_{\text{开始}}}{HZ}$$

```

cat /proc/second: no such file or directory
[root@localhost assignment2]# cat /proc/second
The running time is 123 s
[root@localhost assignment2]# cat /proc/second
The running time is 130 s
[root@localhost assignment2]# cat /proc/second
The running time is 138 s
[root@localhost assignment2]#

```

- 运行的效果如上所示。