《操作系统》课第十一次实验报告

学院:	软件学院
姓名:	杨万里
学号:	2013774
邮箱:	2013774@mail.nankai.edu.cn
时间:	2022/11/25

0. 开篇感言

我们可以相信，一个成熟的工业产品，当你希望它拥有某些功能的时候，它的设计人员很有可能已经实现了这个功能。比如对于 Linux 内核功能的增加，之前几次实验我们的同学们都因为需要重新编译、安装、启动内核而感到很麻烦，实际上 Linux 内核模块的设计就很巧妙地避免了这些麻烦。

同时，通过自己摸索发现了加载模块函数的最后一条输出语句在卸载模块函数执行这样的特点，也解决了自己实验过程中的疑惑。

1. 实验题目

Linux 内核模块驱动（1）

2. 实验目标

（1） 写一个 C/C++程序；

（2） 实现一个 Linux 内核模块；

（3） 该模块可以实现列举当前状态所有进程的信息和总进程数。

（4） 需要列出每个进程的名字、进程 ID 号、父进程 ID 号、进程状态、学号姓名等。

3. 原理方法

很多时候我们希望能扩展 Linux 内核的功能，比如此前实验遇到的：希望能实现列举所有进程信息、希望在 Linux 内核中实现文件拷贝等等。

在之前我们**惯用的手段是增加一个系统调用**（先声明想要添加的系统调用；接着实现系统调用的具体内容；最后再注册一个系统调用号），但是希望系统调用生效需要重新编译并安装内核再重启机器，我们可以体会到这个过程的麻烦。

这时候再看 Linux 内核模块（LKM），会觉得非常讨人喜欢。Linux 内核模块是一些在**已经启动**的操作系统内核**需要时**可以**直接载入内核执行**的代码块，不需要时由操作系统卸载。它实现了在扩展内核功能的同时**避免重新编译、安装内核以及重启系统**的麻烦。同时也避免了操作系统过渡的臃肿，如果通过系统调用的方式增加很多功能，就会导致内核当中存在很多系统调用，即使平时不使用。而内核模块可以在需要的时候再载入内核当中执行，显然便捷不少。

Linux 内核模块的实现原理：

（1） 编写内核模块的 C 程序（本实验为 proinfomodule.c），其主要内容：

1）模块加载函数（本实验为 proinfomodule_init），其意义为：当通过 insmod 或 modprobe 命令**加载内核模块时**，模块的加载函数会自动被内核执行，完成本模块的相关初始化工作。

在本实验中，其主要实现的功能就是打印启动提示信息，并且列举所有进程信息及进程数目。

```
/* init function */
static int proinfomodule_init(void)
{
    struct task_struct *p = &init_task;
    int pro_num = 0;
    printk(KERN_ALERT "2013774ywl:simple module initialized\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Start 2013774ywl'S Module***************\n");
    printk("Hello! This is 2013774ywl's module to list all process.\n");
    for_each_process(p){
        printk("%-20s %6d %6d %4c 2013774ywl\n",p->comm,p->pid,p->parent-
>pid,task_state_to_char(p));
        pro_num++;
    }
    printk("There are %d processes 2013774ywl!", pro_num);
    return 0;
}
```

进程信息的列举和数目统计之前实验已经实现多次，原理就不再

赘述了。需要解释的是 jiffies 这个变量，这个变量记录了系统启动以来经历了多少 tick，tick 代表多长时间取决于内核的定义。也就代表了加载模块的时刻。

2）模块卸载函数（本实验为 proinfomodule_exit）其意义为：当通过 rmmod 命令卸载模块时，模块的卸载函数会自动被内核执行，完成卸载。

在本实验中，其主要功能就是打印提示模块卸载的信息。

```
/* exit function - logs that the module is being removed */
static void proinfomodule_exit(void)
{
    printk(KERN_ALERT "2013774ywl:simple module is being unloaded\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Goodbye 2013774ywl!***************\n");
}
```

此时打印 jiffies 代表了卸载模块的时刻，与加载模块时刻对应。

3）将前文编写的 proinfomodule_init 和 proinfomodule_exit 正式声明为该模块的模块加载函数和模块卸载函数。

```
module_init(proinfomodule_init);
module_exit(proinfomodule_exit);
```

4）模块许可证（LICENSE）声明：描述**内核模块的许可权限**，如果不声明 LICENSE，模块被加载时将收到内核被污染的警告。毕竟操作系统的内核还是要严格保护，不能随意修改的。

多数情况内核模块遵循 **GPL 兼容许可权**。

模块作者、描述和版本的声明。

```
MODULE_LICENSE ("GPL");
MODULE_AUTHOR ("LKD");
MODULE_DESCRIPTION ("Simple Kernel Module");
MODULE_VERSION("1.01");
```

以上则为内核模块的 C 程序需要编写的全部内容。

（2） 编写 Makefile 文件：

1）关于模块文件名称 proinfomodule.c 等信息的声明：

```
ModuleName=proinfomodule
obj-m +=${ModuleName}.o
all:${ModuleName}.ko
${ModuleName}.ko:${ModuleName}.c
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

2）定义加载模块的指令 testload 以及加载模块的具体动作：

```
testload:${ModuleName}.ko
        sudo dmesg -C
        sudo insmod ${ModuleName}.ko
        sudo dmesg | grep 2013774ywl
```

可以发现加载模块实际上的动作即开启 dmesg 信息管道，并且通过 insmod 执行加载模块函数，执行完毕，筛选包含“2013774ywl”（学号姓名）的信息。

3）定义卸载模块的指令 testunload 以及卸载模块的具体动作：

```
testunload:${ModuleName}.ko
        sudo dmesg -C
        sudo rmmod ${ModuleName}.ko
        sudo dmesg | grep 2013774ywl
```

开启 dmesg 信息管道，通过 rmmod 执行卸载模块函数，执行完毕，筛选包含“2013774ywl”的信息。

以上则为整个实验的原理方法。

4. 具体步骤

（1） 按照上一小节的原理方法编写 proinfomodule.c 文件和 Makefile 文件。

proinfomodule.c

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/task.h>
#include <linux/sched/signal.h>

/* init function */
static int proinfomodule_init(void)
{
    struct task_struct *p = &init_task;
    int pro_num = 0;
    printk(KERN_ALERT "2013774ywl:simple module initialized\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Start 2013774ywl'S Module***************\n");
    printk("Hello! This is 2013774ywl's module to list all process.\n");
    for_each_process(p){
        printk("%-20s %6d %6d %4c 2013774ywl\n",p->comm,p->pid,p->parent->pid,task_state_to_char(p));
        pro_num++;
    }
    printk("There are %d processes 2013774ywl!", pro_num);
    return 0;
}

/* exit function - logs that the module is being removed */
static void proinfomodule_exit(void)
{
    printk(KERN_ALERT "2013774ywl:simple module is being unloaded\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Goodbye 2013774ywl!***************\n");
}

module_init(proinfomodule_init);
module_exit(proinfomodule_exit);

MODULE_LICENSE ("GPL");
MODULE_AUTHOR ("LKD");
MODULE_DESCRIPTION ("Simple Kernel Module");
MODULE_VERSION("1.01");
```

Makefile

```
ModuleName=proinfomodule
obj-m +=${ModuleName}.o
all:${ModuleName}.ko
${ModuleName}.ko:${ModuleName}.c
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
testload:${ModuleName}.ko
	sudo dmesg -C
	sudo insmod ${ModuleName}.ko
	sudo dmesg | grep 2013774ywl
testunload:${ModuleName}.ko
	sudo dmesg -C
	sudo rmmod ${ModuleName}.ko
	sudo dmesg | grep 2013774ywl

```

（2） 编译内核模块：

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$ make
make -C /lib/modules/6.0.9/build M=/home/wanliyang2013774/OSWorkSpace/lab11_code
 modules
make[1]: Entering directory '/home/wanliyang2013774/linux-6.0.9'
  CC [M]  /home/wanliyang2013774/OSWorkSpace/lab11_code/proinfomodule.o
  MODPOST /home/wanliyang2013774/OSWorkSpace/lab11_code/Module.symvers
  CC [M]  /home/wanliyang2013774/OSWorkSpace/lab11_code/proinfomodule.mod.o
  LD [M]  /home/wanliyang2013774/OSWorkSpace/lab11_code/proinfomodule.ko
make[1]: Leaving directory '/home/wanliyang2013774/linux-6.0.9'
```

可以看出编译内核模块的时候，操作系统进入了 /home/wanliyang2013774/linux-6.0.9 文件夹进行了一些操作，然后离开内核文件夹。

（3） 执行加载模块的指令 make testload：

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$ make testload
sudo dmesg -C
[sudo] password for wanliyang2013774:
sudo insmod proinfomodule.ko
sudo dmesg | grep 2013774ywl
[ 1339.893083] 2013774ywl:simple module initialized
[ 1339.893093] 2013774ywl:Jiffies value: 4295226869
[ 1339.893095] ***************Start 2013774ywl'S Module***************
[ 1339.893096] Hello! This is 2013774ywl's module to list all process.
[ 1339.893098] systemd                  1      0    S 2013774ywl
[ 1339.893102] kthreadd                 2      0    S 2013774ywl
[ 1339.893105] rcu_gp                   3      2    I 2013774ywl
[ 1339.893108] rcu_par_gp               4      2    I 2013774ywl
[ 1339.893110] slub_flushwq             5      2    I 2013774ywl
[ 1339.893112] netns                    6      2    I 2013774ywl
[ 1339.893115] kworker/0:0H             8      2    I 2013774ywl
```

……

```
[ 1339.893833] bash                  3213   3182    S 2013774ywl
[ 1339.893836] make                  3512   3213    S 2013774ywl
[ 1339.893838] sudo                  3516   3512    S 2013774ywl
[ 1339.893841] sudo                  3517   3516    S 2013774ywl
[ 1339.893843] insmod                3518   3517    R 2013774ywl
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$
```

可以看出如期执行了 proinfomodule.c 文件当中的 proinfomodule_init 函数，也就是加载模块函数。输出了模块开始的信息

（包括时间等）以及当前所有进程的相关信息（进程名称、进程号、父进程号、进程状态）和学号姓名。

（4） 执行卸载模块的指令 make testunload：

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$ make testunload
sudo dmesg -C
sudo rmmod proinfomodule.ko
sudo dmesg | grep 2013774ywl
[ 1339.893845] There are 308 processes 2013774ywl!
[ 1549.991848] 2013774ywl:simple module is being unloaded
[ 1549.991869] 2013774ywl:Jiffies value: 4295279391
[ 1549.991871] ***************Goodbye 2013774ywl!***************
```

此时，执行了 proinfomodule.c 文件当中的 proinfomodule_exit 函数，也就是卸载模块函数。输出了模块结束的信息。

以上则完成了本次实验的内容。

但是可以发现一个有趣的问题：

```
static int proinfomodule_init(void)
{
    struct task_struct *p = &init_task;
    int pro_num = 0;
    printk(KERN_ALERT "2013774ywl:simple module initialized\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Start 2013774ywl'S Module***************\n");
    printk("Hello! This is 2013774ywl's module to list all process.\n");
    for_each_process(p){
        printk("%-20s %6d %6d %4c 2013774ywl\n",p->comm,p->pid,p->parent->pid,task_state_to_char(p));
        pro_num++;
    }
    printk("There are %d processes 2013774ywl!", pro_num);
    return 0;
}
```

实际上，我是在**加载模块函数** proinfomodule_init 当中执行了**打印总进程数**的指令，但是**最终是在执行卸载模块**的指令时才**输出了总进程数目** 308。

可以发现**输出进程数的语句**是加载模块函数的**最后一条输出语句**，我们不妨在这之后加一条用于测试的输出语句，如下图所示：

```
/* init function */
static int proinfomodule_init(void)
{
    struct task_struct *p = &init_task;
    int pro_num = 0;
    printk(KERN_ALERT "2013774ywl:simple module initialized\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Start 2013774ywl'S Module***************\n");
    printk("Hello! This is 2013774ywl's module to list all process.\n");
    for_each_process(p){
        printk("%-20s %6d %6d %4c 2013774ywl\n",p->comm,p->pid,p->parent->pid,task_state_to_char(p));
        pro_num++;
    }
    printk("There are %d processes 2013774ywl!", pro_num);
    printk("test2013774ywl");
    return 0;
}
```

再次进行测试：

```
[ 1640.168475] sudo                  5258    5257    S 2013774ywl
[ 1640.168477] insmod                5259    5258    R 2013774ywl
[ 1640.168479] There are 310 processes 2013774ywl!
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$ make testu
nload
sudo dmesg -C
sudo rmmod proinfomodule.ko
sudo dmesg | grep 2013774ywl
[ 1640.168480] test2013774ywl
[ 1729.850291] 2013774ywl:simple module is being unloaded
[ 1729.850308] 2013774ywl:Jiffies value: 4295324383
[ 1729.850310] ***************Goodbye 2013774ywl!***************
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$
```

可以发现加载模块函数输出了总进程数，但是其中最后一条输出信息“test2013774ywl”在卸载模块函数部分打印。

不妨再增加一条测试语句：

```
/* init function */
static int proinfomodule_init(void)
{
    struct task_struct *p = &init_task;
    int pro_num = 0;
    printk(KERN_ALERT "2013774ywl:simple module initialized\n");
    printk("2013774ywl:Jiffies value: %lu\n", jiffies);
    printk("***************Start 2013774ywl'S Module***************\n");
    printk("Hello! This is 2013774ywl's module to list all process.\n");
    for_each_process(p){
        printk("%-20s %6d %6d %4c 2013774ywl\n",p->comm,p->pid,p->parent->pid,task_state_to_char(p));
        pro_num++;
    }
    printk("There are %d processes 2013774ywl!", pro_num);
    printk("test2013774ywl");
    printk("TEST2013774ywl");
    return 0;
}
```

测试一下：

```
[ 1957.490679] insmod                5582    5581    R 2013774ywl
[ 1957.490682] There are 308 processes 2013774ywl!
[ 1957.490684] test2013774ywl
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab11_code$ make testu
nload
sudo dmesg -C
sudo rmmod proinfomodule.ko
sudo dmesg | grep 2013774ywl
[ 1957.490686] TEST2013774ywl
[ 1963.542817] 2013774ywl:simple module is being unloaded
[ 1963.542841] 2013774ywl:Jiffies value: 4295382805
[ 1963.542844] ***************Goodbye 2013774ywl!***************
```

可以发现，系统总会把**加载模块函数的最后一条语句放到卸载模块函数部分执行**。但是不是很明确这么做的意义，如果老师能解答就好了，谢谢老师！

因此，前面出现的问题也就可以解释了：输出进程总数的语句是加载模块函数的最后一条执行语句，因此放到了卸载模块函数执行输出。

5. 总结心得

（1） 通过本次实验，我了解到了系统内核模块这一概念，相较于内核系统调用，

内核模块在增加功能这方面要便利不少。这也让我体会到了 Linux 内核设计的精妙；

（2） 通过编写 Makefile 文件让我更深入地了解了 Makefile 的相关用法；

（3） 同时通过自己的探索，摸索出了操作系统内核模块的一些执行特点。

6. 参考资料

实验指导书《README_kernelmodulesimple.pdf》

7. 源代码

proinfomodule.c 文件：

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/task.h>
#include <linux/sched/signal.h>

/* init function */
static int proinfomodule_init(void)
{
        struct task_struct *p = &init_task;
        int pro_num = 0;
        printk(KERN_ALERT "2013774ywl:simple module initialized\n");
        printk("2013774ywl:Jiffies value: %lu\n", jiffies);
        printk("***************Start 2013774ywl'S Module***************\n");
        printk("Hello! This is 2013774ywl's module to list all process.\n");
        for_each_process(p){
            printk("%-20s %6d %6d %4c 2013774ywl\n",p->comm,p->pid,p->parent->pid,task_state_to_char(p));
            pro_num++;
        }
        printk("There are %d processes 2013774ywl!", pro_num);
        return 0;
```

```
22.  }
23.
24.  /* exit function - logs that the module is being r
   emoved */
25.  static void proinfomodule_exit(void)
26.  {
27.      printk(KERN_ALERT "2013774ywl:simple module is
    being unloaded\n");
28.      printk("2013774ywl:Jiffies value: %lu\n", jiff
   ies);
29.      printk("***************Goodbye 2013774ywl!****
   ***********\n");
30.  }
31.
32.  module_init(proinfomodule_init);
33.  module_exit(proinfomodule_exit);
34.
35.  MODULE_LICENSE ("GPL");
36.  MODULE_AUTHOR ("LKD");
37.  MODULE_DESCRIPTION ("Simple Kernel Module");
38. MODULE_VERSION("1.01");
```

Makefile 文件：

```
1. ModuleName=proinfomodule
2. obj-m +=${ModuleName}.o
3. all:${ModuleName}.ko
4. ${ModuleName}.ko:${ModuleName}.c
5.     make -C /lib/modules/$(shell uname -
   r)/build M=$(PWD) modules
6. testload:${ModuleName}.ko
7.     sudo dmesg -C
8.     sudo insmod ${ModuleName}.ko
9.     sudo dmesg | grep 2013774ywl
10.  testunload:${ModuleName}.ko
11.      sudo dmesg -C
12.      sudo rmmod ${ModuleName}.ko
13.      sudo dmesg | grep 2013774ywl
```