苏州大学实验报告

院、系	计算机学院	年级专业	20 计科	姓名	柯骅	学号	2027405033
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无		实验日期	2023.05.11	

实 验 名 称 实验 8 内核模块编写

一、 实验目的

1. 理解针对 Linux 提出内核模块这种机制的意义。
2. 理解并掌握 Linux 实现内核模块机制的基本技术路线。
3. 运用 Linux 提供的工具和命令，掌握操作内核模块的方法。
4. 掌握利用内核模块（机制）实现较复杂功能的方法。
5. 学习并掌握/proc 文件系统。

二、 实验内容

1. **（编写一个简单的内核模块）**

   本实验室内核模块的演示，旨在帮助读者理解和掌握如何进行内核模块的编写与载入。具体的实验内容是编写一个简单的具备基本要素的内核模块，并编写这个内核模块所需要的 Makefile，最后编译内核并将其载入系统。

2. **（利用内核模块实现/proc 文件系统）**

   利用内核模块在/proc 目录下创建 proc_example 目录，并在该目录下创建三个普通文件（foo、bar、jiffies）和一个文件链接（jiffies_too）

三、 实验步骤

1. **（编写一个简单的内核模块）**

Linux 内核模块介绍：

Linux 操作系统的内核是单体系结构(monolithic kernel)的，即整个内核是一个单独的非常大的程序。这样的操作系统内核把所有的模块都集成在了一起，系统的速度和性能都很好，但是可扩展性和可维护性就相对比较差。

为了改善单一体系结构内核的可扩展性和可维护性，Linux 操作系统使用了一种全新的内核模块机制——动态可加载内核模块(loadable kernel module,LKM)。用户可以根据需求，在不需要对内核重新编译的情况下，让模块能动态地装入内核或从内核移出。模块扩展了内核的功能，而无须重启系统。模块不是作为进程执行的，而是像其他静态连接的内核函数一样，在内核态代表当前进程执行。

模块与内核是在同样的地址空间中运行的，因此模块编程在一定意义上也就是内核编程。但并不是内核中所有的功能都可以使用模块来实现。Linux 内核中极为重要的一些功能，如进程管理、内存管理等，仍难以通过模块来实现，而必须直接对内核进行修改才能实现。在 Linux 系统中，经常利用内核模块实现的有文件系统、SCSI 高级驱动程序、大部分的 SCSI 普通驱动程序、多数 CD-ROM 驱动程序、以太网驱动程序等。

内核模块的使用：

内核模块必须至少有两个函数:一个是名为 init_module()的初始化函数，其会在模块被载入内核时被调用;另一个是名为 cleanup_module()的清理函数，其只会在模块被卸载之前被调用。也就是说，init_module()和 cleanup_module()函数分别是在执行 insmod 和 rmmod 命令的时候被调用的，并且 insmod 和 rmmod 命令只识别这两个特殊的函数。

但实际上，从 Linux 内核 2.3.13 版本开始，情况就发生了变化，现在用户可以自己定义任何名称来作为模块的开始和结束函数。但是，还是有许多人仍在使用 init_module()和 cleanup_module()作为模块的开始和结束函数。通常，init_module()要么为内核注册一个处理程序，要么用自己的代码替换其中的一个内核函数；cleanup_module()函数由于能够撤销 init_module()所做的任何操作，因此可以安全地卸载模块。

最后，每个内核模块都需要包含 linux/module.h。

命令介绍：

在使用内核模块时，会用到 Linux 为此开发的一些内核模块操作命令，如 lsmod、insmod、rmmod

```
# lsmod    //用于列出当前已加载的模块。
# modinfo  //查看模块的基本信息
# insmod   //用于加载模块。
# rmmod    //用于删除模块。
```

为了完成一个简单的具备基本要素的内核模块,需要执行的具体步骤操作包括内核模块源程序的编写、编译、卸载、装载及卸载等。本实验给出的内核模块源代码功能非常简单——仅在控制台输出“Hello World!”之类的字符串。具体步骤如下：

(1) 编写内核模块源代码文件

代码如下，详见附件/helloworld/helloworld.c:

```
1. #define MODULE
2. #include<linux/module.h>
3. int init_module(void){
4.     printk("<1>Hello World!\n");
5.     return 0;
6. }
7.  void cleanup_module(void){
8.     printk("<1>Goodbye!");
9. }
10. MODULE_LICENSE("GPL");
```

其中，init_module()函数会在 helloworld 模块被载入内核时，即执行 insmod 命令时被调用，cleanup_module()函数会在 helloworld 模块被卸载时，即执行 rmmod 命令时被调用。可以通过 dmesg 命令查看调用结果。

(2) 编写编译内核模块时要用到的 Makefile 文件

代码如下，详见附件/helloworld/Makefile:

```
1. obj-m+=helloworld.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
```

(3) 编译 helloworld.c

```
# make
```

执行如上命令编译后，可以得到模块文件 helloworld.ko,结果如下：

```
kh@ubuntu:~/test/test8/helloworld$ make
make -C /lib/modules/5.4.0-147-generic/build/ M=/home/kh/test/test8/helloworld modules
make[1]: 进入目录“/usr/src/linux-headers-5.4.0-147-generic”
  CC [M]  /home/kh/test/test8/helloworld/helloworld.o
/home/kh/test/test8/helloworld/helloworld.c:1:0: warning: "MODULE" redefined
 #define MODULE

<command-line>:0:0: note: this is the location of the previous definition
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/kh/test/test8/helloworld/helloworld.mod.o
  LD [M]  /home/kh/test/test8/helloworld/helloworld.ko
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-147-generic”
```

(4) 执行内核模块装入命令

执行如下命令将 helloworld 模块加载进内核：

```
# sudo insmod helloworld.ko
```

可以通过 dmesg 命令查看控制台输出，预期结果为“<1>Hello World!”,命令如下：

```
# sudo dmesg | tail
```

也可以使用 lsmod 命令查看模块信息。lsmod 命令的作用是列出内核中运行的所有模块的信息，包括模块的名称、占用空间的大小、当前状态以及依赖性等，命令如下：

```
# lsmod
```

(5) 当不需要使用 helloworld 模块时，就卸载这个模块

命令如下：

```
# sudo rmmod helloworld
```

可以通过 dmsg 命令查看控制台的输出，预期结果为“<1>Goodbye!”

2. （利用内核模块实现/proc 文件系统）

/proc 文件简介：

/proc 文件系统是一个伪文件系统，它只存在内存当中，而不占用外存空间。它以文件系统的方式为访问系统内核数据的操作提供接口。用户和应用程序可以通过/proc 得到系统的信息，并可以改变内核的某些参数。由于系统的信息，如进程，是动态改变的，所以用户或应用程序读取/proc 文件时，/proc 文件系统是动态从系统内核读出所需信息并提交的。

具体步骤如下：

(1) 编写 procfs_example.c 文件

在初始化函数 static int __init init_procfs_example(void)中

通过 proc_mkdir 创建目录/procfs_example

通过 proc_create 创建 jiffies，内容为系统启动后经过的时间戳

通过 proc_create 创建 foo，并写入 name 和 value 都为“foo”的内容

通过 proc_create 创建 bar，并写入 name 和 value 都为“bar”的内容

通过 proc_symlink 创建 jiffies_too，该文件是 jiffies 的符号链接

在清理函数 static void __exit cleanup_procfs_example(void) 中

通过 remove_proc_entry 删除目录和文件

代码详见附件/procfs_example/procfs_example.c

(2) 编写 Makefile 文件

代码详见附件/procfs_example/procfs_example.c

(3) 编译代码，装入模块，卸载模块

命令如下：

```
# sudo insmod procfs_example.ko //安装模块
# cd /proc/procfs_example //进入 proc 目录
# cat bar foo jiffies jiffies_too //指定的四个文件
# ls -l //查看他们的属性
# vim jiffies //编辑 jiffies
```

四、 实验结果

1. （编写一个简单的内核模块）

(1) 装载模块

```
# sudo insmod helloworld.ko
```

可以通过 dmesg 命令查看控制台输出，如下图所示：

```
# sudo dmesg | tail
```

```
[ 1002.278945] <1>Hello World!
```

这时，可以看到输出结果“<1>Hello World!”，此内容实在 init_module()函数中定义的。由此说明，helloworld 模块已经成功被装载到了内核中。

也可以使用 lsmod 命令查看模块信息，如下图所示：

```
# lsmod
```

```
kh@ubuntu:~/test/test8/helloworld$ lsmod
Module                  Size  Used by
helloworld             16384  0
btrfs                1249280  0
xor                    24576  1 btrfs
```

可以看到，系统中存在名为 helloworld 的模块，其大小为 16 384 字节。

(2) 卸载模块

当不需要 helloworld 模块时，可以通过如下命令卸载这个模块：

```
# sudo rmmod helloworld
```

可以通过如下 dmesg 命令查看控制台的输出：

```
# sudo dmesg | tail
```

结果如下：

```
[ 6646.703918] <1>Goodbye!
```

此时，可以看到输出结果为“<1>Goodbye!”，此内容实在 cleanup_module()函数中定义的。由此说明，helloworld 模块已被删除。如果这时候再使用 lsmod 命令，就会发现 helloworld 模块已经不存在了。

2. （利用内核模块实现/proc 文件系统）

(1) 装载模块

执行如下命令装载模块并查看是否成功：

```
# sudo insmod procfs_example //装载模块
# lsmod | grep procfs //查找模块
```

运行结果如下：

```
kh@ubuntu:~/test/test8/procfs_example$ sudo insmod procfs_example.ko
kh@ubuntu:~/test/test8/procfs_example$ lsmod | grep procfs
procfs_example         16384  0
```

可以发现成功查找到了大小为 16 384 字节的 procfs_example 模块，表明装载成功。

(2) 查看/proc/porcfs_example 目录中个文件及属性

命令如下：

```
# cd /proc/procfs_example
# cat bar foo jiffies jiffies_too
# ls -l
```

运行结果如下：

```
kh@ubuntu:~/test/test8/procfs_example$ cd /proc/procfs_example
kh@ubuntu:/proc/procfs_example$ cat bar foo jiffies jiffies_too
bar='bar'
foo='foo'
jiffies=4298845400
jiffies=4298845400
kh@ubuntu:/proc/procfs_example$ ls -l
总用量 0
-r--r--r-- 1 root root 0 5月   9 19:04 bar
-r--r--r-- 1 root root 0 5月   9 19:04 foo
-r--r--r-- 1 root root 0 5月   9 19:04 jiffies
lrwxrwxrwx 1 root root 7 5月   9 19:04 jiffies_too -> jiffies
```

可以发现成功查看了指定的四个文件的属性。

(3) 尝试修改 jiffies 文件

使用如下命令尝试修改 jiffies 文件：

```
# vim jiffies
```

发现文件为只读文件：

```
"jiffies" [只读] 1L, 19C
```

尝试修改结果如下：

```
"jiffies"
"jiffies" E212: 无法打开并写入文件
请按 ENTER 或其它命令继续
```

表明只读文件无法写入。

五、 实验思考与总结

1. 说明为什么内核源代码中的输出函数选用了printk()而不是常用的printf()。

printf()在终端显示,printk()函数为内核空间里边的信息打印函数，就像c编程时用的printf()函数一样，专供内核中的信息展示用。在内核源代码中没有使用printf()的原因是在编译内核时还没有C的库函数可以供调用。

2. 思考bar、foo、jiffies 和jiffies_too文件分别是什么类型，它们是否可以进行读写。

bar和foo为可读文件，jiffies为只读文件，jiffies_too为jiffies的链接文件（可读可写）。

3. dmesg 是一种程序，用于检测和控制内核环缓冲。程序用来帮助用户了解系统的启动信息。

常见用法：

```
# dmesg | tail (-[行数 n])
# dmesg | head (-[行数 n])
```

查看 dmesg 命令的头部（head）或末尾（tail）n 行日志

```
# dmesg | grep (-i) [string]
```

过滤出包含 string 字符串的日志,-i 表示忽略大小写

```
# dmesg -c
```

清空 dmesg 日志

```
# watch “dmesg | tail -20”
```

实时监控 dmesg 日志输出

4. 实验总结

本次实验是 Linux 内核模块编写的实验，在实验一中，编写了一个简单的内核模块，并使用 insmod、rmmmod 命令将它在内核中装载、卸载，Linux 可以在内核中动态地添加模块，这一机制无需重复编译内核，有效地降低了添加模块的复杂度。

在第二个实验中，利用内核模块在/proc 目录下创建了 proc_example 目录，并在该目录下创建了三个普通文件（foo,bar,jiffies）和一个文件链接（jiffies_too），在此过程中，掌握了利用内核模块（机制）实现较复杂功能（例如创建目录与文件）的方法，同时学习并掌握了/proc 文件系统的相关知识。