Lab 1 : Module Programming

实 验 报 告

刘子涵　518021910690

【1】实验要求

编写四个 Linux 内核模块，分别实现以下功能：

- Module 1：加载和卸载模块时在系统日志中输出消息
- Module 2：支持整型、字符串、数组参数，加载时读入并在系统日志中输出
- Module 3：在 `/proc` 下创建只读文件
- Module4：在 `/proc` 下创建文件夹，并创建一个可读可写的文件

【2】实验环境

- Linux 内核版本：5.4.0
- GCC 版本：9.3.0
- 操作系统：Ubuntu 20.04

【3】实验思路及具体实现

Module 1

- Makefile 编写：调用 `/lib/modules/../build` 中的顶层 Makefile 进行编译。代码如下（下同）：

```
obj-m:=mod1.o
KDIR:=/lib/modules/$(shell uname -r)/build
PWD:=$(shell pwd)
all:
    make -C $(KDIR) M=$(PWD) modules
clean:
    make -C $(KDIR) M=$(PWD) clean
```

- C 源文件编写：

  需要为模块定义初始化函数（ `__init` ）和退出函数（ `__exit` ），在这两个函数中使用 `printk` 函数打印内核消息（可以使用 `dmesg` 命令查看），前三个字符表示消息级别（loglevel），此处的 `KERN_INFO` 在 `<linux/kernel.h>` 中有定义。还需要用 `module_init` 和 `module_exit` 宏加载定义的函数。代码如下：

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

static int __init mod1_init(void) {
    printk(KERN_INFO "Insert Module 1\n");
    return 0;
}

static void __exit mod1_exit(void) {
```

```
    printk(KERN_INFO "Remove Module 1\n");
}

module_init(mod1_init);
module_exit(mod1_exit);
```

Module 2

- C 源文件编写：

  首先声明用于接收参数的变量（`int` 整型、 `char*` 字符串、`int[]` 整型数组）。使用 `module_param` 宏来指定模块接收的整型和字符串参数，而对于数组，需要使用 `module_param_array` ，第二个参数是数组类型，第三个是参数个数的地址。这两个宏均定义在 `<linux/moduleparam.h>` 。同样用 `printk` 函数打印这些参数到消息中。

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/moduleparam.h>

static int a;
static char *b;
static int n_para = 1; // 4 parameters
static int c[4]; // 4 parameters

module_param(a, int, 0644);
module_param(b, charp, 0644);
module_param_array(c, int, &n_para, 0644);

static int __init mod2_init(void) {
    printk(KERN_INFO "Insert Module 2:\na = %d, b = %s, c = [%d, %d, %d,
%d]\n", a, b, c[0], c[1], c[2], c[3]);
    return 0;
}

static void __exit mod2_exit(void) {
    printk(KERN_INFO "Remove Module 2\n");
}

module_init(mod2_init);
module_exit(mod2_exit);
```

Module 3

- proc 编程常用函数：

```
/* 创建 proc 文件 */
static inline struct proc_dir_entry *proc_create (
    const char *name, // proc 文件的名字
    umode_t mode, // 权限，如0644
    struct proc_dir_entry *parent, // 父目录，如果为 NULL ， 则在 /proc 创建
    const struct file_operations *proc_fops // 文件操作结构体指针
);

/* 创建目录 */
struct proc_dir_entry *proc_mkdir (
    const char *name, // 目录的名字
```

```
    struct proc_dir_entry *parent // 父目录
);

/* 删除 proc 文件 */
void proc_remove (
    struct proc_dir_entry *de // 需要删除的目录指针
);
```

- C 源文件编写：

  在模块初始化函数 `mod3_init` 中，使用 `proc_create` 函数创建一个 proc 文件。文件名为 `mod3_proc`，权限为 `0444`，即对所有用户均为只读权限，父目录为 `NULL`，说明直接在根目录（`/proc`）创建该文件。另外，将指向自定义文件操作结构体 `file_operations` 的指针作为第四个参数。函数返回一个 proc 目录结构体，存储在全局变量 `entry` 中。

  在模块退出函数 `mod3_exit` 中，使用 `proc_remove` 函数删除此 proc 文件。

  定义文件操作集合 `myops`，将自己定义的 `myread` 函数作为操作集合的读属性。

  在自己定义的 `myread` 函数中，先初始化一个固定大小的缓冲区 `char buf[BUFSIZE]` 和一个读指针 `len`。然后使用 `sprintf` 函数将需要显示的字符串写入缓冲区，并更新读指针位置。将读指针赋值给 `*ppos`，并返回 `len`。

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#define BUFSIZE 100

MODULE_LICENSE("GPL");

static struct proc_dir_entry *entry;

static ssize_t myread(struct file *file, char __user *ubuf, size_t count,
loff_t *ppos) {
    char buf[BUFSIZE];
    int len = 0;
    if(*ppos > 0 || count < BUFSIZE)
        return 0;
    len += sprintf(buf, "mod3: %s\n", "hello world");

    if(copy_to_user(ubuf, buf, len))
        return -EFAULT;
    *ppos = len;
    return len;
}

static struct file_operations myops = {
    .owner = THIS_MODULE,
    .read = myread
};

static int __init mod3_init(void) {
    entry = proc_create("mod3_proc", 0444, NULL, &myops);
    return 0;
}
```

```
static void __exit mod3_exit(void) {
    proc_remove(entry);
}

module_init(mod3_init);
module_exit(mod3_exit);
```

Module 4

- C 源文件编写：

  与 Module 3 不同，这里需要在 `/proc` 下创建一个文件夹，并在里面创建 proc 文件，需要实现读写操作。读操作类似，这里不再赘述。

  写操作的实现需要自己定义写函数，比如 `mywrite` 函数。同样需要定义一个缓冲区 `char buf[BUFSIZE]`，定义一个临时接收用户输入的数组 `char tmp[MAXSIZE]` 。使用 `sscanf` 函数将命令行输入读入 `tmp` 中，赋值给全局变量 `str` 。然后计算缓冲区字符个数并返回。

  另外，`/proc` 下创建一个文件夹只需要使用 `proc_mkdir` 函数，名字为 `proc4_folder_2` ，父目录为 `NULL` ，即创建目录 `/proc/proc4_folder_2` ，返回一个 proc 目录结构体指针 `base` 。然后，`proc_create` 可以以此目录为父目录创建 proc 文件。退出函数需要删除所有创建的目录。

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#define BUFSIZE 100
#define MAXSIZE 1024

MODULE_LICENSE("GPL");

static char str[MAXSIZE];

static struct proc_dir_entry *entry;
static struct proc_dir_entry *base;

static ssize_t myread(struct file *file, char __user *ubuf, size_t count,
loff_t *ppos) {
    char buf[BUFSIZE];
    int len = 0;
    if(*ppos > 0 || count < BUFSIZE)
        return 0;
    len += sprintf(buf, "mod4: %s\n", str);
    printk(KERN_INFO "read from proc file: %s", str);

    if(copy_to_user(ubuf, buf, len))
        return -EFAULT;
    *ppos = len;
    return len;
}

static ssize_t mywrite(struct file *file, const char __user *ubuf, size_t
count, loff_t *ppos)
{
    char buf[BUFSIZE];
    if(*ppos > 0 || count > BUFSIZE)
        return -EFAULT;
```

```
    if(copy_from_user(buf, ubuf, count))
        return -EFAULT;
    char tmp[MAXSIZE];
    sscanf(buf, "%s", tmp);
    strcpy(str, tmp);
    printk(KERN_INFO "write (%s) to proc file", str);
    int c = strlen(buf);
    *ppos = c;
    return c;
}

static struct file_operations myops = {
    .owner = THIS_MODULE,
    .read = myread,
    .write = mywrite
};

static int __init mod4_init(void) {
    base = proc_mkdir("proc4_folder_2", NULL);
    entry = proc_create("mod4_proc", 0666, base, &myops);
    return 0;
}

static void __exit mod4_exit(void) {
    proc_remove(entry);
    proc_remove(base);
}

module_init(mod4_init);
module_exit(mod4_exit);
```

【4】实验测试及效果截图

Module 1

使用 `make` 命令编译模块 C 源文件，生成编译好的内核模块目标文件 `mod1.ko` 。

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# make
make -C /lib/modules/5.4.0-53-generic/build M=/root/kernel/project1/mod1 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-53-generic'
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /root/kernel/project1/mod1/mod1.o
see include/linux/module.h for more information
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-53-generic'
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# ls
Makefile  mod1.ko   mod1.mod.c  mod1.o         Module.symvers
mod1.c    mod1.mod  mod1.mod.o  modules.order
```

使用 `insmod mod1.ko` 命令加载指定模块 `mod1.ko` ，使用 `lsmod` 命令查看当前正在运行的模块，下图说明已经成功加载。使用 `dmesg` 命令查看打印的消息，下图说明加载的过程中实现了打印。使用 `rmmod mod1` 卸载当前模块，同样打印了消息。

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# insmod mod1.ko
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# lsmod | grep mod1
mod1                   16384  0
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# dmesg | tail -1
[1214085.535656] Insert Module 1
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# rmmod mod1
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod1# dmesg | tail -1
[1214113.783017] Remove Module 1
```

Module 2

在加载模块时输入参数，即在 `insmod` 命令后带参数，即可将参数传入模块内。加载时将传入的参数打印出来如下图。

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod2# ls
Makefile  mod2.ko   mod2.mod.c  mod2.o         Module.symvers
mod2.c    mod2.mod  mod2.mod.o  modules.order
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod2# insmod mod2.ko a=666 b=hello c=
1,2,3,4
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod2# lsmod | grep mod2
mod2                   16384  0
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod2# dmesg | tail -2
[1244675.630722] Insert Module 2:
                 a = 666, b = hello, c = [1, 2, 3, 4]
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod2# rmmod mod2
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod2# dmesg | tail -2
                 a = 666, b = hello, c = [1, 2, 3, 4]
[1244711.857938] Remove Module 2
```

Module 3

加载模块后，可以发现 `/proc` 文件夹下新创建了一个 `mod3_proc` 文件。使用 `cat` 命令读这个文件，可以发现成功读入文件的内容。

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod3# ls
Makefile  mod3.ko   mod3.mod.c  mod3.o         Module.symvers
mod3.c    mod3.mod  mod3.mod.o  modules.order
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod3# insmod mod3.ko
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod3# lsmod | grep mod3
mod3                   16384  0
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod3# cat /proc/mod3_proc
mod3: hello world
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod3# rmmod mod3
```

Module 4

加载模块后，可以发现 `/proc` 文件夹下创建了 `proc4_folder_2` 文件夹，在该目录下创建了 `mod4_proc` 文件，使用 `cat` 命令读，起初文件中没有内容，所以读出的内容只有 `mod4:` 。使用 `echo` 命令通过管道将字符串 hello 写入该文件，发现成功修改了该文件（再次读出的内容为 `mod4: hello`）。

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# ls
Makefile  mod4.ko   mod4.mod.c  mod4.o         Module.symvers
mod4.c    mod4.mod  mod4.mod.o  modules.order
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# insmod mod4.ko
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# ls /proc
1       131810  255  641  86         diskstats    kpageflags      self
10      131818  283  645  87         dma          loadavg         slabinfo
102     14      3    669  88         driver       locks           softirqs
105     15      311  678  88571      execdomains  mdstat          stat
11      158     397  70   88762      fb           meminfo         swaps
118     16      4    709  89         filesystems  misc            sys
12      17      432  71   9          fs           modules         sysrq-trigger
129645  178     483  72   91         interrupts   mounts          sysvipc
129739  179     500  73   92         iomem        mtrr            thread-self
129792  18      501  74   acpi       ioports      net             timer_list
13      19      534  75   buddyinfo  irq          pagetypeinfo    tty
130180  2       535  76   bus        kallsyms     partitions      uptime
130190  20      538  77   cgroups    kcore        pressure        version
130192  21      571  78   cmdline    keys         proc4_folder    version_signature
130196  22      581  81   consoles   key-users    proc4_folder_2  vmallocinfo
130217  223     583  82   cpuinfo    kmsg         sched_debug     vmstat
130753  23      588  84   crypto     kpagecgroup  schedstat       zoneinfo
131307  24      6    85   devices    kpagecount   scsi
```

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# ls /proc/proc4_folder_2
mod4_proc
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# cat /proc/proc4_folder_2/mod4_p
roc
mod4:
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# echo hello > /proc/proc4_folder
_2/mod4_proc
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# cat /proc/proc4_folder_2/mod4_p
roc
mod4: hello
```

删除模块后，发现 `/proc` 下文件夹 `proc4_folder_2` 目录已经被删除。（p.s. 注意不是 `proc4_folder` 目录

```
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# rmmod mod4
root@iZuf63xs8u1971bor8zpc1Z:~/kernel/project1/mod4# ls /proc
1       131810  24   6    85          devices      kpagecount    self
10      131826  255  641  86          diskstats    kpageflags    slabinfo
102     131827  283  645  87          dma          loadavg       softirqs
105     14      3    669  88          driver       locks         stat
11      15      311  678  88571       execdomains  mdstat        swaps
118     158     397  70   88762       fb           meminfo       sys
12      16      4    709  89          filesystems  misc          sysrq-trigger
129645  17      432  71   9           fs           modules       sysvipc
129739  178     483  72   91          interrupts   mounts        thread-self
129792  179     500  73   92          iomem        mtrr          timer_list
13      18      501  74   acpi        ioports      net           tty
130180  19      534  75   buddyinfo   irq          pagetypeinfo  uptime
130190  2       535  76   bus         kallsyms     partitions    version
130192  20      538  77   cgroups     kcore        pressure      version_signature
130196  21      571  78   cmdline     keys         proc4_folder  vmallocinfo
130217  22      581  81   consoles    key-users    sched_debug   vmstat
130753  223     583  82   cpuinfo     kmsg         schedstat     zoneinfo
131307  23      588  84   crypto      kpagecgroup  scsi
```

【5】实验心得

这次实验基于 Linux 内核模块理论课，通过实践深入学习 Linux 内核模块编程基本方法。老师和学长并未提供编程中需要使用函数的具体用法，而是给我们适量的提示。这让我们有意识地去查看 Linux 内核源码中的一些 API，并 Google 许多资料，提高了我们自主编写和调试 C 程序的能力。在这次实验中，我对模块的特点加深了理解，比如模块是按序动态加载、卸载的，用于扩展内核功能的程序。我还进一步了解了 /proc 文件系统的模块编程和 Makefile 的基本编写方法。除了编写这次实验的 Makefile ，我还进一步学习了 Makefile 的设计思想，在今后很多项目都会频繁使用到 Makefile 。在编写模块 4 的代码中，我遇到了诸多 bug ，比如没有写 `MODULE_LICENSE` ，全局变量 `str` 没有分配空间（开个大数组解决）等等，在同学和 Google 的帮助下一步步解决，最终成功实现。总之，这次实验虽然相对简单，但让我收获了很多 Linux 内核模块编程和 Makefile 的知识！