Project 1: Linux 内核模块

Chentao Wu吴晨涛
Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

课程目标

- 使用Vmware Workstation Pro创建linux ubuntu虚拟机
- 学习如何创建内核模块并加载到Linux内核中
- 学习使用/proc文件系统来访问内核和进程统计信息

创建虚拟机

- 可参考课程群中助教发的虚拟机安装教程

内核模块总览

- 5-9行 simple_init 为模块入口点，模块加载进内核时调用。模块入口点函数必须返回整数值int，0表示成功，其他任何值表示失败

- 11-14行 simple_exit 为模块出口点，模块从内核中移除时调用。模块出口点函数返回void。模块入口点和模块出口点都不传入任何参数

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
/* This function is called when the module is loaded. */
int simple_init(void)
{
  printk(KERN_INFO "Loading Kernel Module\n");
  return 0;
}
/* This function is called when the module is removed. */
void simple_exit(void)
{
  printk(KERN_INFO "Removing Kernel Module\n");
}
/* Macros for registering module entry and exit points.
module_init(simple_init);
module_exit(simple_exit);
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("SGG");
```

内核模块总览

- 16-17行使用 module_init 和 module_exit 两个宏分别注册模块入口和出口点
- 内核态打印使用 printk 函数，用法相当于 printf。printk 的输出将发送到内核日志缓冲区，可使用 dmesg 命令读取

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
/* This function is called when the module is loaded. */
int simple_init(void)
{
  printk(KERN_INFO "Loading Kernel Module\n");
  return 0;
}
/* This function is called when the module is removed. */
void simple_exit(void)
{
  printk(KERN_INFO "Removing Kernel Module\n");
}
/* Macros for registering module entry and exit points.
module_init(simple_init);
module_exit(simple_exit);
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("SGG");
```

内核模块编译

- 我们使用 Makefile 编译源码，具体的 Makefile 以及simple.c 可以从课程发布的项目源码中获得（已上传至 canvas ，解压 final-src-osc10e.zip 即可获得）。

- 在 Makefile 所在目录下，调用以下命令

  make

  将编译生成多个文件。文件 simple.ko 为已编译的内核模块。

- 有兴趣了解Makefile的同学可以参考

  http://ruanyifeng.com/blog/2015/02/make.html

内核模块加载与卸载

- 我们使用 insmod 来加载内核模块，命令如下：

  sudo insmod simple.ko

  为了检查模块是否加载成功，可以调用 lsmod命令并查找是否有模块 simple。另外，由于前面实现的模块入口点函数 simple_init 调用了printk 打印，我们可以调用如下指令查看内核日志缓冲区:

  dmesg

  如果加载成功，你将看到”Loading Module.”。

- 我们使用rmmod来卸载内核模块，命令如下：

  sudo rmmod simple

  我们同样可以用 dmesg 查看内核日志缓冲区，来判断内核模块是否成功卸载。

/proc 文件系统

- /proc 文件系统是一个“伪”文件系统，只存在于内核内存中，主要用于查询各种内核和进程统计信息。

- 在24行的模块入口点 proc_init 中，我们使用 proc_create 创建一个 /proc/hello 入口。proc_create 传入了proc_ops，这是一个 file_operations 结构体，初始化了 .owner 和 .read 两个成员。其中 ./read 赋值成了 proc_read，这意味着当读取 /proc/read 时，就会调用 proc_read 函数。

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>

#define BUFFER_SIZE 128

#define PROC_NAME "hello"
#define MESSAGE "Hello World\n"

/**
 * Function prototypes
 */
ssize_t proc_read(struct file *file, char *buf, size_t count,
loff_t *pos);
static struct file_operations proc_ops = {
        .owner = THIS_MODULE,
        .read = proc_read,
};


/* This function is called when the module is loaded. */
int proc_init(void)
{

        // creates the /proc/hello entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC_NAME, 0, NULL, &proc_ops);

        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);

    return 0;
}

/* This function is called when the module is removed. */
void proc_exit(void) {

        // removes the /proc/hello entry
        remove_proc_entry(PROC_NAME, NULL);

        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}
```

/proc 文件系统

- 在 proc_read 中，我们可以看到字符串 “Hello World\n” 被写入到缓存区 buffer 中，由于 buffer 是 proc_read 的局部变量，而 proc_read 是在内核中调用的，因此buffer属于内核内存。为了让其可以从用户空间访问，需要使用内核函数 copy_to_user 将缓存区数据拷贝到用户空间。

```c
ssize_t proc_read(struct file *file, char __user *usr_buf,
size_t count, loff_t *pos)
{
        int rv = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;

        if (completed) {
                completed = 0;
                return 0;
        }

        completed = 1;

        rv = sprintf(buffer, "Hello World\n");

        // copies the contents of buffer to userspace usr_buf
        copy_to_user(usr_buf, buffer, rv);

        return rv;
}
```

/proc 文件系统

- 每次读取 /proc/hello 文件时，都会重复调用proc_read，直到它返回0，因此必须有逻辑确保该函数在收集数据（在本例中为“Hello World\n”）后返回0，这些数据将进入相应的 /proc/hello 文件。

- 我们可以使用 cat 命令访问该文件，命令如下：

  cat /proc/hello

```
1  ssize_t proc_read(struct file *file, char __user *usr_buf,
   size_t count, loff_t *pos)
2  {
3          int rv = 0;
4          char buffer[BUFFER_SIZE];
5          static int completed = 0;
6
7          if (completed) {
8                  completed = 0;
9                  return 0;
10         }
11
12         completed = 1;
13
14         rv = sprintf(buffer, "Hello World\n");
15
16         // copies the contents of buffer to userspace usr_buf
17         copy_to_user(usr_buf, buffer, rv);
18
19         return rv;
20 }
```

作业及评分

自行阅读课本第二章的 Programming Projects 部分，并完成以下四个任务，完成后共计11分。

- （课本习题 4分）设计一个内核模块：创建一个名为 /proc/jiffies 的 /proc 文件。在读取 /proc/jiffies 文件时报告 jiffies 的当前值。结果可使用 cat 获取，命令如下：

  cat /proc/jiffies

  确保在删除模块时删除 /proc/jiffies。
- （课本习题 4分）设计一个内核模块：创建一个名为 /proc/seconds 的 /proc 文件。在读取 /proc/seconds 文件时报告内核模块加载后经过的秒数，这将涉及到 jiffies 值以及 HZ 频率。结果可使用cat获取，命令如下：

  cat /proc/seconds

  确保在删除模块时删除 /proc/seconds。
- （报告 2分）做一个简单的报告解释你的代码，报告要求不能超过2页（防内卷）。
- （Bonus 1分）在 proc_read 函数中使用 copy_to_user 将内核 buffer 数据拷贝到用户态 buffer。而在用户态程序中，内存拷贝通常使用 memcpy 。请查阅相关资料，简单讨论二者的差异。