

Project 1 -- Introduction to Linux Kernel Modules

陈思远 518021910567

Project 1 -- Introduction to Linux Kernel Modules

- 1. 实验内容和目标
 - 2. 升级内核
 - 2.1 实现步骤
 - 3. 加载/卸载内核模块
 - 3.1 要求
 - 3.2 实现
 - 3.3 结果
 - 4. Assignment
 - 4.1 要求
 - 4.2 实现
 - 4.2.1 jiffies模块
 - 4.2.2 seconds模块
 - 4.3 结果
- 总结与反思

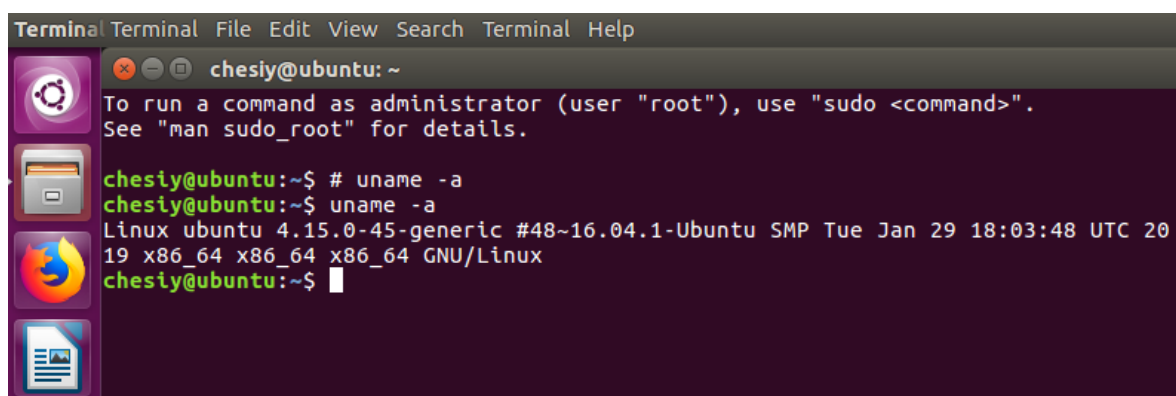
1. 实验内容和目标

1. 学习Linux终端命令行的基本使用，完成内核升级。
2. 学习创建内核模块，并将其载入Linux内核。
3. 修改内核模块，使之能在 `/proc` 文件系统中创建一个条目，并可以在终端打印一些内容。

2. 升级内核

2.1 实现步骤

1. 安装VMware虚拟机和Ubuntu操作系统，版本为16.04。（ps：可以把虚拟机的硬盘大小设置得大一些，不然后续编译内核的过程中可能出现空间不足的情况）
2. 在terminal中输入 `uname -a` 查看当前的内核版本为4.15.0，运行结果如下。

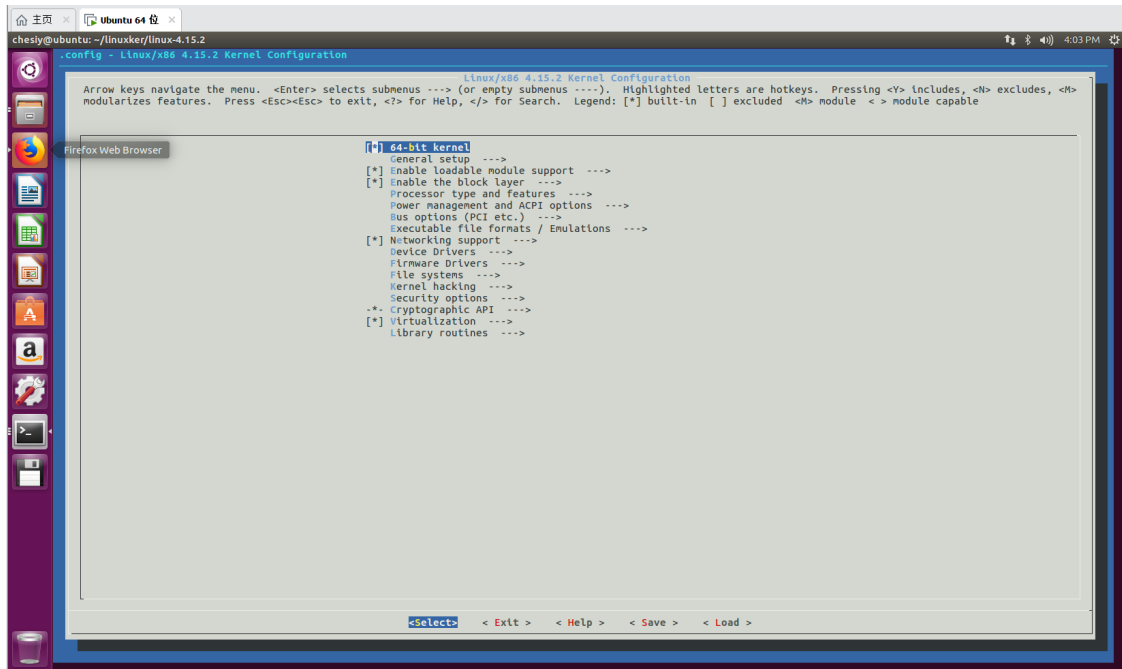


```
Terminal Terminal File Edit View Search Terminal Help
chesiy@ubuntu: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
chesiy@ubuntu:~$ # uname -a
chesiy@ubuntu:~$ uname -a
Linux ubuntu 4.15.0-45-generic #48~16.04.1-Ubuntu SMP Tue Jan 29 18:03:48 UTC 20
19 x86_64 x86_64 x86_64 GNU/Linux
chesiy@ubuntu:~$
```

3. 从<https://www.kernel.org/>上下载更高版本的内核文件，并解压。我选取了4.15.2的内核。
4. 为了编译内核，还需要配置好环境，安装一些软件。采用如下命令完成。

```
sudo apt-get install fakeroot build-essential ncurses-dev
sudo apt-get install xz-utils libssl-dev bc flex libelf-dev bison
```

- 配置需要包含的模块。输入命令 `make menuconfig`，该命令会打开一个如下图所示的配置工具，从中选择启用或禁用某些可删减的模块。（ps：这一步可能会出现 `your display is too small to run Menuconfig!` 的报错，只需把窗口放大即可解决）



- 执行 `make` 命令编译内核。如果是多核处理器，可以采用 `make -j4` 将编译拆成4个线程执行，可以显著提高编译速度。由于是虚拟机，我的编译过程持续了3小时左右。
- 执行 `make module_install` 安装内核模块。
- 执行 `make install` 安装新内核。
- 执行如下命令启用内核作为引导。4.15.2为编译完的内核版本号。

```
sudo update-initramfs -c -k 4.15.2
```

- 执行 `sudo update-grub` 命令更新内核版本号。
- 再次输入 `uname -a` 查看当前的内核版本，为4.15.2，说明内核升级成功。

```
chesity@ubuntu: ~  
chesity@ubuntu:~$ uname -a  
Linux ubuntu 4.15.2 #1 SMP Wed Mar 18 20:41:27 CST 2020 x86_64 x86_64 x86_64 GNU  
/Linux  
chesity@ubuntu:~$
```

3. 加载/卸载内核模块

3.1 要求

- 在 `simple_init()` 函数中打印常数 `GOLDEN_RATIO_PRIME`，`jiffies` 和 `HZ`。
- 在 `simple_exit()` 函数中打印3300和24的最大公因数，以及常数 `jiffies`。

3.2 实现

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

```

#include <asm/param.h>
#include <linux/jiffies.h>

/* This function is called when the module is loaded. */
static int simple_init(void)
{
    printk(KERN_INFO "Loading Module\n");
    //打印GOLDEN_RATIO_PRIME
    printk(KERN_INFO "GOLDEN_RATIO_PRIME= %lu\n", GOLDEN_RATIO_PRIME);
    //打印jiffies
    printk(KERN_INFO "init_jiffies= %lu\n", jiffies);
    printk(KERN_INFO "HZ= %lu\n", HZ);

    return 0;
}

/* This function is called when the module is removed. */
static void simple_exit(void) {
    printk(KERN_INFO "Removing Module\n");
    unsigned long res=gcd(3300,24);
    //打印3300和24的最大公因数
    printk(KERN_INFO "gcd_res= %lu\n", res);
    //打印jiffies
    printk(KERN_INFO "exit_jiffies= %lu\n", jiffies);
}

/* 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");

```

3.3 结果

1. 用 make 指令编译simple.c, 生成simple.ko文件。
2. 在terminal中输入 `sudo dmesg -c` 清空kernel log buffer。
3. 输入 `sudo insmod simple.ko` 加载内核模块。
4. 输入 `dmesg` 打印kernel log, 可以发现输出了常数GOLDEN_RATIO_PRIME、jiffies和HZ的值, simple_init()函数已被执行。
5. 输入 `sudo rmmod simple` 卸载内核模块。
6. 输入 `dmesg` 打印kernel log, 可以发现输出了3300和24的最大公因数、以及当时jiffies的值, simple_exit()函数被执行。

```

chesiy@ubuntu:~/osProject/Proj1$ sudo insmod simple.ko
chesiy@ubuntu:~/osProject/Proj1$ dmesg
[ 2343.369602] Loading Module
[ 2343.369604] GOLDEN_RATIO_PRIME= 7046029254386353131
[ 2343.369605] init_jiffies= 4295478052
[ 2343.369605] HZ= 250
chesiy@ubuntu:~/osProject/Proj1$ sudo rmmod simple
chesiy@ubuntu:~/osProject/Proj1$ dmesg
[ 2343.369602] Loading Module
[ 2343.369604] GOLDEN_RATIO_PRIME= 7046029254386353131
[ 2343.369605] init_jiffies= 4295478052
[ 2343.369605] HZ= 250
[ 2363.099353] Removing Module
[ 2363.099354] gcd_res= 12
[ 2363.099355] exit_jiffies= 4295482985

```

4. Assignment

4.1 要求

1. 设计一个可以创建/proc文件/proc/jiffies的内核模块，用于打印/proc/jiffies文件被读入时的的jiffies值。
2. 设计一个可以创建/proc文件/proc/seconds的内核模块，用于打印从该内核模块被加载起过去的时间。

4.2 实现

4.2.1 jiffies模块

该模块的实现较为简单，只需根据书本给的hello模块修改，在proc_read()函数中加入输出jiffies的语句即可。

```

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

#define BUFFER_SIZE 128
#define PROC_NAME "jiffies"

static 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. */
static int proc_init(void)
{
    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. */
static void proc_exit(void) {
    remove_proc_entry(PROC_NAME, NULL);
    printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}
/**
 * This function is called each time the /proc/jiffies is read.
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 */
static 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, "%lu\n", jiffies);
    // copies the contents of buffer to userspace usr_buf
    copy_to_user(usr_buf, buffer, rv);
    return rv;
}

/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("jiffies Module");
MODULE_AUTHOR("SGG");

```

4.2.2 seconds模块

该模块的实现需要定义一个全局变量，记录模块刚被加载时的jiffies，在模块初始化时赋值。此外，由于jiffies表示计时器中断产生的次数，HZ表示计时器的频率，它们与过去的时间second的关系为

$$second = \frac{jiffies - init_jiffies}{HZ}$$

具体代码如下。

```

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/uaccess.h>
#include <linux/jiffies.h>
#include <asm/param.h>

#define BUFFER_SIZE 128
#define PROC_NAME "seconds"

```

```

unsigned long init_jiffies; //定义一个全局变量，用于记录模块刚被加载时的jiffies

static 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. */
static int proc_init(void)
{
    proc_create(PROC_NAME, 0, NULL, &proc_ops);
    init_jiffies=jiffies; //模块刚被加载时的jiffies
    printk(KERN_INFO "/proc/%s created\n", PROC_NAME);
    return 0;
}

/* This function is called when the module is removed. */
static void proc_exit(void) {
    remove_proc_entry(PROC_NAME, NULL);
    printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}

/**
 * This function is called each time the /proc/seconds is read.
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 */
static 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;
    unsigned long sec=(jiffies-init_jiffies)/HZ;//计算从模块刚加载过去的秒数
    rv = sprintf(buffer, "The number of seconds=%lu\n",sec);

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

    return rv;
}

/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");

```

```
MODULE_DESCRIPTION("Seconds Module");  
MODULE_AUTHOR("SGG");
```

4.3 结果

1. 输入 `sudo insmod jiffies.ko` 加载内核模块jiffies。
2. 输入 `dmesg` 打印kernel log, 可以发现/proc/jiffies已被成功创建。
3. 输入 `cat /proc/jiffies` 读入/proc/jiffies文件, 可以发现输出了常数jiffies。jiffies表示计时器中断产生的次数, 随时间流逝而增大。多次输出jiffies发现其单调递增, 说明是正确的。
4. 输入 `sudo rmmod jiffies` 卸载内核模块jiffies。

```
chesiy@ubuntu:~/osProject/Proj1/jiffies$ sudo insmod jiffies.ko  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ dmesg  
[ 6914.703017] /proc/jiffies created  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ /cat /proc/jiffies  
bash: /cat: No such file or directory  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ cat /proc/jiffies  
4296631759  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ cat /proc/jiffies  
4296633153  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ cat /proc/jiffies  
4296634287  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ sudo rmmod jiffies  
chesiy@ubuntu:~/osProject/Proj1/jiffies$ dmesg  
[ 6914.703017] /proc/jiffies created  
[ 6992.858782] /proc/jiffies removed
```

1. 输入 `sudo insmod seconds.ko` 加载内核模块seconds。
2. 输入 `dmesg` 打印kernel log, 可以发现/proc/seconds已被成功创建。
3. 输入 `cat /proc/seconds` 读入/proc/seconds文件, 可以发现输出了常数从模块被加载到输出时过去的秒数。
4. 输入 `sudo rmmod seconds` 卸载内核模块seconds。

```
chesiy@ubuntu:~/osProject/Proj1/seconds$ sudo insmod seconds.ko  
chesiy@ubuntu:~/osProject/Proj1/seconds$ dmesg  
System Settings /proc/seconds created  
chesiy@ubuntu:~/osProject/Proj1/seconds$ cat /proc/seconds  
The number of seconds=30  
chesiy@ubuntu:~/osProject/Proj1/seconds$ cat /proc/seconds  
The number of seconds=40  
chesiy@ubuntu:~/osProject/Proj1/seconds$ cat /proc/seconds  
The number of seconds=53  
chesiy@ubuntu:~/osProject/Proj1/seconds$ cat /proc/seconds  
The number of seconds=64  
chesiy@ubuntu:~/osProject/Proj1/seconds$ sudo rmmod second  
rmmod: ERROR: Module second is not currently loaded  
chesiy@ubuntu:~/osProject/Proj1/seconds$ sudo rmmod seconds  
chesiy@ubuntu:~/osProject/Proj1/seconds$ dmesg  
[ 7884.524534] /proc/seconds created  
[ 7981.079231] /proc/seconds removed
```

总结与反思

在这个project中, 我实现了内核的升级、内核模块代码的编写, 以及内核模块的加载和卸载。通过实践, 我对terminal的使用更加熟练, 也学习了如何编写makefile, 更体会到了亲手写一个内核模块并加载、卸载它的过程, 使原本神秘的内核变得更加真实可感。在写project的过程中, 我也遇到了不少问题, 比如直接复制书本上的hello.c代码编译会有如下编译错误。

