

Report for OS Project 1

Introduction to Linux Kernel Modules

519021910071 陈文迪

I. 实验任务

1. 从源码编译Linux内核，并装载到当前的发行版中。
2. 了解如何创建内核模块，并加载进Linux内核中。
3. 了解 `/proc` 文件系统的基本结构，内核模块编程的基本概念以及如何加载和卸载内核模块。
4. 了解一些基本的内核变量或常量，类似于 `jiffies`、`HZ`。
5. 编写一个名为 `jiffies` 的内核模块，在其加载入内核后，可以通过访问 `/proc` 文件系统访问目前 `jiffies` 的值。
6. 编写一个名为 `seconds` 的内核模块，其功能为可以通过访问 `/proc` 文件系统来得到其被加载入内核后经过的时间。

II. 实验思路

1. Linux内核与内核模块

Linux内核是整个Linux操作系统的核心，其不包括任何应用程序，但包含了如进程调度，内存管理等操作系统服务。我们现在所使用的Linux发行版事实上是由Linux内核加上一些便于用户使用的应用程序软件与系统管理软件等所组成的。通过对内核源码的编译，我们可以将当前发行版的内核更换成指定的版本。

由于Linux采用了模块化设计的思想，程序员可以自行设计内核模块来实现与内核的交互，并且可以使用内核函数来设计程序。当然我们需要非常仔细地编写代码，**避免可能的错误**，因为内核代码的错误很可能导致整个系统的崩溃。在本次课程设计中，我们使用虚拟机开发环境来尽可能避免内核崩溃导致的损失。

2. `jiffies`与`HZ`内核变量

`jiffies` 内核变量记录了从系统启动以来共发生了多少次计数器中断，而 `HZ` 变量记录了每秒钟发生计时器中断的个数。因此当我们需要载入后经过时间 `seconds` 时可以通过如下公式计算：

$$seconds = \frac{jiffies_{now} - jiffies_{init}}{HZ}$$

3. 编写内核模块时的细节

任何内核模块都有两个重要函数。

```
int proc_init(void);    \\当模块被转载时自动执行
void proc_exit(void);   \\当模块被卸载时自动执行
```

同时，在本项目中还有一个重要函数。

```
ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *pos);    \\访问proc文件系统中相应条目时调用
```

利用上述这几个函数我们可以非常轻松地实现实验任务中的几个功能。

同时我们也需要注意几个细节。

其一，在内核中，输入输出函数具有不同的形式，`printf` 函数要替换为 `printk`。

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

其二，我们在内核模块中创建的buffer位于内核内存中，我们需要使用 `copy_to_user` 函数将其复制到用户内存中。

```
rv = sprintf(buffer, "%lu\n",jiffies);
```

4. 项目中所需要的一些shell命令

在本次项目中，代码的编写可以用任意编译器，但编译、加载等操作需要需要使用shell完成。

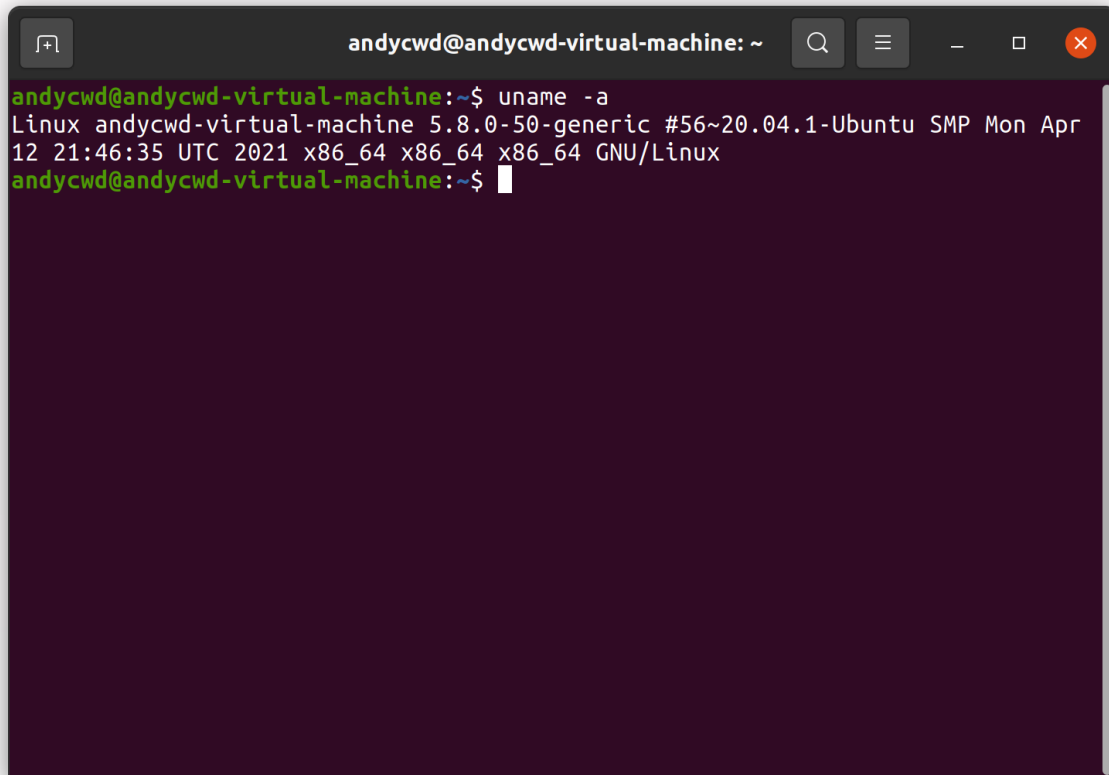
命令	功能
cd	切换用户当前工作目录
ls	显示目录内容列表
cat	显示文件内容
sudo	使用root身份执行命令
apt install	Debian Linux中使用APT软件包管理器安装新包
apt update	更新软件包列表
apt upgrade	更新所有已安装的包
lsmod	显示已载入系统的模块
insmod	将给定模块加载到内核中
rmmod	从运行内核中移除指定内核模块
dmesg	显示内核环形缓冲区的内容
dmesg -c	显示信息后，清楚环形缓冲区的内容
uname -a	按顺序打印全部的机器和操作系统信息
make menuconfig	图形化界面配置
make -j[n]	多线程（n个）编译内核
make modules_install	编译内核模块
make install	安装内核模块
xz -d	解压缩
tar -xf	解包

注意：本项目使用Ubuntu20.04开发。由于网络访问条件的限制，建议将软件源切换到清华源或交大源。

III. 实验过程

1. Linux 内核编译

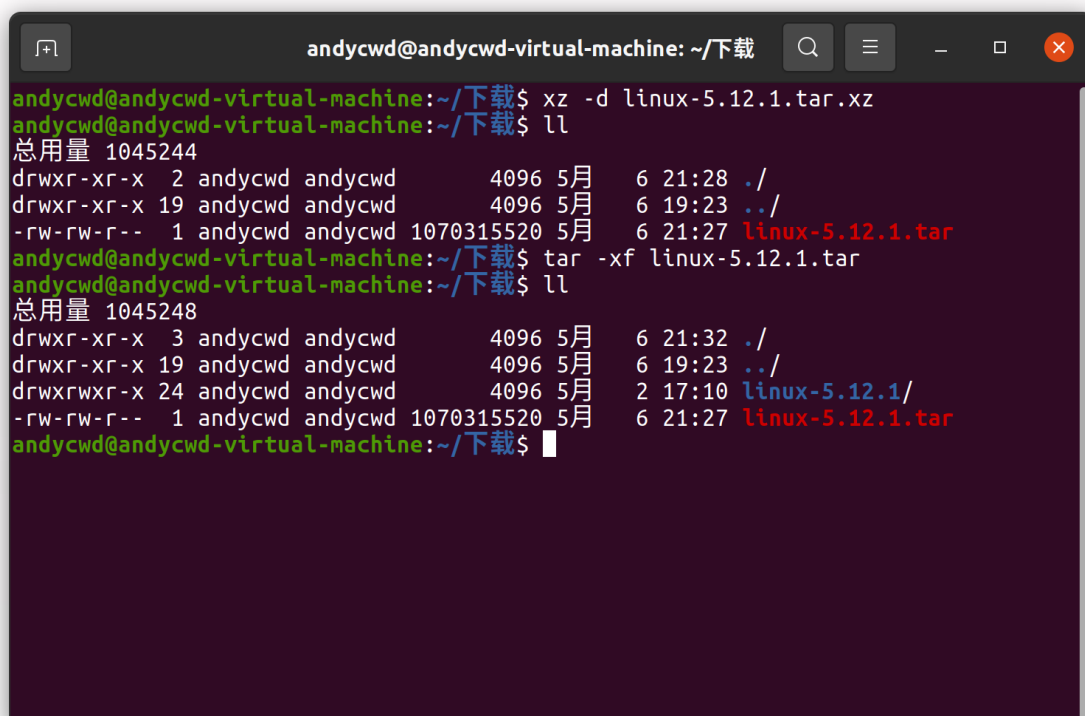
我们先用 `uname-a` 查看当前的内核版本。



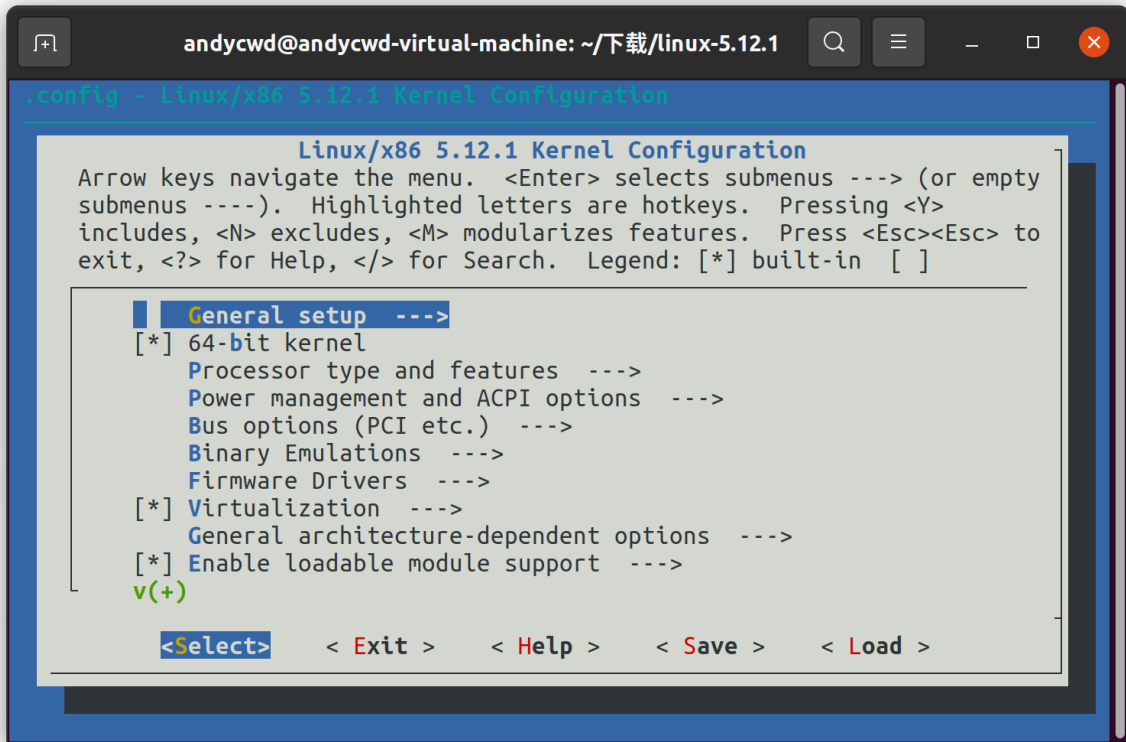
```
andycwd@andycwd-virtual-machine: ~  
andycwd@andycwd-virtual-machine:~$ uname -a  
Linux andycwd-virtual-machine 5.8.0-50-generic #56~20.04.1-Ubuntu SMP Mon Apr  
12 21:46:35 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux  
andycwd@andycwd-virtual-machine:~$
```

可以看到当前的Linux 内核版本为5.8.0。

接着我们前往kernel.org下载最新的内核版本，使用 `xz -d` 解压缩并使用 `tar -xf` 解包后，利用 `make menuconfig` 图形化界面编辑编译配置并保存。

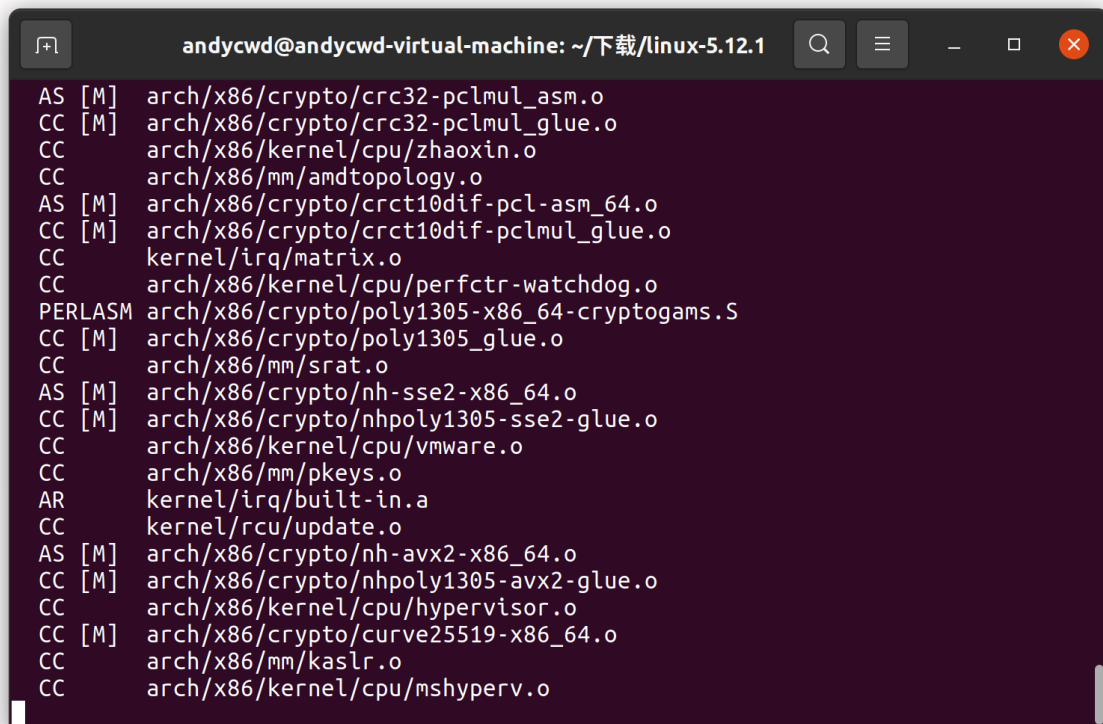


```
andycwd@andycwd-virtual-machine: ~/下载  
andycwd@andycwd-virtual-machine:~/下载$ xz -d linux-5.12.1.tar.xz  
andycwd@andycwd-virtual-machine:~/下载$ ll  
总用量 1045244  
drwxr-xr-x  2 andycwd andycwd      4096 5月  6 21:28 ./  
drwxr-xr-x 19 andycwd andycwd      4096 5月  6 19:23 ../  
-rw-rw-r--  1 andycwd andycwd 1070315520 5月  6 21:27 linux-5.12.1.tar  
andycwd@andycwd-virtual-machine:~/下载$ tar -xf linux-5.12.1.tar  
andycwd@andycwd-virtual-machine:~/下载$ ll  
总用量 1045248  
drwxr-xr-x  3 andycwd andycwd      4096 5月  6 21:32 ./  
drwxr-xr-x 19 andycwd andycwd      4096 5月  6 19:23 ../  
drwxrwxr-x 24 andycwd andycwd      4096 5月  2 17:10 linux-5.12.1/  
-rw-rw-r--  1 andycwd andycwd 1070315520 5月  6 21:27 linux-5.12.1.tar  
andycwd@andycwd-virtual-machine:~/下载$
```



由于本次编译采用默认的配置，因此在此处我们不对编译配置进行额外的修改。

接着，我们使用 `make -j4` 来编译内核。



然后，使用 `sudo make modules_install` 对内核模块进行编译并安装。

```
andycwd@andycwd-virtual-machine: ~/下载/linux-5.12.1
andycwd@andycwd-virtual-machine: ~/... x andycwd@andycwd-virtual-machine: ~/... x
INSTALL crypto/essiv.ko
INSTALL crypto/fcrypt.ko
INSTALL crypto/keywrap.ko
INSTALL crypto/khazad.ko
INSTALL crypto/lrw.ko
INSTALL crypto/lz4.ko
INSTALL crypto/lz4hc.ko
INSTALL crypto/md4.ko
INSTALL crypto/michael_mic.ko
INSTALL crypto/nhpoly1305.ko
INSTALL crypto/ofb.ko
INSTALL crypto/pcbc.ko
INSTALL crypto/pcrypt.ko
INSTALL crypto/poly1305_generic.ko
INSTALL crypto/rmd160.ko
INSTALL crypto/seed.ko
INSTALL crypto/serpent_generic.ko
INSTALL crypto/sha3_generic.ko
INSTALL crypto/sm3_generic.ko
INSTALL crypto/sm4_generic.ko
INSTALL crypto/streebog_generic.ko
INSTALL crypto/tcrypt.ko
INSTALL crypto/tea.ko
INSTALL crypto/twofish_common.ko
```

接着，我们使用 `sudo make install` 安装新内核。

```
andycwd@andycwd-virtual-machine: ~/下载/linux-5.12.1
andycwd@andycwd-virtual-machine:~/下载/linux-5.12.1$ sudo make install
sh ./arch/x86/boot/install.sh 5.12.1 arch/x86/boot/bzImage \
    System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.12.1 /boot/vmlinuz-5.12.1
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.12.1 /boot/vmlinuz-5.12.1
update-initramfs: Generating /boot/initrd.img-5.12.1
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.12.1 /boot/vmlinuz-5.12.1
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.12.1 /boot/vmlinuz-5.12.1
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.12.1 /boot/vmlinuz-5.12.1
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
正在生成 grub 配置文件 ...
找到 Linux 镜像: /boot/vmlinuz-5.12.1
找到 initrd 镜像: /boot/initrd.img-5.12.1
找到 Linux 镜像: /boot/vmlinuz-5.8.0-50-generic
找到 initrd 镜像: /boot/initrd.img-5.8.0-50-generic
找到 Linux 镜像: /boot/vmlinuz-5.8.0-43-generic
找到 initrd 镜像: /boot/initrd.img-5.8.0-43-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
完成
andycwd@andycwd-virtual-machine:~/下载/linux-5.12.1$
```

重启系统后，在 grub 界面选择我们新编译好的内核版本启动。启动后，我们再次使用 `uname-a` 查看当前的内核版本，可以看到我们的内核版本已经切换到 5.12.1。

```
andycwd@andycwd-virtual-machine: ~  
andycwd@andycwd-virtual-machine:~$ uname -a  
Linux andycwd-virtual-machine 5.12.1 #1 SMP Fri May 7 09:06:04 CST 2021 x86_64 x  
86_64 x86_64 GNU/Linux  
andycwd@andycwd-virtual-machine:~$
```

注意：在上述编译过程中，若提示有某个依赖包没有安装，可以通过 `sudo apt install` 命令安装所需的依赖包。

2. 编写jiffies内核模块

书本附带的代码已经给出了项目的基本框架，我们只需要修改 `proc_read` 函数，使得每次对 `/proc/jiffies` 访问时可以将此时的 `jiffies` 打印出来。

```
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);  
    copy_to_user(usr_buf, buffer, rv);  
  
    return rv;  
}
```

接着，我们编译后使用 `insmod` 命令加载模块，使用 `cat` 命令访问 `/proc` 文件系统，最后使用 `rmmod` 卸载模块。每次转载、卸载，可以用 `dmesg -c` 查看环形缓冲区的内容。

以下是运行结果：

```
andycwd@andycwd-virtual-machine: ~/桌面/CS307-Operatin...
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo insmod jiffies.ko
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo dmesg -c
[ 1477.287862] /proc/jiffies created
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
cat /proc/jiffies
4295266182
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo rmmod jiffies
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo dmesg -c
[ 1502.828397] /proc/jiffies removed
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
```

3. 编写seconds内核模块

在本项目中，不仅需要修改 `proc_read` 函数，还需要修改 `proc_init` 函数来记录载入时的 `jiffies` 的值。这样通过利用我们在上文中导出的公式就可以得出模块加载后经过的时间。

```
unsigned long int first_loaded;

int proc_init(void)
{
    proc_create(PROC_NAME, 0, NULL, &my_proc_ops);
    printk(KERN_INFO "/proc/%s created\n", PROC_NAME);
    first_loaded = jiffies;
    return 0;
}

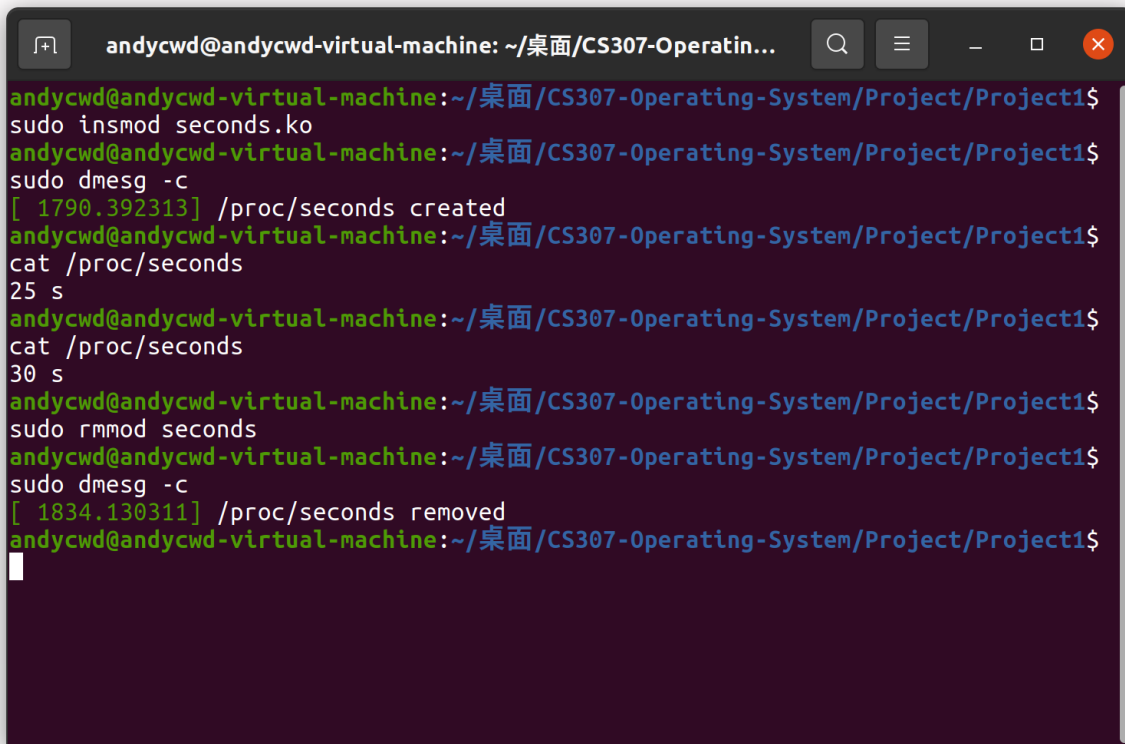
ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t
*pos)
{
    int rv = 0;
    char buffer[BUFFER_SIZE];
    static int completed = 0;

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

    completed = 1;
    rv = sprintf(buffer, "%lu s\n", (jiffies-first_loaded)/HZ);
    copy_to_user(usr_buf, buffer, rv);

    return rv;
}
```

以下是运行结果：



```
andycwd@andycwd-virtual-machine: ~/桌面/CS307-Operatin...
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo insmod seconds.ko
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo dmesg -c
[ 1790.392313] /proc/seconds created
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
cat /proc/seconds
25 s
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
cat /proc/seconds
30 s
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo rmmod seconds
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
sudo dmesg -c
[ 1834.130311] /proc/seconds removed
andycwd@andycwd-virtual-machine:~/桌面/CS307-Operating-System/Project/Project1$
```

4. 编写Makefile

在原Makefile中添加我们所编写模块的目标文件。

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

IV. 遇到的问题

1. 在编译内核的过程中曾遇到如下错误


```
andycwd@andycwd-virtual-machine: ~/下载/linux-5.12.1
CC [M] arch/x86/kvm/svm/svm.o
CC kernel/panic.o
CC kernel/cpu.o
AS [M] arch/x86/kvm/svm/vmenter.o
CC [M] arch/x86/kvm/svm/pmu.o
CC [M] arch/x86/kvm/svm/nested.o
CC [M] arch/x86/kvm/svm/avic.o
CC [M] arch/x86/kvm/svm/sev.o
CC kernel/exit.o
LD [M] arch/x86/kvm/kvm.o
make[1]: *** 没有规则可制作目标“debian/canonical-certs.pem”，由“certs/x509_certificate_list”需求。 停止。
make[1]: *** 正在等待未完成的任务....
CC certs/system_keyring.o
make: *** [Makefile:1851: certs] 错误 2
make: *** 正在等待未完成的任务....
LD [M] arch/x86/kvm/kvm-intel.o
CC kernel/softirq.o
CC kernel/resource.o
LD [M] arch/x86/kvm/kvm-amd.o
CC kernel/sysctl.o
CC kernel/capability.o
CC kernel/ptrace.o
CC kernel/user.o
```

在修改 `.config` 编译配置文件中 `CONFIG_SYSTEM_TRUSTED_KEY` 一行后修复。

2. 用make编译模块时报错

在首次编译时，编译器报告如下错误：向 `proc_create` 函数传递的参数错误。通过查阅相关资料发现，在5.12.1内核版本中，`proc_create` 函数的第四个参数应该为 `const struct proc_ops*`。因此我们需要传递对应的结构体参数，通过查阅内核源代码，我们获得了该结构体的具体结构。依照此结构构造结构体对象并正确传参后便可以正确编译了。