Project 1: Introduction to Linux Kernel Modules

CS307-Operating System (D), CS356-Operating System Course Design, Chentao Wu, Spring 2020.

Name: 方泓杰(Hongjie Fang) Student ID: 518030910150 Email: galaxies@sjtu.edu.cn

1 Compiling Linux Kernel

In this section, we will upgrade the Linux Kernel, and we will compile Linux Kernel during the process. Here are more detailed information.

1.1 Preparation

We download and install the VMWare Workstation Pro 15.5.0 software, which is distinguished for its virtual machine technology. Then we download the image file of the Ubuntu operating system ubuntu-18.04.4-desktop-amd64.iso, and install it as a virtual machine in VMWare Workstation.

After installation, we open the terminal of the virtual machine and input the following instruction, and we can see the current Linux Kernel version of the Ubuntu system.

```
1 uname -a
```

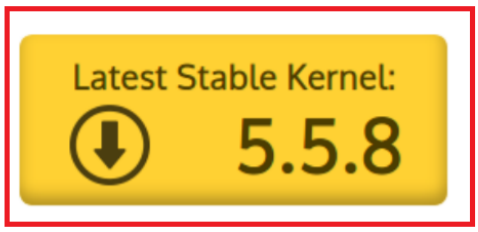
The current Linux Kernel version of my Ubuntu is 5.3.0-40-generic (Fig. 1).

```
galaxies@ubuntu:~$ uname -a
Linux ubuntu 5.3.0-40-generic #32~18.04.1-Ubuntu SMP Mon Feb 3 14:05:59 UTC 2020
 x86_64 x86_64 x86_64 GNU/Linux
```

Figure 1: The current Linux Kernel version

Then we visit www.kernel.org to download the high-version kernel source. I choose the latest stable kernel version 5.5.8 to download (Fig. 2).

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/



mainline:	5.6-rc5	2020-03-09	[tarball] [patch] [inc. patch] [view diff] [browse]
stable:	5.5.8	2020-03-05	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]

Figure 2: The latest stable kernel version

Then we use the following instruction to unzip the Linux Kernel source file to the directory `/usr/src`, then we can find the corresponding files under that directory (Fig. 3).

```
1 tar xvJf linux-5.5.8.tar.xz -C /usr/src
```

```
galaxies@ubuntu:/usr/src$ ls
linux-5.5.8                      linux-headers-5.3.0-40
linux-headers-5.3.0-28           linux-headers-5.3.0-40-generic
linux-headers-5.3.0-28-generic
```

Figure 3: The files after unzipping

We use the following instructions to install and upgrade necessary programs before compiling.

```
1 sudo apt update
2 sudo apt upgrade
3 sudo apt-get install git fakeroot build-essential ncurses-dev
4 sudo apt-get install xz-utils libssl-dev bc flex libelf-dev bison
```

Then, we enter the following instruction to set some configurations of kernel (Fig. 4).

```
1 sudo make menuconfig
```

```
galaxies@ubuntu:/usr/src/linux-5.5.8$ sudo make menuconfig
  HOSTCC  scripts/basic/fixdep
  UPD     scripts/kconfig/mconf-cfg
  HOSTCC  scripts/kconfig/mconf.o
  HOSTCC  scripts/kconfig/lxdialog/checklist.o
  HOSTCC  scripts/kconfig/lxdialog/inputbox.o
  HOSTCC  scripts/kconfig/lxdialog/menubox.o
  HOSTCC  scripts/kconfig/lxdialog/textbox.o
  HOSTCC  scripts/kconfig/lxdialog/util.o
```

Figure 4: Set configurations of kernel

The configuration menu is displayed as follows (Fig. 5).

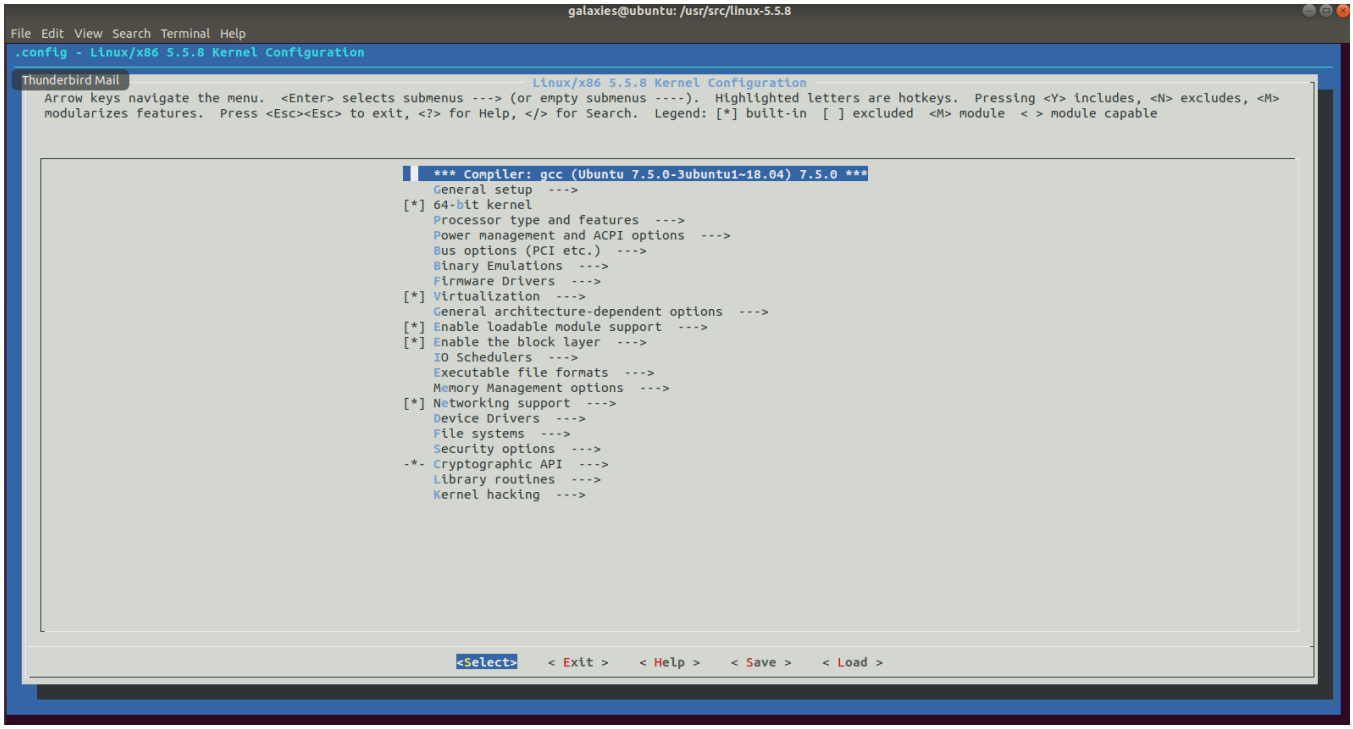


Figure 5: The configuration menu

We will use the default settings so we can exit the menu directly, and the configurations will be saved automatically.

1.2 Compilation

We use the following instruction to compile the kernel, where `-j3` means three threads compile in parallel so the compilation time will be reduced to only about 3 hours.

```
sudo make -j3
```

The following figure (Fig. 6) shows the beginning of compilation.

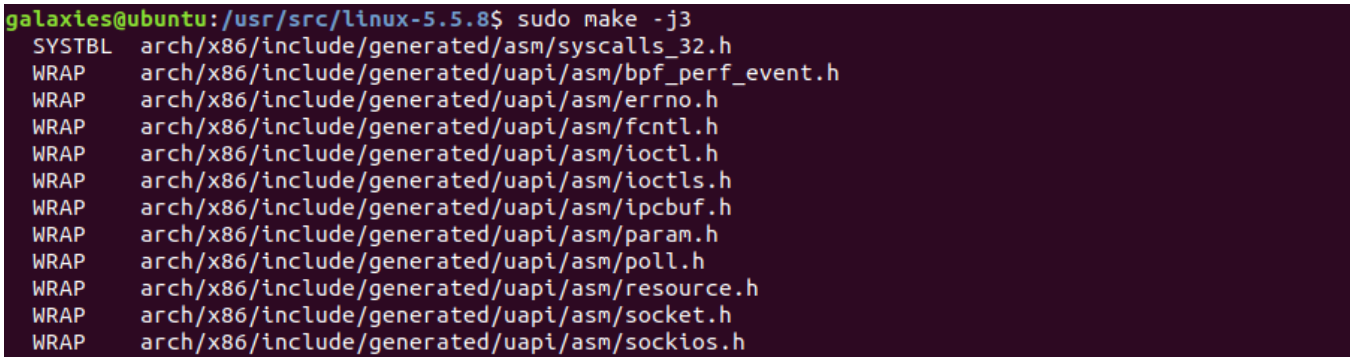


Figure 6: The beginning of compilation

Three hours later, the compilation process successfully finishes (Fig. 7).

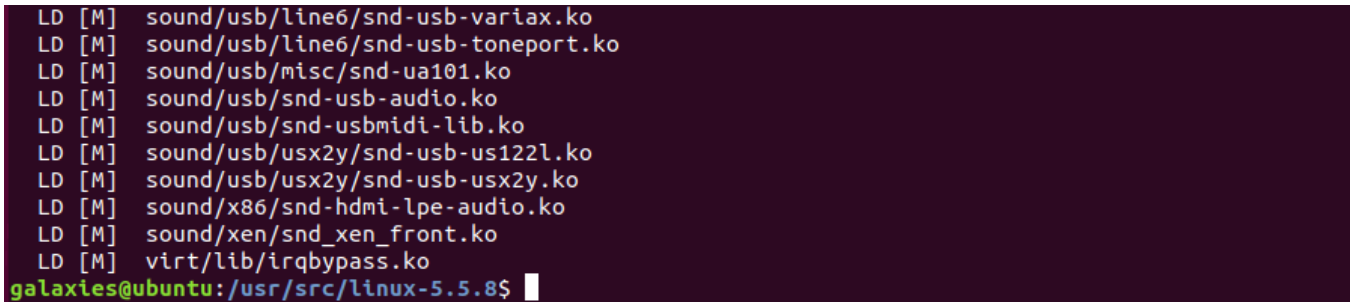


Figure 7: The end of compilation

1.3 Installation

After compilation, we can install the new kernel in our Ubuntu system. First we need to install kernel modules using the following instruction (Fig. 8).

```
sudo make modules_install
```

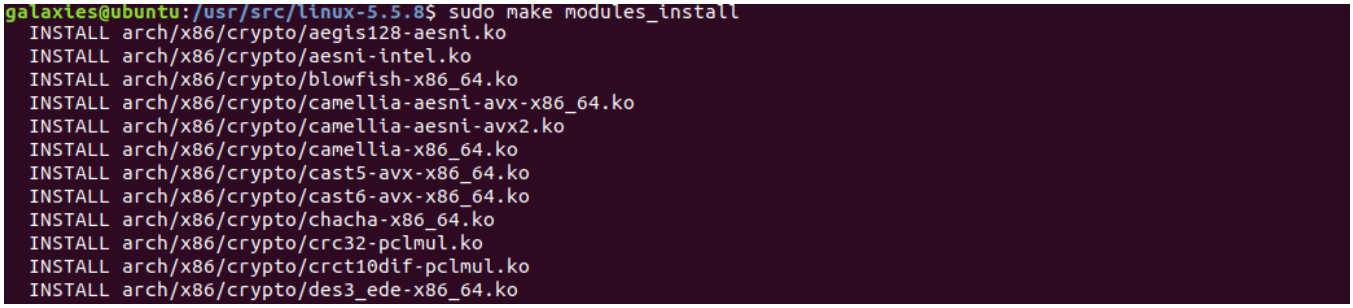


Figure 8: The beginning of kernel modules installation

A few moments later, the modules are successfully installed (Fig. 9).

```
  INSTALL sound/x86/snd-hdmi-lpe-audio.ko
  INSTALL sound/xen/snd_xen_front.ko
  INSTALL virt/lib/irqbypass.ko
  DEPMOD  5.5.8
galaxies@ubuntu:/usr/src/linux-5.5.8$
```

Figure 9: The end of kernel modules installation

After that, we can install kernel using the following instruction.

```
1 sudo make install
```

```
galaxies@ubuntu:/usr/src/linux-5.5.8$ sudo make install
sh ./arch/x86/boot/install.sh 5.5.8 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.5.8 /boot/vmlinuz-5.5.8
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.5.8 /boot/vmlinuz-5.5.8
update-initramfs: Generating /boot/initrd.img-5.5.8
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.5.8 /boot/vmlinuz-5.5.8
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.5.8 /boot/vmlinuz-5.5.8
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.5.8 /boot/vmlinuz-5.5.8
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.5.8
Found initrd image: /boot/initrd.img-5.5.8
Found linux image: /boot/vmlinuz-5.3.0-47-generic
Found initrd image: /boot/initrd.img-5.3.0-47-generic
Found linux image: /boot/vmlinuz-5.3.0-40-generic
Found initrd image: /boot/initrd.img-5.3.0-40-generic
Found linux image: /boot/vmlinuz-5.3.0-28-generic
Found initrd image: /boot/initrd.img-5.3.0-28-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

Figure 10: Kernel installation

Then we can reboot the system using the following instruction.

```
1 reboot
```

1.4 Result

After system rebooting, the kernel is successfully updated. We can use `uname` instruction mentioned above to check the new version (Fig. 11).

```
galaxies@ubuntu:~$ uname -a
Linux ubuntu 5.5.8 #1 SMP Mon Apr 27 20:14:52 PDT 2020 x86_64 x86_64 x86_64 GNU/Linux
```

Figure 11: The new Linux Kernel version

We can use the following instruction to find out all the program installed in the computer, and we can search “linux” to find out all the kernel versions.

```
1 sudo dpkg --get-selections
```

Then we can use the following instruction to clear the old version kernel (Fig. 12).

```
1 sudo apt-get purge linux-image-5.3.0-40-generic
```

```
galaxies@ubuntu:~$ sudo apt-get purge linux-image-5.3.0-40-generic
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  efibootmgr libfwup1 libwayland-egl1-mesa linux-headers-5.3.0-28
  linux-headers-5.3.0-28-generic linux-image-5.3.0-28-generic
  linux-modules-5.3.0-28-generic linux-modules-extra-5.3.0-28-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  linux-image-unsigned-5.3.0-40-generic
Suggested packages:
  fdutils linux-hwe-doc-5.3.0 | linux-hwe-source-5.3.0 linux-hwe-tools
The following packages will be REMOVED:
  linux-image-5.3.0-40-generic* linux-modules-extra-5.3.0-40-generic*
The following NEW packages will be installed:
  linux-image-unsigned-5.3.0-40-generic
0 upgraded, 1 newly installed, 2 to remove and 0 not upgraded.
Need to get 8,776 kB of archives.
After this operation, 181 MB disk space will be freed.
```

Figure 12: Remove the old version kernel

2 Designing Kernel Module

In this section, we will talk about the projects and the assignments in the textbook, and we will provide our solutions to assignments.

2.1 The Simple Module

We are first required to print out the Golden Ratio Prime in the `simple_init()` function and print out the greatest common divisor of 3300 and 24 in the `simple_exit()` function.

Actually this part is very simple, we only need to call the `gcd()` function in the `<linux/gcd.h>` library and use the `GOLDEN_RATIO_PRIME` in the `<linux/hash.h>` library.

The `simple.c` program is displayed as follows.

```
1  # include <linux/gcd.h>
2  # include <linux/hash.h>
3  # include <linux/init.h>
4  # include <linux/kernel.h>
5  # include <linux/module.h>
6
7  int simple_init(void) {
8    printk(KERN_INFO "Loading Kernel Module\n");
9    printk(KERN_INFO "The golden ratio prime is: %llu\n", GOLDEN_RATIO_PRIME);
10   return 0;
11 }
12
13 void simple_exit(void) {
14   printk(KERN_INFO "Removing Kernel Module\n");
15   printk(KERN_INFO "The greatest common divisor of %d and %d is: %lu\n", 3300, 24,
         gcd(3300, 24));
16 }
17
```

```
module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("Galaxies");
```

The `Makefile` file is as follows.

```
obj-m := simple.o

all:
	make -C /usr/src/linux-5.5.8/ M=$(shell pwd) modules
clean:
	make -C /usr/src/linux-5.5.8/ M=$(shell pwd) clean
```

We can use the following instruction to test the simple module.

```
sudo make
sudo dmesg -C
sudo insmod simple.ko
sudo dmesg
sudo rmmod simple
sudo dmesg
```

The execution results of these instructions are as follows (Fig. 13).

```
galaxies@ubuntu:~/CS307-Projects/Project1/simple$ sudo make
make -C /usr/src/linux-5.5.8/ M=/home/galaxies/CS307-Projects/Project1/simple modules
make[1]: Entering directory '/usr/src/linux-5.5.8'
  CC [M]  /home/galaxies/CS307-Projects/Project1/simple/simple.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/galaxies/CS307-Projects/Project1/simple/simple.mod.o
  LD [M]  /home/galaxies/CS307-Projects/Project1/simple/simple.ko
make[1]: Leaving directory '/usr/src/linux-5.5.8'
galaxies@ubuntu:~/CS307-Projects/Project1/simple$ sudo dmesg -C
galaxies@ubuntu:~/CS307-Projects/Project1/simple$ sudo insmod simple.ko
galaxies@ubuntu:~/CS307-Projects/Project1/simple$ dmesg
[ 4986.469959] Loading Kernel Module
[ 4986.469960] The golden ratio prime is: 7046029254386353131
galaxies@ubuntu:~/CS307-Projects/Project1/simple$ sudo rmmod simple
galaxies@ubuntu:~/CS307-Projects/Project1/simple$ dmesg
[ 4986.469959] Loading Kernel Module
[ 4986.469960] The golden ratio prime is: 7046029254386353131
[ 4997.040196] Removing Kernel Module
[ 4997.040198] The greatest common divisor of 3300 and 24 is: 12
```

Figure 13: The execution results of these instructions in simple module

We are also required to print out the value of `HZ` and `jiffies` in the `simple_init()` function, and print out the value of `jiffies` in the `simple_exit()` function. This requirement is basically a simple version of the assignment, which we will discussed in the next sub-section. So we do not discuss further here.

2.2 The Jiffies Module

Design a kernel module that creates a `/proc` file named `/proc/jiffies` that reports the current value of `jiffies` when the `/proc/jiffies` file is read, such as with the command:

```
cat /proc/jiffies
```

Be sure to remove `/proc/jiffies` when the module is removed.

Solution. The problem is very similar with the `hello` module example provided in textbook. Therefore we can use the similar method to write this kernel module. Here are some explanations.

- `jiffies` is a system variable defined in the `linux/jiffies.h`, which means the total clock interrupt times since the system started. Therefore, its value is changing all the time, and we have to include the corresponding headfile to use `jiffies`.
- The data type of `jiffies` is `unsigned long volatile` where `volatile` means it cannot be optimized by compiler, so we need to use `%lu` to output its value.

The `jiffies.c` program is displayed as follows.

```
# include <linux/init.h>
# include <linux/kernel.h>
# include <linux/module.h>
# include <linux/proc_fs.h>
# include <asm/uaccess.h>
# include <linux/uaccess.h>
# include <linux/jiffies.h> // jiffies is in this headfile

# define BUFFER_SIZE 128
# define PROC_NAME "jiffies"

ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *
    pos);

static struct file_operations proc_ops = {
  .owner = THIS_MODULE,
  .read = proc_read,
};

int proc_init(void) {
  /* create /proc files */
  proc_create(PROC_NAME, 0666, NULL, &proc_ops);
  printk(KERN_INFO "/proc/" PROC_NAME " is created!\n");
  return 0;
}

void proc_exit(void) {
  /* remove /proc files */
  remove_proc_entry(PROC_NAME, NULL);
  printk(KERN_INFO "/proc/" PROC_NAME " is removed!\n");
}

```

```
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, "The current value of jiffies is: %lu\n", jiffies);

  copy_to_user(usr_buf, buffer, rv);

  return rv;
}

module_init(proc_init);
module_exit(proc_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Jiffies Module");
MODULE_AUTHOR("Galaxies");
```

The `Makefile` file is as follows.

```
obj-m := jiffies.o

all:
	make -C /usr/src/linux-5.5.8/ M=$(shell pwd) modules
clean:
	make -C /usr/src/linux-5.5.8/ M=$(shell pwd) clean
```

We can use the following instruction to test the jiffies module.

```
sudo make
sudo dmesg -C
sudo insmod jiffies.ko
sudo dmesg
cat /proc/jiffies
cat /proc/jiffies
sudo rmmod jiffies
sudo dmesg
```

The execution results of these instructions are as follows (Fig. 14).

```
galaxies@ubuntu:~/projects/1/jiffies$ sudo make
make -C /usr/src/linux-5.5.8/ M=/home/galaxies/projects/1/jiffies modules
make[1]: Entering directory '/usr/src/linux-5.5.8'
  CC [M]  /home/galaxies/projects/1/jiffies/jiffies.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/galaxies/projects/1/jiffies/jiffies.mod.o
  LD [M]  /home/galaxies/projects/1/jiffies/jiffies.ko
make[1]: Leaving directory '/usr/src/linux-5.5.8'
galaxies@ubuntu:~/projects/1/jiffies$ sudo dmesg -C
galaxies@ubuntu:~/projects/1/jiffies$ sudo insmod jiffies.ko
galaxies@ubuntu:~/projects/1/jiffies$ sudo dmesg
[ 4860.587572] /proc/jiffies is created!
galaxies@ubuntu:~/projects/1/jiffies$ cat /proc/jiffies
The current value of jiffies is: 4296114696
galaxies@ubuntu:~/projects/1/jiffies$ cat /proc/jiffies
The current value of jiffies is: 4296115555
galaxies@ubuntu:~/projects/1/jiffies$ sudo rmmod jiffies
galaxies@ubuntu:~/projects/1/jiffies$ sudo dmesg
[ 4860.587572] /proc/jiffies is created!
[ 4901.577188] /proc/jiffies is removed!
```

Figure 14: The execution results of the testing instructions of jiffies module

□

2.3 The Seconds Module

Design a kernel module that creates a `proc` file named `/proc/seconds` that reports the number of elapsed seconds since the kernel module was loaded. This will involve using the value of `jiffies` as well as the `HZ` rate. When a user enters the command

```
cat /proc/jiffies
```

your kernel module will report the number of seconds that have elapsed since the kernel module was first loaded. Be sure to remove `/proc/seconds` when the module is removed.

Solution. We use the similar method (like the methods of the previous assignment) to solve this problem. Here are some explanations.

- We use a variable to memorize the `jiffies` when we load the kernel module, say `begin_count`. Then every time we need to know the number of seconds that have elapsed since the kernel module was first loaded, we can use the formula `(jiffies - begin_count) / HZ` to get the answer. Notice that kernel mode do not support floating number operations, therefore we can only give the integer answer to the number of seconds.

- `HZ` is a system macro definition defined in the `linux/param.h`, which means the system clock frequency. Therefore, its value is changing all the time, and we have to include the corresponding headfile to use `HZ` (the headfile `linux/jiffies.h` has already included `linux/param.h`, so we do not need to write it again).

- The data type of `jiffies` is `unsigned long volatile`, so we need to use the same data type to store the initial value of `jiffies`.

The `seconds.c` program is displayed as follows.

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

```
3  # include <linux/module.h>
4  # include <linux/proc_fs.h>
5  # include <asm/uaccess.h>
6  # include <linux/uaccess.h>
7  # include <linux/jiffies.h> // jiffies is in this headfile
8
9  # define BUFFER_SIZE 128
10 # define PROC_NAME "seconds"
11
12 unsigned long volatile begin_count, seconds;
13
14 ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *
       pos);
15
16 static struct file_operations proc_ops = {
17   .owner = THIS_MODULE,
18   .read = proc_read,
19 };
20
21 int proc_init(void) {
22   /* create /proc files */
23   proc_create(PROC_NAME, 0666, NULL, &proc_ops);
24   printk(KERN_INFO "/proc/" PROC_NAME " is created!\n");
25   begin_count = jiffies;
26   return 0;
27 }
28
29 void proc_exit(void) {
30   /* remove /proc files */
31   remove_proc_entry(PROC_NAME, NULL);
32   printk(KERN_INFO "/proc/" PROC_NAME " is removed!\n");
33 }
34
35 ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *
       pos) {
36   int rv = 0;
37   char buffer[BUFFER_SIZE];
38   static int completed = 0;
39
40   if (completed) {
41     completed = 0;
42     return 0;
43   }
44
45   completed = 1;
46
47   seconds = (jiffies - begin_count) / HZ;
48   rv = sprintf(buffer, "%lu seconds have elapsed since the kernel module was first
         loaded\n", seconds);
49
```

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

  return rv;
}

module_init(proc_init);
module_exit(proc_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Seconds Module");
MODULE_AUTHOR("Galaxies");
```

The `Makefile` file is as follows.

```
obj-m := seconds.o

all:
	make -C /usr/src/linux-5.5.8/ M=$(shell pwd) modules
clean:
	make -C /usr/src/linux-5.5.8/ M=$(shell pwd) clean
```

We can use the following instruction to test the seconds module.

```
sudo make
sudo dmesg -C
sudo insmod seconds.ko
sudo dmesg
cat /proc/seconds
cat /proc/seconds
sudo rmmod seconds
sudo dmesg
```

The execution results of these instructions are as follows (Fig. 15).

```
galaxies@ubuntu:~/projects/1/seconds$ sudo make
make -C /usr/src/linux-5.5.8/ M=/home/galaxies/projects/1/seconds modules
make[1]: Entering directory '/usr/src/linux-5.5.8'
  CC [M]  /home/galaxies/projects/1/seconds/seconds.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/galaxies/projects/1/seconds/seconds.mod.o
  LD [M]  /home/galaxies/projects/1/seconds/seconds.ko
make[1]: Leaving directory '/usr/src/linux-5.5.8'
galaxies@ubuntu:~/projects/1/seconds$ sudo dmesg -C
galaxies@ubuntu:~/projects/1/seconds$ sudo insmod seconds.ko
galaxies@ubuntu:~/projects/1/seconds$ sudo dmesg
[ 6701.768556] /proc/seconds is created!
galaxies@ubuntu:~/projects/1/seconds$ cat /proc/seconds
15 seconds have elapsed since the kernel module was first loaded
galaxies@ubuntu:~/projects/1/seconds$ cat /proc/seconds
19 seconds have elapsed since the kernel module was first loaded
galaxies@ubuntu:~/projects/1/seconds$ sudo rmmod seconds
galaxies@ubuntu:~/projects/1/seconds$ sudo dmesg
[ 6701.768556] /proc/seconds is created!
[ 6729.473942] /proc/seconds is removed!
```

Figure 15: The execution results of the testing instructions of seconds module

□

3 Personal Thoughts

During this project, I've experienced the process of compiling Linux Kernel and installing Linux Kernel. I also develop an brief understanding about Kernel Module and `/proc` file system, and I try to write the kernel modules related to `/proc` file system in the assignments. This experience benefits me a lot. The project helps me enhance my understanding of Linux Kernel and helps me improve my Linux Kernel programming skills.

By the way, you can find all the source codes in the "src" folder. You can also refer to my github to see my codes of this project, and they are in the `Project1` folder.