Project1: Introduction to Linux Kernel Modules

Preparation:

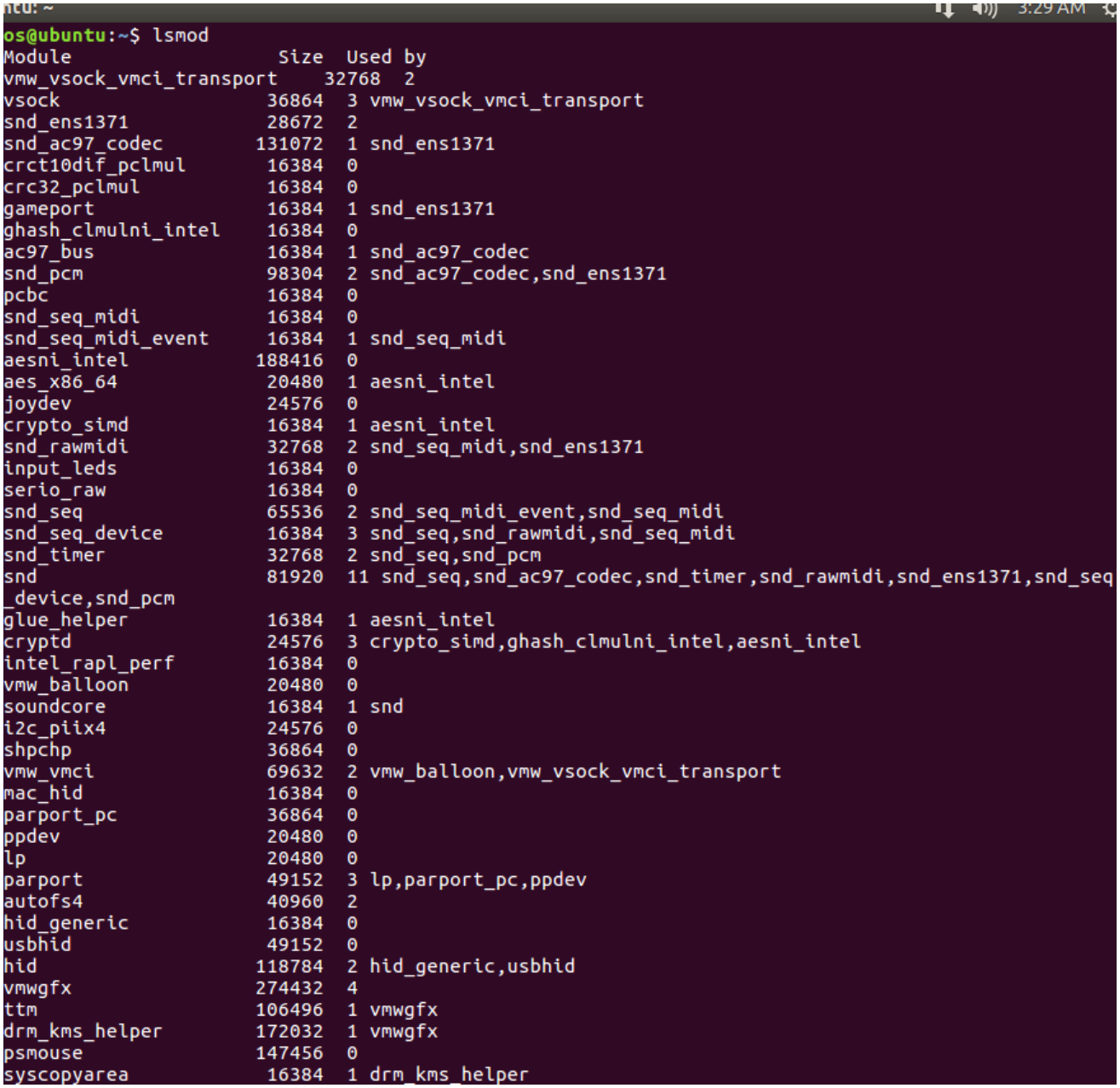
Install Ubuntu:

I install ubuntu 14.04 (download from website and its kernel is 4.4)on the Vmware Workstation 15. Not the double system as it maybe destory the guidance and the system cannot use.

What's more, do not install the ubuntu 16.04 because this index will cause the failure when compiling *.c files. And I cann't solve it after I look up this question in Google and Baidu.

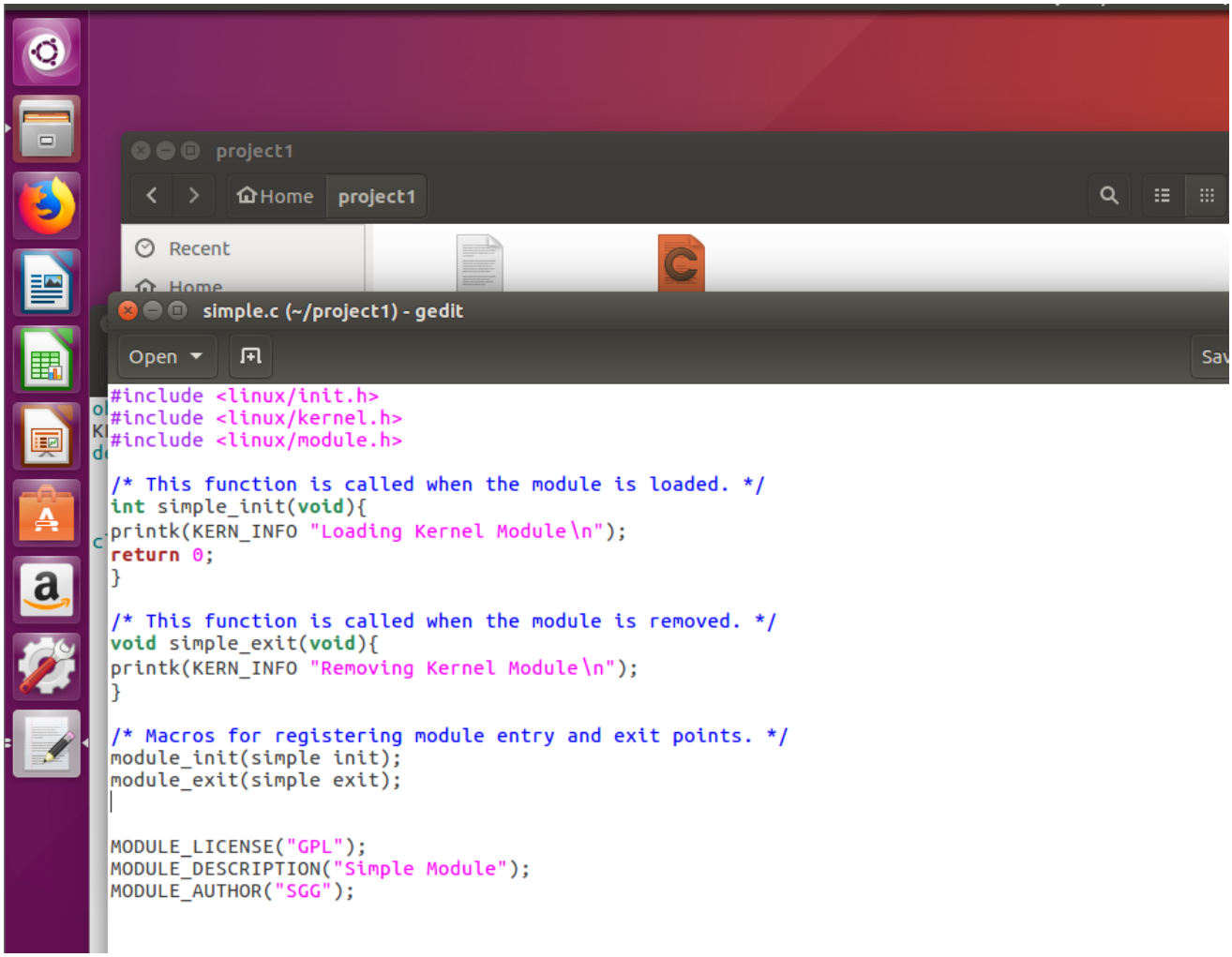
Procedure:

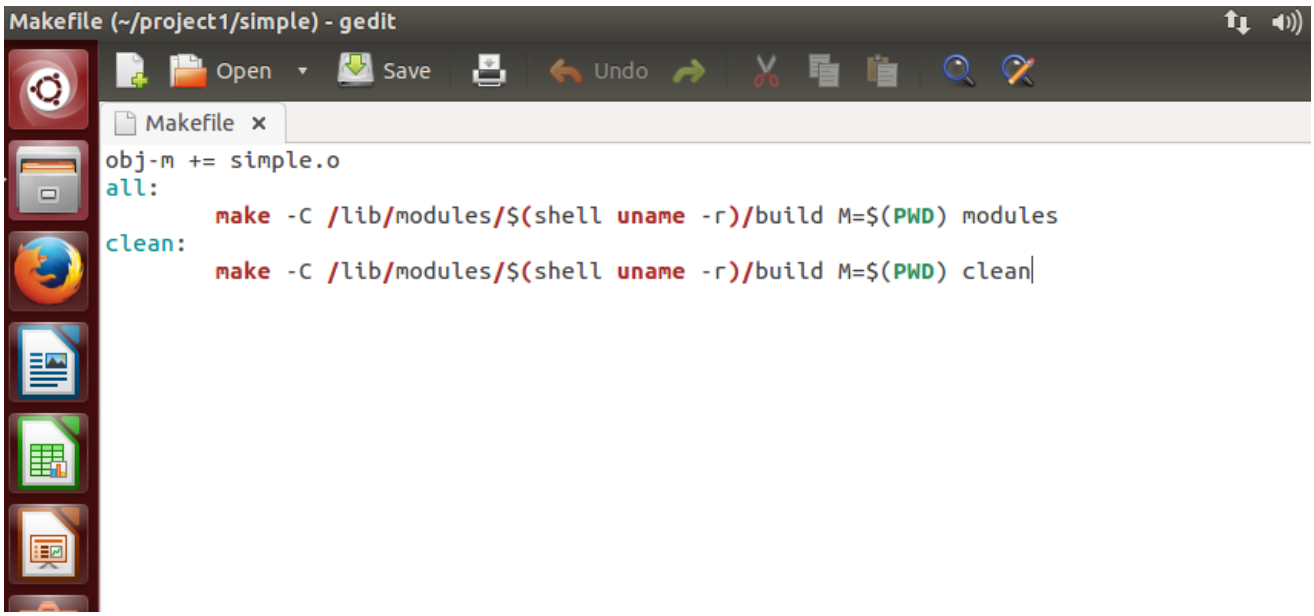
1. Commands: lsmod, this command will list all the modules in the kernel. Just like the figures below:



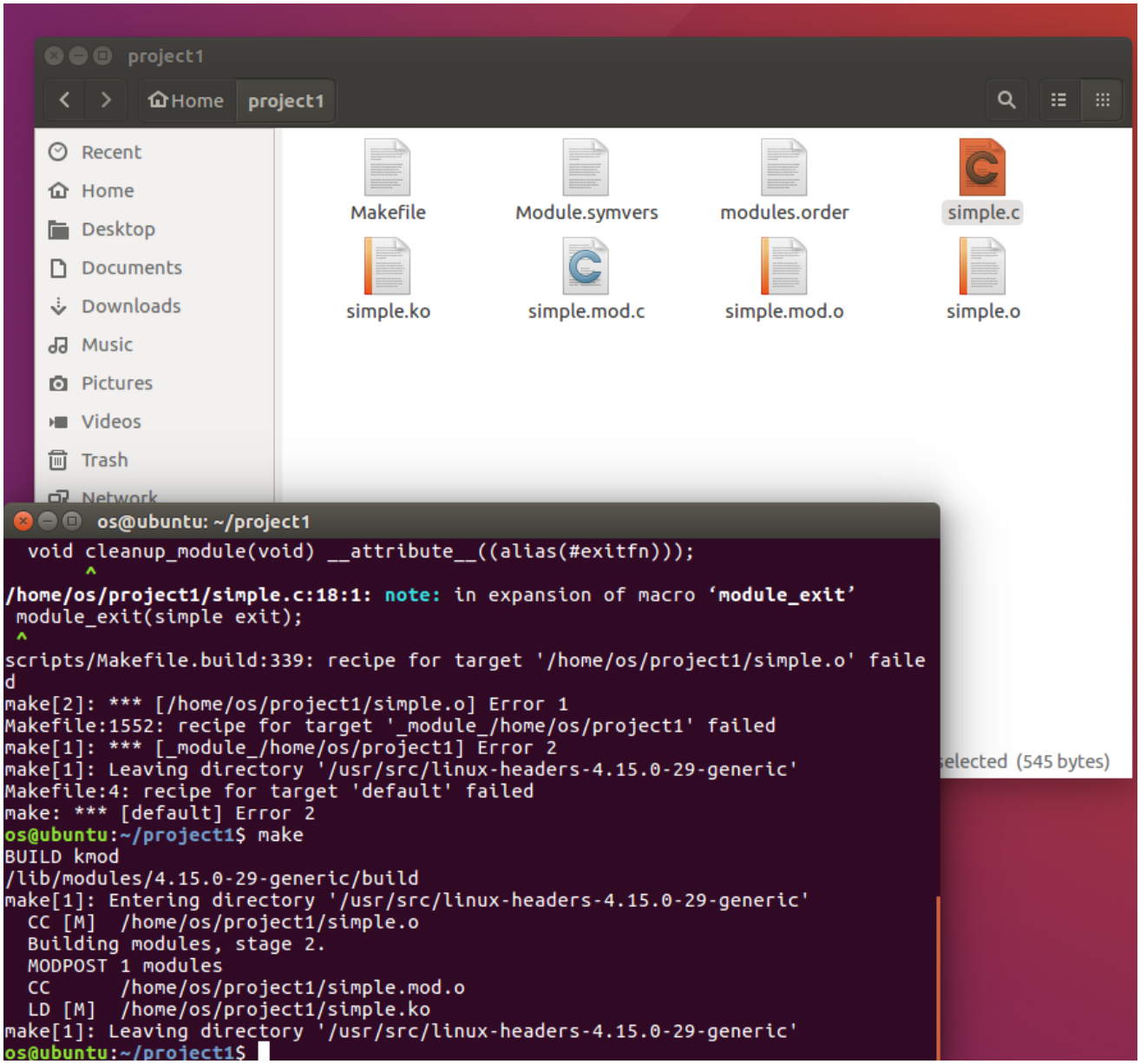
```
os@ubuntu:~$ lsmod
Module                  Size  Used by
vmw_vsock_vmci_transport    32768  2
vsock                   36864  3 vmw_vsock_vmci_transport
snd_ens1371             28672  2
snd_ac97_codec         131072  1 snd_ens1371
crct10dif_pclmul        16384  0
crc32_pclmul            16384  0
gameport                16384  1 snd_ens1371
ghash_clmulni_intel     16384  0
ac97_bus                16384  1 snd_ac97_codec
snd_pcm                 98304  2 snd_ac97_codec,snd_ens1371
pcbc                    16384  0
snd_seq_midi            16384  0
snd_seq_midi_event      16384  1 snd_seq_midi
aesni_intel            188416  0
aes_x86_64              20480  1 aesni_intel
joydev                  24576  0
crypto_simd             16384  1 aesni_intel
snd_rawmidi             32768  2 snd_seq_midi,snd_ens1371
input_leds              16384  0
serio_raw               16384  0
snd_seq                 65536  2 snd_seq_midi_event,snd_seq_midi
snd_seq_device          16384  3 snd_seq,snd_rawmidi,snd_seq_midi
snd_timer               32768  2 snd_seq,snd_pcm
snd                     81920  11 snd_seq,snd_ac97_codec,snd_timer,snd_rawmidi,snd_ens1371,snd_seq
_device,snd_pcm
glue_helper             16384  1 aesni_intel
cryptd                  24576  3 crypto_simd,ghash_clmulni_intel,aesni_intel
intel_rapl_perf         16384  0
vmw_balloon             20480  0
soundcore               16384  1 snd
i2c_piix4               24576  0
shpchp                  36864  0
vmw_vmci                69632  2 vmw_balloon,vmw_vsock_vmci_transport
mac_hid                 16384  0
parport_pc              36864  0
ppdev                   20480  0
lp                      20480  0
parport                 49152  3 lp,parport_pc,ppdev
autofs4                 40960  2
hid_generic             16384  0
usbhid                  49152  0
hid                    118784  2 hid_generic,usbhid
vmwgfx                 274432  4
ttm                    106496  1 vmwgfx
drm_kms_helper         172032  1 vmwgfx
psmouse                147456  0
syscopyarea             16384  1 drm_kms_helper
```

2. Then I create the simple.c and filled up with the example code. And create the Makefile file.

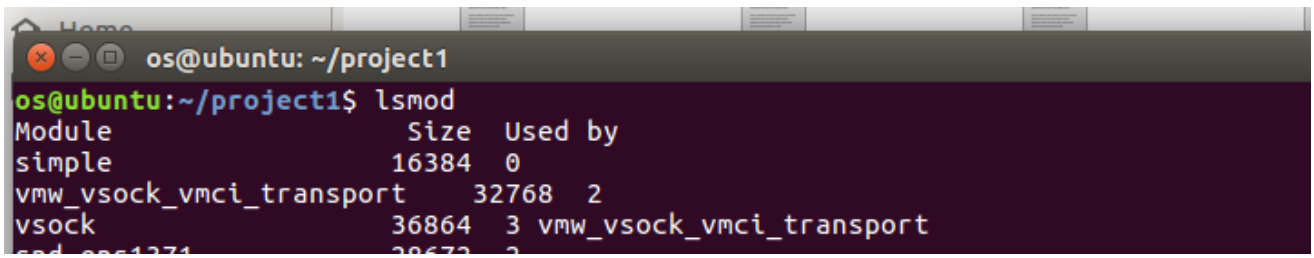




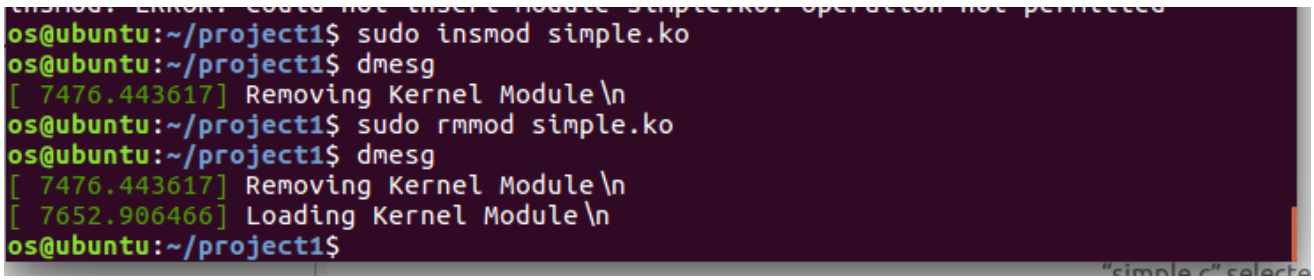
After make command, it produce many files include *.ko file, the result.



And I found the module in the lsmod.



And I check the message by using dmesg command. Just as following:



3. /Proc FIle System.

Just like the code in the sample, I create the hello.c file, then compile it. After compile, I inserted it into the kernel module. Then cat /proc/hello. The result is as following:

```
oskernel@ubuntu:~/project1$ sudo insmod hello.ko
oskernel@ubuntu:~/project1$ cat /proc/hello
Hello World
oskernel@ubuntu:~/project1$ sudo rmmod hello.ko
oskernel@ubuntu:~/project1$ dmesg
[ 1000.069443] /proc/hello created
[ 1019.710443] /proc/hello removed
[ 1046.461622] e1000: eth0 NIC Link is Down
[ 1052.477732] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: N
one
[ 1054.481466] e1000: eth0 NIC Link is Down
[ 1058.490132] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: N
one
```

Practise:

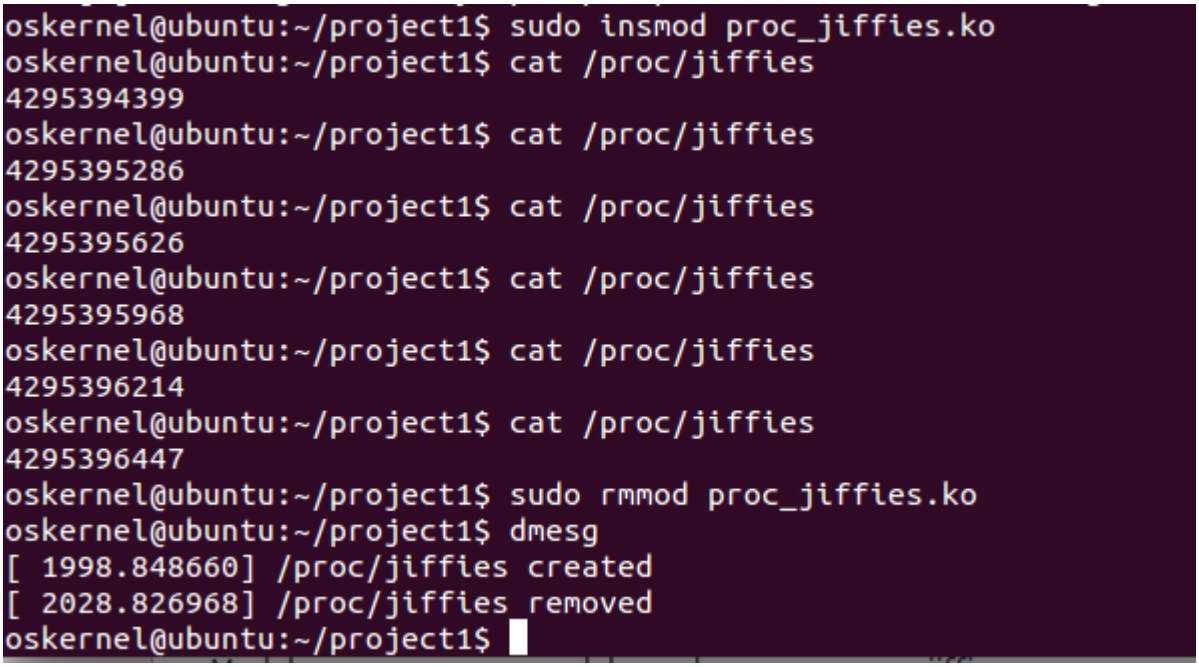
1. Print out the value of the GOLDEN_RATIO_PRIME in the init function and print out the greatest commaon divisor of 3300 and 24 in the exit function.

```
os@ubuntu:~/project1$ sudo insmod simple.ko
os@ubuntu:~/project1$ sudo rmmod simple.ko
os@ubuntu:~/project1$ dmesg
[ 7476.443617] Removing Kernel Module\n
[ 7652.906466] Loading Kernel Module\n
[ 7667.428429] Removing Kernel Module\n
[ 8479.252742] Loading Kernel Module
[ 8479.252743] 7046029254386353131
[ 8516.783787] Removing Kernel Module
[ 8516.783788] 12
[ 8537.214433] Loading Kernel Module
[ 8537.214434] 7046029254386353131
[ 8544.688635] Removing Kernel Module
[ 8544.688637] 12
```

2. print out the values of jiffies and HZ in the init function and print out the value of jiffies in the exit function.

```
os@ubuntu:~/project1$ sudo insmod simple.ko
os@ubuntu:~/project1$ sudo rmmod simple.ko
os@ubuntu:~/project1$ dmesg
[ 9082.171384] Loading Kernel Module
[ 9082.171385] 7046029254386353131
[ 9082.171386] 250
[ 9082.171386] 4297162887
[ 9085.020070] Removing Kernel Module
[ 9085.020071] 12
[ 9085.020072] 4297163599
```

3. print the jiffies every time I cat it.



4. print the seconds after I insert the module.

   In the code, I use global var to record the start jiffies when inserting the modules. Then everytime I cat it, it will compute the difference between start jiffies and the current jiffies. Then divide by HZ. I can get the seconds then print it.

```
oskernel@ubuntu:~/project1$ make
make -C /lib/modules/4.4.0-31-generic/build M=/home/oskernel/project1 modules
make[1]: Entering directory `/usr/src/linux-headers-4.4.0-31-generic'
  CC [M]  /home/oskernel/project1/proc_seconds.o
/home/oskernel/project1/proc_seconds.c: In function ‘proc_read’:
/home/oskernel/project1/proc_seconds.c:87:9: warning: ISO C90 forbids mixed decl
arations and code [-Wdeclaration-after-statement]
         int pass_time = (jiffies-start_time)/HZ;
         ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/oskernel/project1/proc_seconds.mod.o
  LD [M]  /home/oskernel/project1/proc_seconds.ko
make[1]: Leaving directory `/usr/src/linux-headers-4.4.0-31-generic'
oskernel@ubuntu:~/project1$ sudo insmod proc_seconds.ko
[sudo] password for oskernel:
oskernel@ubuntu:~/project1$ dmesg
[ 1008.848660] /proc/jiffies created
[illegible]3] /proc/jiffies removed
[ 3230.864273] /proc/seconds created
oskernel@ubuntu:~/project1$ cat /proc/seconds
14
oskernel@ubuntu:~/project1$ cat /proc/seconds
18
oskernel@ubuntu:~/project1$ cat /proc/seconds
19
oskernel@ubuntu:~/project1$ cat /proc/seconds
20
oskernel@ubuntu:~/project1$ cat /proc/seconds
21
oskernel@ubuntu:~/project1$ cat /proc/seconds
22
oskernel@ubuntu:~/project1$ cat /proc/seconds
23
oskernel@ubuntu:~/project1$ sudo rmmod proc_seconds.ko
oskernel@ubuntu:~/project1$
```

Codes:

I will add the notation to the codes where I have changed.

1. Here is the code for 1、2 practise.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.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. */
int simple_init(void){
printk(KERN_INFO "Loading Kernel Module\n");
// This code is print Golden ratio prime, when init the simple module.
printk(KERN_INFO "%lu\n", GOLDEN_RATIO_PRIME);
// This code is print HZ, when init the simple module.
printk(KERN_INFO "%lu\n", HZ);
```

```
// This code is print jiffies, when init the simple module.
printk(KERN_INFO "%lu\n", jiffies);
return 0;
}

/* This function is called when the module is removed. */
void simple_exit(void){
printk(KERN_INFO "Removing Kernel Module\n");
// This code is print the gcd of 3300 and 24, when exit the simple module.
printk(KERN_INFO "%lu\n", gcd(3300,24));
// This code is print jiffies, when exit the simple module.
printk(KERN_INFO "%lu\n", jiffies);
}

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

/* The Module description. */
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("Bing Han");
```

2. Here is the code for 3 practise.

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

#define BUFFER_SIZE 128
#define PROC_NAME "jiffies"
#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);
}
/**
 * This function is called each time the /proc/hello 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.
 *
 * params:
 *
 * file:
 * buf: buffer in user space
 * count:
 * pos:
 */
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;
    // I just add this line, to print the jiffies when cat the proc file.
        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);
/* The Module description. */
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("jiffies Module");
MODULE_AUTHOR("Bing Han");
```

3. Here is the code for 4 practice:

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

```c
#include <linux/jiffies.h>
#include <linux/hash.h>
#include <asm/uaccess.h>

#define BUFFER_SIZE 128
#define PROC_NAME "seconds"
#define MESSAGE "Hello World\n"

/**
 * Function prototypes
 */
// I create a global var to store the start_jiffies when create the file.
unsigned long start_time = 0;

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);
    // When initing the file, store the start_jiffies.
        start_time = jiffies;
        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);
}
/**
 * This function is called each time the /proc/hello 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.
 *
 * params:
 *
 * file:
 * buf: buffer in user space
 * count:
 * pos:
```

```
 */
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;
    // everytime cat the file, to get the difference between the current_jiffies
and start_jiffies, then divide by HZ to get the time past.
        int pass_time = (jiffies-start_time)/HZ;
    //print the time past.
        rv = sprintf(buffer, "%d\n", pass_time);
        // 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);
/* The Module description. */
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("seconds Module");
MODULE_AUTHOR("Bing Han");
```