CS356 Project-1

HaotianXue 518021910506

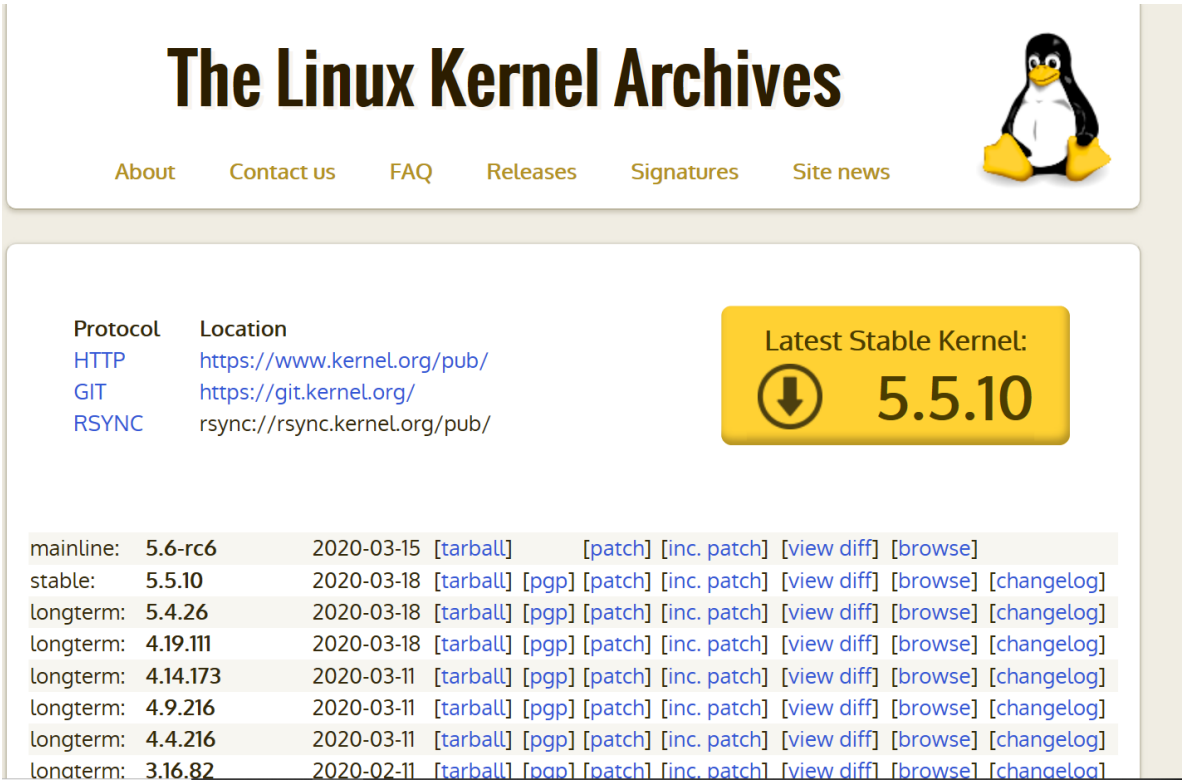
Introduction

The project-1 for CS356 is aimed to encourage us to explore the linux kernel in three aspects. In the first section we ***compile a new kernel*** for the present linux virtual machine. Then we try to program and ***insert new kernel modules*** into the kernel. Lastly, we learn about the /proc file system in linux and try to insert some simple kernel modules which will ***use the /proc system***.

Kernel Compilation

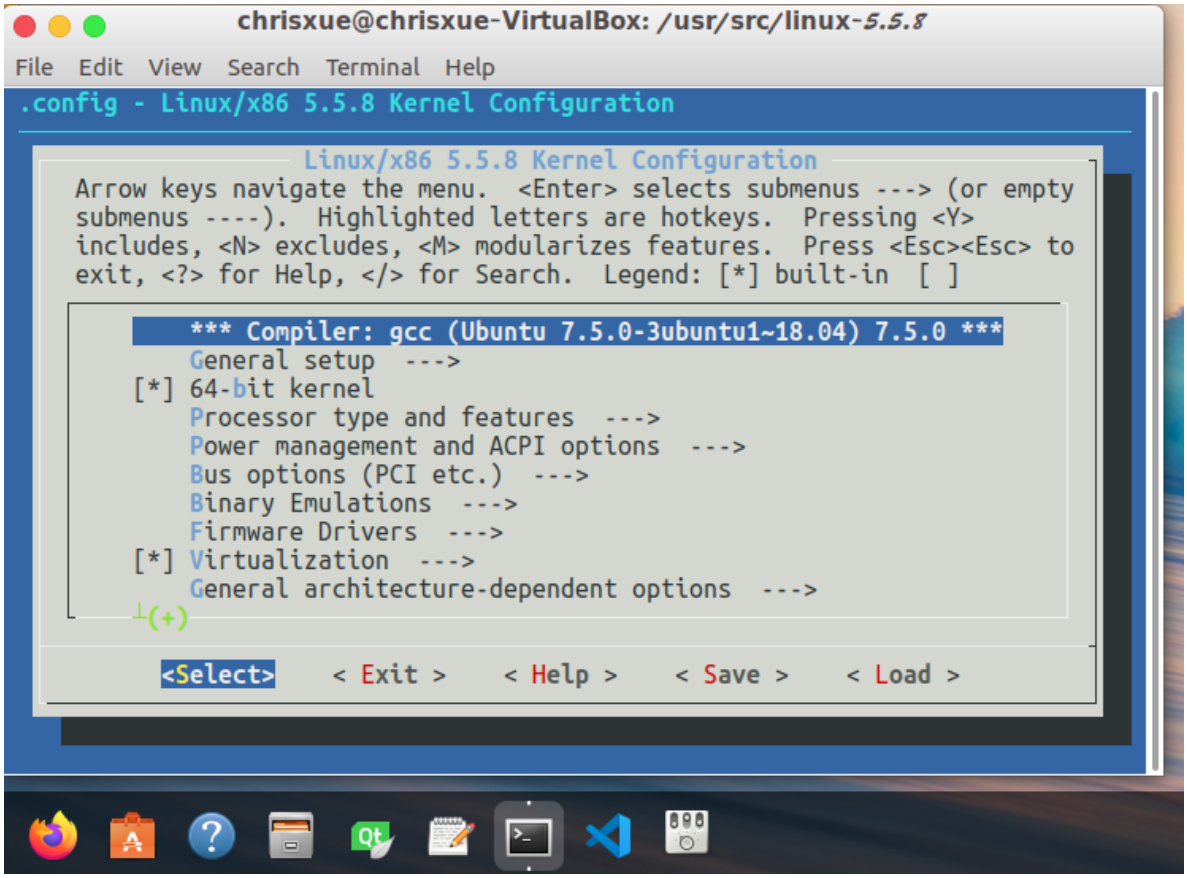
Basic Configurations

The original kernel version of my linux machine is 5.3.0, and I try to compile the newest kernel at that time: Linux 5.5.8(the picture below is fetched new weeks after the project is done). The raw code for the new kernel is downloaded from https://www.kernel.org/.



Compiling Process

We move the raw code under /usr/src and unzip it. Then we enter the linux-5.5.8 directory to do the compilation work. We use `make menuconfig` to set some configurations, where I just use the default parameters.



Before compiling, we install some necessary libraries which will be used in the next steps. What to install is determined by the error through out by the compiler.

```
chrisxue@chrisxue-VirtualBox:/usr/src/linux-5.5.8$ ls
arch            fs       LICENSES                net          usr
block           include  MAINTAINERS             README       virt
certs           init     Makefile                samples      vmlinux
COPYING         ipc      mm                      scripts      vmlinux-gdb.py
CREDITS         Kbuild   modules.builtin         security     vmlinux.o
crypto          Kconfig  modules.builtin.modinfo sound
Documentation   kernel   modules.order           System.map
drivers         lib      Module.symvers          tools
```

After doing necessary work, we use `make` to do the compilation, it should be noted that we can use `make -j4` to speed up the compilation. And it took me about 1hour to finish that.

Then we run `make modules_install` and `make install` , after that the new kernel is made successfully.

We use the `make clean` to clean some intermediate files generated during the compilation:

```
chrisxue@chrisxue-VirtualBox:/usr/src$ cd linux-5.5.8/
chrisxue@chrisxue-VirtualBox:/usr/src/linux-5.5.8$ sudo make clean
[sudo] password for chrisxue:
  CLEAN   arch/x86/entry/vdso
  CLEAN   arch/x86/kernel/cpu
  CLEAN   arch/x86/kernel
  CLEAN   arch/x86/purgatory
  CLEAN   arch/x86/realmode/rm
  CLEAN   arch/x86/lib
  CLEAN   certs
  CLEAN   crypto/asymmetric_keys
  CLEAN   drivers/eisa
  CLEAN   drivers/gpu/drm/radeon
  CLEAN   drivers/net/wan
  CLEAN   drivers/scsi/aic7xxx
  CLEAN   drivers/scsi
  CLEAN   drivers/tty/vt
  CLEAN   fs/unicode
  CLEAN   kernel/debug/kdb
  CLEAN   kernel
  CLEAN   lib/raid6
```

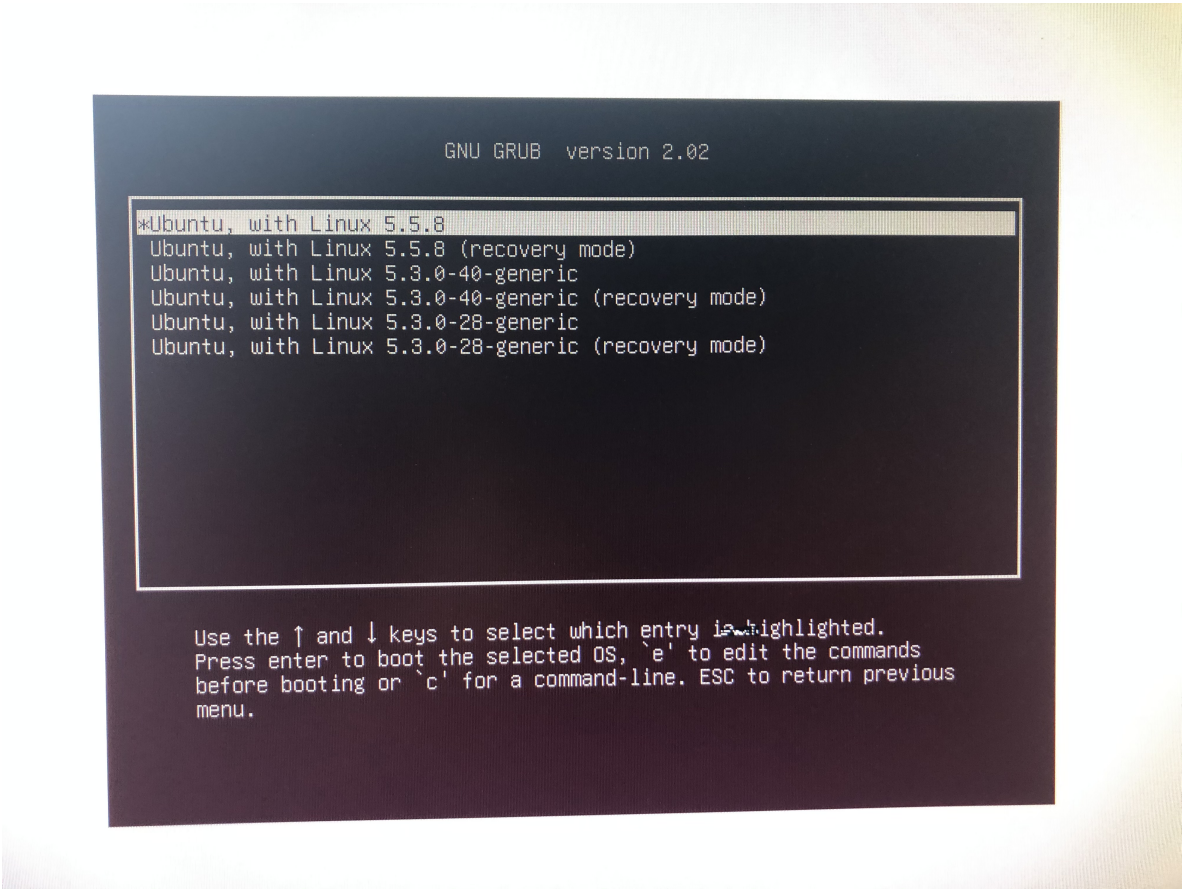
Compiling Result

We can run `uname -a` to check the current kernel version, and the original output and the output after doing kernel compilation is as follows:

```
chrisxue@chrisxue-VirtualBox:~$ uname -r
5.3.0-40-generic
chrisxue@chrisxue-VirtualBox:~$
```

```
chrisxue@chrisxue-VirtualBox:~$ uname -r
5.5.8
chrisxue@chrisxue-VirtualBox:~$
```

And we can also set the grub to choose the kernel we want when starting the system:



Loading and Removing Kernel Modules

Code Structure

`hello.c`

```
int my_init(void){
        printk("Loading kernel module--\ngolden ratio prime:%lu \njffies:%u\n HZ:%u\n", \
             GOLDEN_RATIO_PRIME, jiffies, HZ);
        return 0;
}

void my_exit(void){
        printk(KERN_INFO "Removing kernel module---\n gcd(3300,24) = %d\n jffies:%u\n", \
                        gcd(3300, 24), jiffies);
}


module_init(my_init);
module_exit(my_exit);
```

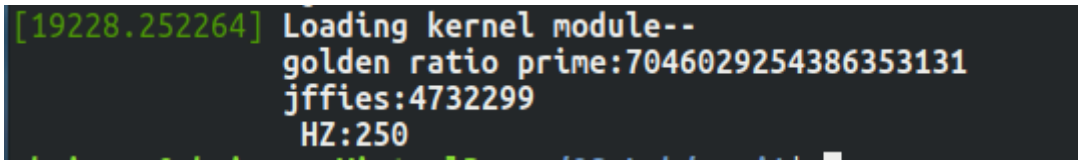
The main part is the initiation and exit function, where we output some parameters when we insert or remove the module into the kernel, and the function is designed as refer to the COS book Project-1 Page 3, to be specific:

- In `my_init`, we output the golden ratio prime, jiffies and HZ for the system
- In `my_exit`, we output the gcd for 3300 and 24 and the jiffies.

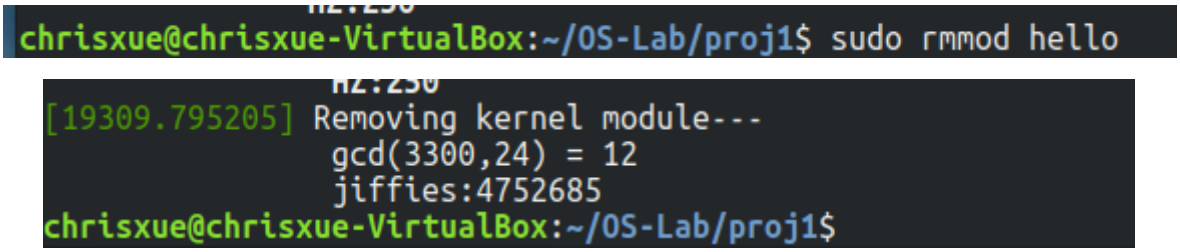
Results

After making the code, we first insert it into kernel using `sudo insmod hello.ko` and using `dmesg` to check the instances, we get:

```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj1$ sudo insmod hello.ko
[sudo] password for chrisxue:
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj1$ dmesg
```



Them we remove the modules using `sudo rmmod hello`, then we get the result using `dmesg`:



The /proc File System

Code Structure

`hellow.c`: (for jiffies)

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>   // different from <asm/uaccess.h> in the COS book
#include <linux/jiffies.h>
#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){
    printk("load hellow into kernel module!\n");
    proc_create(PROC_NAME, 0666, NULL, &proc_ops);
    return 0;
}

void proc_exit(void){
    printk("remove hellow from kernel module!\n");
    remove_proc_entry(PROC_NAME, NULL);
}

ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t
*pos){
    int rv=0;
```

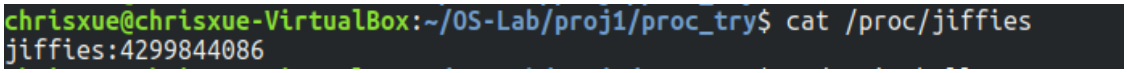
```
    char buffer[BUFFER_SIZE];
    static int completed  = 0;
    if(completed) {
       completed = 0;
       return 0;
    }
    completed = 1;
    rv = sprintf(buffer, "jiffies:%ld\n", jiffies);
    copy_to_user(usr_buf, buffer, rv);
    return rv;
}

module_init(proc_init);
module_exit(proc_exit);
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Hello Module");
MODULE_AUTHOR("SGG");
```

Results

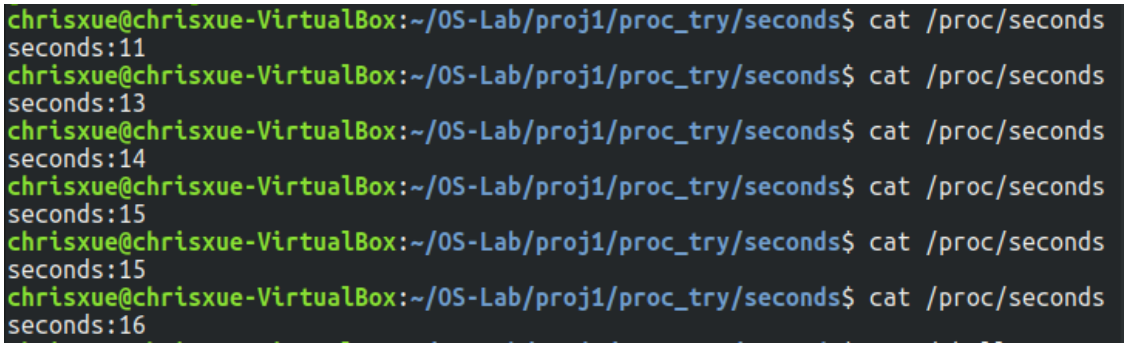
- cat /proc/jiffies

  First we `make` the code, then insert the module into kernel like what we did in section 2. That we call `cat /proc/jiffies` and get this:



- cat /proc/seconds

  Also we use the same method to implement this, and the result is:



Conclusion

This project helps me learn some basic concepts about linux kernel operations including inserting and recompile the linux kernel, removing a module into the kernel and utilizing the proc file system.

Thanks for the instructions and useful help offered by Prof. Wu and all the TAs!