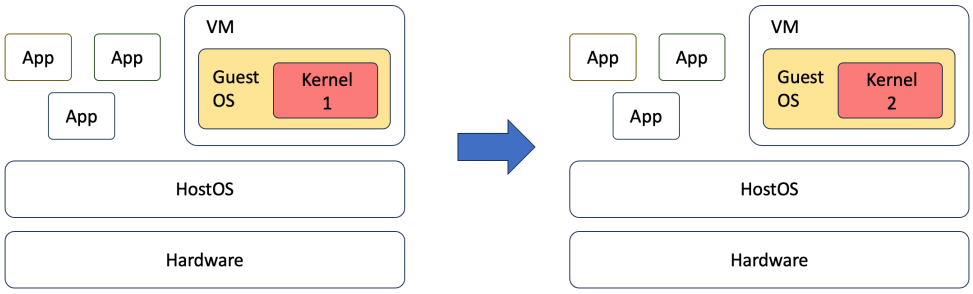
Kennesaw State University

CS 3502 Operating Systems

Project 1 - System call

Instructor: Kun Suo
Points Possible: 100

Part A: Build the Linux kernel (50 points) --- Please work this on VMs in your laptop



Create a virtual machine using VirtualBox or UTM (for Apple Silicon) on your laptop. As the kernel compiling is pretty large, please make sure your VM has at least 4GB memory and 80GB storage (if your disk is small, create a 60GB disk for the VM). For the Operating System, use Ubuntu 22.04 iso: https://cdimage.ubuntu.com/jammy/daily-live/current/. There exist images for two architectures: x86 (e.g., Intel, AMD) and arm (e.g., Apple silicon). Please select the one that fits your machine.

How to build one Ubuntu VM?

Windows 10 (x86):
https://www.youtube.com/watch?v=QbmRXJJKsvs
MacOS (x86):
https://www.youtube.com/watch?v=GDoCrfPma2k&t=321s
MacOS (arm):
https://youtu.be/O19mv1pe76M?si=4cYayFiqPNoHoY1w

Step 1: Get the Linux kernel code

Before you download and compile the Linux kernel source, make sure you have development tools installed on your system. We recommend you work this project on your virtual machine.

In Ubuntu, install this software using apt:

$ sudo apt-get install -y gcc libncurses5-dev make wget flex bison vim libssl-dev libelf-dev

To obtain the version of your current kernel, type:
$ uname -r
5.15

For newer distributions of Ubuntu, you can see 5.x or 6.x. Please screenshot it and save it as the original kernel version.

Here is the screenshot of my current kernel:

```
ksuo@ksuo-vm ~/linux-5.19> uname -r
5.15.0-79-generic
```

Then, download kernel 5.19 and extract the source:
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.19.tar.gz
$ tar xvzf linux-5.19.tar.gz

We will refer LINUX_SOURCE to the top directory of the kernel source. Go to the linux source code folder:
$ cd linux-5.19

Step 2: Configure your new kernel

Before compiling the new kernel, a .config file needs to be generated in the top directory of the kernel source. To generate the config file and make possible changes to the default kernel configurations, type:
$ make localmodconfig
Select all "N" if any questions on the terminal to minimize the configuration file.

Here, we avoid using $ make menuconfig to save the kernel compiling time. You can check .config using the following command under kernel folder.
(https://youtu.be/UyOGF4UOoR0)
$ ls -al

Here is a screenshot of my VM:

```
ksuo@ksuo-vm ~/linux-5.19> ls -al
total 1660
drwxrwxr-x  24 ksuo ksuo   4096 Aug 24 20:50 .
drwxr-x---  17 ksuo ksuo   4096 Aug 24 20:48 ..
drwxrwxr-x  24 ksuo ksuo   4096 Jul 31  2022 arch
drwxrwxr-x   3 ksuo ksuo  12288 Aug 24 20:51 block
drwxrwxr-x   2 ksuo ksuo   4096 Aug 24 20:48 certs
-rw-rw-r--   1 ksuo ksuo  20322 Jul 31  2022 .clang-format
-rw-rw-r--   1 ksuo ksuo     59 Jul 31  2022 .cocciconfig
-rw-rw-r--   1 ksuo ksuo 205832 Aug 24 20:48 .config
-rw-rw-r--   1 ksuo ksuo 372907 Aug 24 20:39 .config.old
-rw-rw-r--   1 ksuo ksuo    496 Jul 31  2022 COPYING
-rw-rw-r--   1 ksuo ksuo 101376 Jul 31  2022 CREDITS
```

----------[Possible Error] ----------

For some distributions of Ubuntu, you may see errors like this when compiling:

No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509_certificate_list'.

[Solution]
Edit the .config file and change the value of CONFIG_SYSTEM_TRUSTED_KEYS to null
$ vim .config

Before:
CONFIG_SYSTEM_TRUSTED_KEYRING=y
CONFIG_SYSTEM_TRUSTED_KEYS="debian/canonical-certs.pem"

After:
CONFIG_SYSTEM_TRUSTED_KEYRING=y
CONFIG_SYSTEM_TRUSTED_KEYS=""

If the CONFIG_SYSTEM_REVOCATION_KEYS="debian/canonical-revoked-certs.pem" is not null, please also set it as null as:
CONFIG_SYSTEM_REVOCATION_KEYS=""

Then recompile the kernel using the make command.

Step 3: Compile the kernel

Please keep in mind that the compiling might take 0.5-1 hour, depending on your machine hardware specs and speed. For instance, in my 2021 Macbook, it takes about 20 mins.

In LINUX_SOURCE, compile to create a compressed kernel image:
$ make
If your VM has more than 1 core, we suggest you use "make -j N" to accelerate the compiling. Here, N denotes the number of CPUs on your VM.

To compile kernel modules:
$ make modules
You can use "make modules -j N" to accelerate the compiling. Here N denotes the number of CPUs on your VM.

Step 4: Install the kernel

Install kernel modules (become a root user, use the su command):
$ sudo make modules_install

Install the kernel:
$ sudo make install

If you are using Ubuntu, you need to create an init ramdisk manually:
$ sudo mkinitramfs -o /boot/initrd.img-5.19.0
$ sudo update-initramfs -c -k 5.19.0

The kernel image and other related files have been installed into the /boot directory. You can check it from /boot/grub/grub.cfg. Linux will boot by default using the 1st menu item.

Step 5: Modify grub configuration file

If you are using Ubuntu: change the grub configuration file:
$ sudo vim /etc/default/grub

Make the following changes:
GRUB_DEFAULT=0
GRUB_TIMEOUT=10

If your GRUB_HIDDEN_TIMEOUT_QUIET=true, change it to GRUB_HIDDEN_TIMEOUT_QUIET =false. If there is no GRUB_HIDDEN_TIMEOUT_QUIET, just ignore it.

If your GRUB_TIMEOUT_STYLE=hidden, change it to GRUB_TIMEOUT_STYLE=menu. If there is no GRUB_TIMEOUT_STYLE, just ignore it.

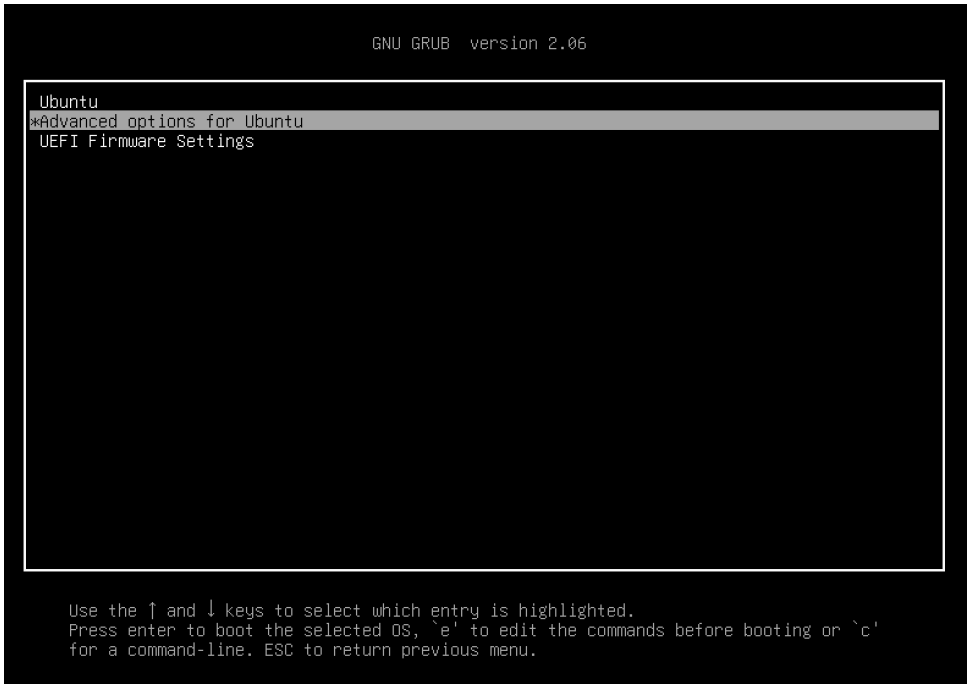
Then, update the grub entry:
$ sudo update-grub2

Step 6: Reboot your VM

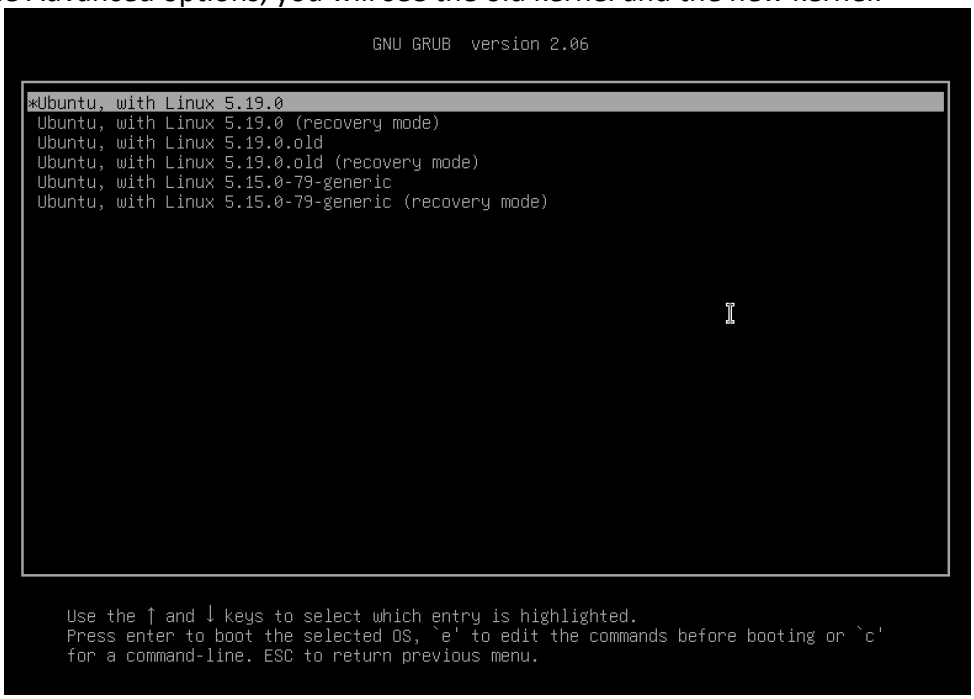
Reboot to the new kernel:
$ sudo reboot

(If you are using university VM, ignore the following steps. It only works for local VM)
Immediately after the BIOS/UEFI splash screen during boot, with BIOS, quickly press and hold the Shift key, which will bring up the GNU GRUB menu. (If you see the Ubuntu logo, you've missed the point where you can enter the GRUB menu.)

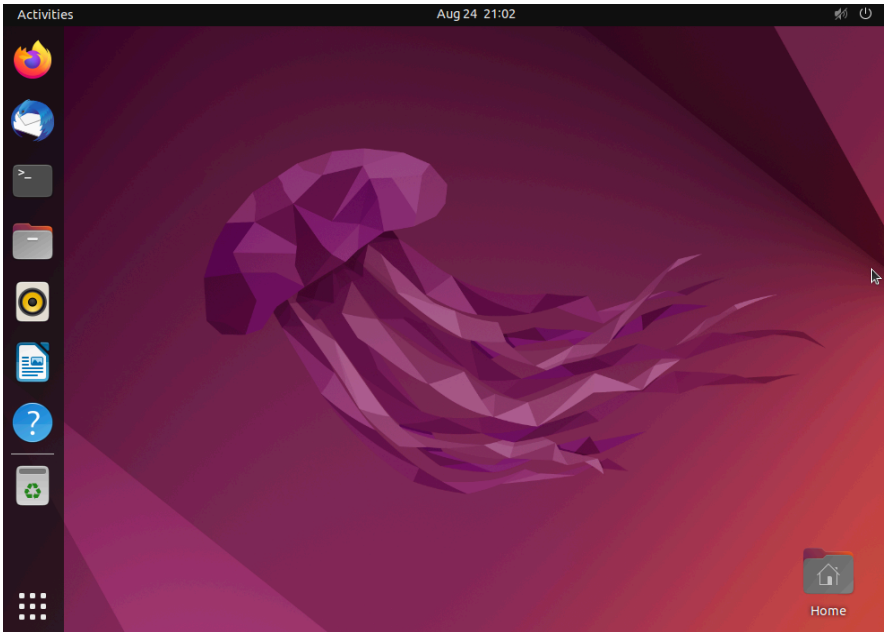
Select the following option:



Under the Advanced options, you will see the old kernel and the new kernel:



Select the new kernel and wait for a few seconds, you will enter the VM with new kernel:

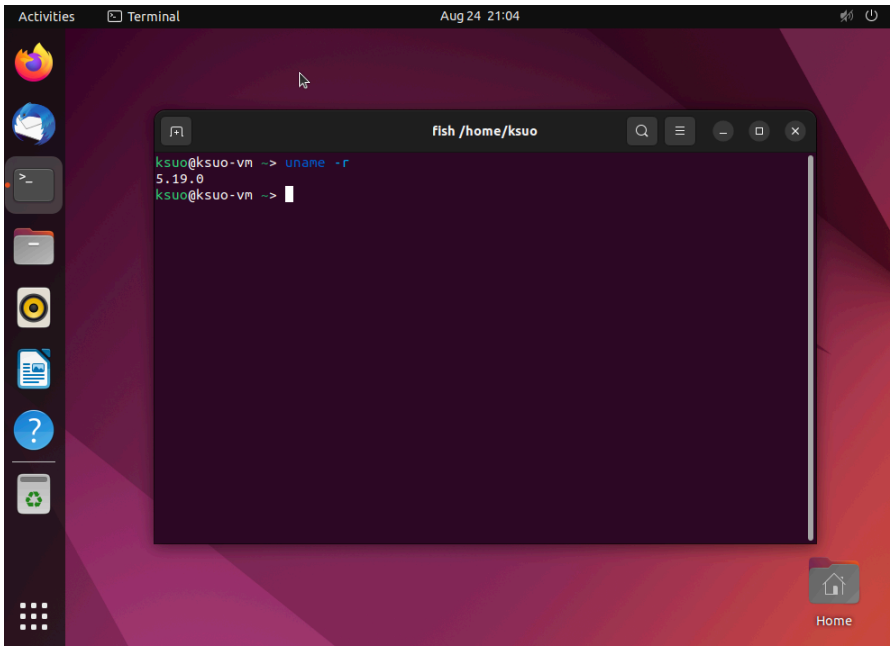


After boot, check if you have the new kernel:

$ uname -r

5.19.0

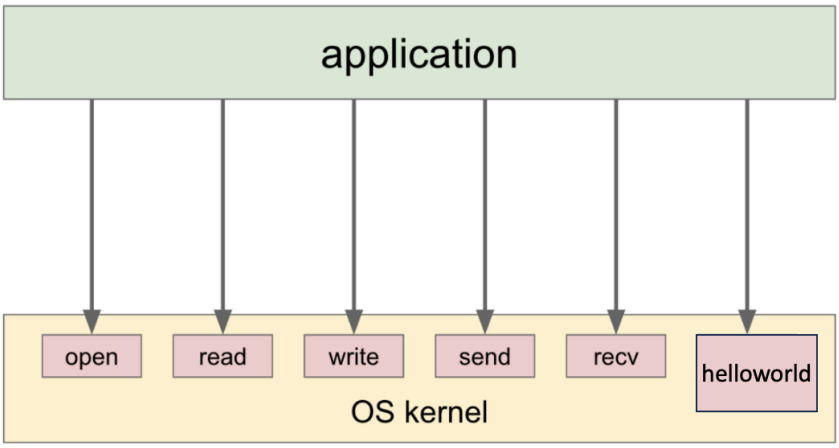
Here is the screenshot of my new kernel:



Submission of Part A:

Please submit the screenshot of $uname -r

Part B: Add a new system call into the Linux kernel (50 points) --- Please work this on VMs in KSU cloud, https://cseview.kennesaw.edu/



In this assignment, we add a simple system call helloworld to the Linux kernel. The system call prints out a hello world message to the syslog. You need to implement the system call in the kernel and write a user-level program to test your new system call.

Please note that the following only works on x86 VM, not on ARM VM. All VMs in KSU datacenter are x86 VMs.

Step 1: Check the available system call number

$ sudo find / -name unistd_64.h

```
fish  /home/administrator/ksuo
administrator@CS3502-14 ~/ksuo> sudo find / -name unistd_64.h
[sudo] password for administrator:
No logon servers
/home/administrator/linux-5.19/arch/x86/include/generated/uapi/asm/unistd_64.h
/home/administrator/linux-5.19/tools/arch/x86/include/uapi/asm/unistd_64.h
find: '/run/user/1000/gvfs': Permission denied
/usr/include/x86_64-linux-gnu/asm/unistd_64.h
/usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/generated/uapi/asm/unistd_64.h
/usr/src/linux-hwe-5.4-headers-5.4.0-150/arch/sh/include/uapi/asm/unistd_64.h
/usr/src/linux-hwe-5.4-headers-5.4.0-149/arch/sh/include/uapi/asm/unistd_64.h
/usr/src/linux-headers-5.4.0-149-generic/arch/x86/include/generated/uapi/asm/unistd_64.h
/usr/src/linux-hwe-5.4-headers-5.4.0-60/arch/sh/include/uapi/asm/unistd_64.h
```

As my current Linux version is 5.4.0-150-generic, so I select the above one.

Then, use cat command to print the file content:

$ sudo cat /usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/generated/uapi/asm/unistd_64.h

```
fish  /home/administrator/ksuo
administrator@CS3502-14 ~/ksuo> sudo cat /usr/src/linux-headers-5.4.0-150-generic/arch/x86/include/generated/uapi/asm/unistd_64.h
#ifndef _ASM_X86_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H 1

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
```

...

```
#define __NR_pkey_alloc 330
#define __NR_pkey_free 331
#define __NR_statx 332
#define __NR_io_pgetevents 333
#define __NR_rseq 334
#define __NR_pidfd_send_signal 424
#define __NR_io_uring_setup 425
#define __NR_io_uring_enter 426
#define __NR_io_uring_register 427
#define __NR_open_tree 428
```

As we can see No.335 is not used yet, so we select 335 as our new system call.

Step 2: Create a kernel module syscall

$ vim syscall.c

Please note that the following only works in 64-bit system. The content of syscall.c is as follows:

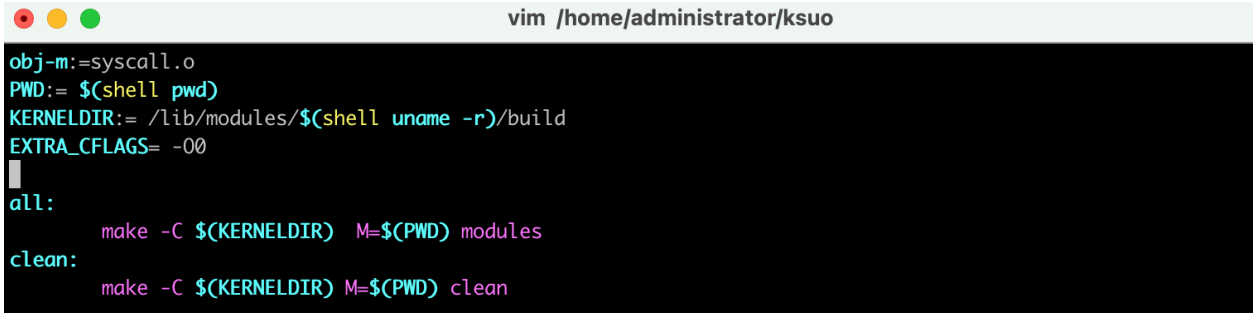
```
vim  /home/administrator/ksuo
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/unistd.h>
#include <linux/sched.h>
#include <linux/time.h>
#include <asm/uaccess.h>
#include <linux/kallsyms.h>

#define __NR_syscall 335        /* system call: 335 */
unsigned long * sys_call_table;

unsigned int clear_and_return_cr0(void);
void setback_cr0(unsigned int val);
static int sys_mycall(void);

int orig_cr0;
unsigned long *sys_call_table = 0;
static int (*anything_saved)(void);
/*
 * set cr0 register No.17 as 0
 */
unsigned int clear_and_return_cr0(void)
{
        unsigned int cr0 = 0;
        unsigned int ret;
```

```
    /* The former is used in 32-bit systems. The latter is used in 64-bit systems, this system is 64-bit */
        //asm volatile ("movl %%cr0, %%eax" : "=a"(cr0));
        asm volatile ("movq %%cr0, %%rax" : "=a"(cr0));

        ret = cr0;
        cr0 &= 0xfffefffff;

        //asm volatile ("movl %%eax, %%cr0" :: "a"(cr0));
        asm volatile ("movq %%rax, %%cr0" :: "a"(cr0));
        return ret;
}


/* Read the value of val to the rax register, and then put the value of the rax register into cr0 */
void setback_cr0(unsigned int val)
{

        //asm volatile ("movl %%eax, %%cr0" :: "a"(val));
        asm volatile ("movq %%rax, %%cr0" :: "a"(val));
}


/* Our system call is here */
static int sys_mycall(void)
{
        int ret = 12345;
        printk("Here is my syscall in OS kerenl!\n");
        return ret;
}


static int __init init_addsyscall(void)
{
        printk("My syscall is starting。。。\n");
        sys_call_table = (unsigned long *)kallsyms_lookup_name("sys_call_table");
        printk("sys_call_table: 0x%p\n", sys_call_table);
        anything_saved = (int(*)(void))(sys_call_table[__NR_syscall]);
        orig_cr0 = clear_and_return_cr0();
        sys_call_table[__NR_syscall] = (unsigned long)&sys_mycall;
        setback_cr0(orig_cr0);
        return 0;
}


static void __exit exit_addsyscall(void)
{
        orig_cr0 = clear_and_return_cr0();
    sys_call_table[__NR_syscall] = (unsigned long)anything_saved;
    setback_cr0(orig_cr0);
        printk("My syscall exit....\n");
}

module_init(init_addsyscall);
module_exit(exit_addsyscall);
MODULE_LICENSE("GPL");
```

Source file: https://github.com/kevinsuo/CS3502/blob/master/syscall.c

Step 3: Define the Makefile

$ vim Makefile



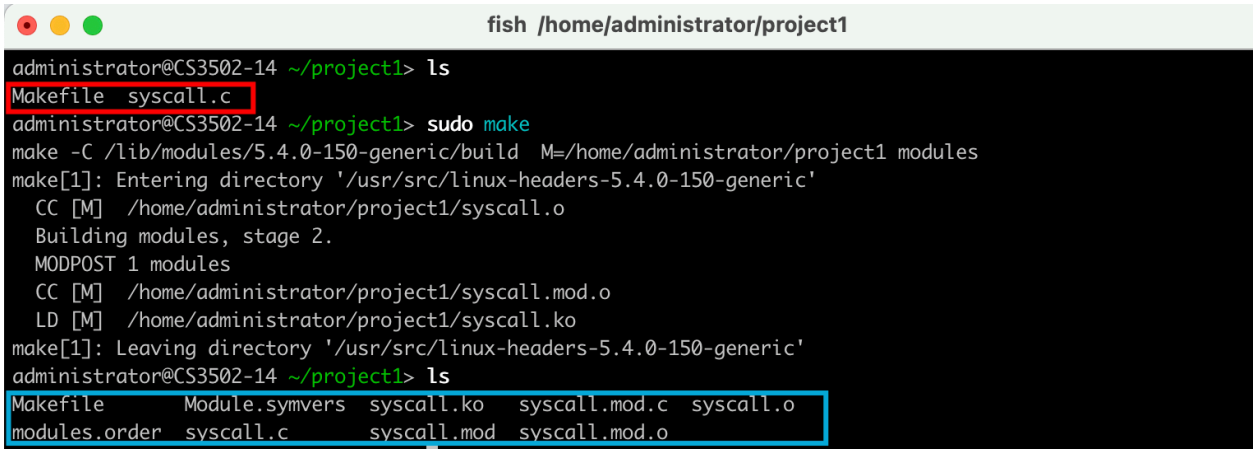
Source file: https://github.com/kevinsuo/CS3502/blob/master/Makefile

Step 4: Compile and enable the module syscall

Under the directory with syscall.c and Makefile, run the make command to compile them.

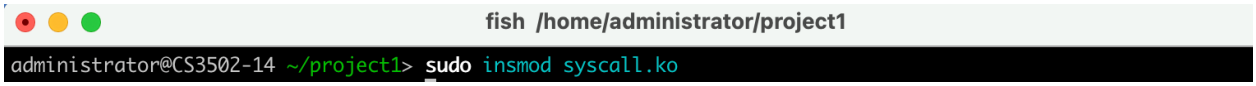
$ sudo make

As the following figure shows, the red and blue parts are before and after the compiling.



Insert kernel modules into the Linux kernel

$ sudo insmod syscall.ko



You can use the $ lsmod to check the enabled modules:

```
fish  /home/administrator/project1
administrator@CS3502-14 ~/project1> lsmod
Module                  Size  Used by
syscall                16384  0
test                   16384  0
ipt_REJECT             16384  2
nf_reject_ipv4         16384  1 ipt_REJECT
xt_tcpudp              20480  2
iptable_filter         16384  1
bpfilter               24576  0
```

If you want to disable one module, try $ sudo rmmod [mod-name].ko

Step 5: write a test program to test your system call

Create a test program test.c

```
vim  /home/administrator/ksuo
#include <syscall.h>
#include <stdio.h>

int main(void)
{
        /* call system call No.335 */
        printf("%d\n",syscall(335));
        return 0;
}
```

Compile the user level program:

$ gcc test.c -o test.o

Test the new system call by running:

$ sudo ./test.o

The test program will call the new system call and output “Here is my syscall in OS kerenl!” message at the tail of the output of dmesg.

$ dmesg | grep my

Here is the screenshot on my VM:

```
fish  /home/administrator/project1                    ⌥⌘1
administrator@CS3502-14 ~/project1> sudo ./test.o
12345
administrator@CS3502-14 ~/project1> dmesg | grep my
[178116.531246] Here is my syscall in OS kerenl!
administrator@CS3502-14 ~/project1>
```

Submission of Part B:

Please submit the screenshot of $ dmesg | grep “by Your name”, and your system call message should be “Here is my syscall in the OS kernel by [Your Name]!”

For instance, a student named Sisi should upload a screenshot like: