

Kennesaw State University

CSE 3502 Operating Systems

Project 1 - System call

Instructor: Kun Suo
Points Possible: 100

Assignments

Assignment 0: Build the Linux kernel (50 points)

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 5.1 and extract the source:

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.19.tar.gz
$ tar xvfz 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

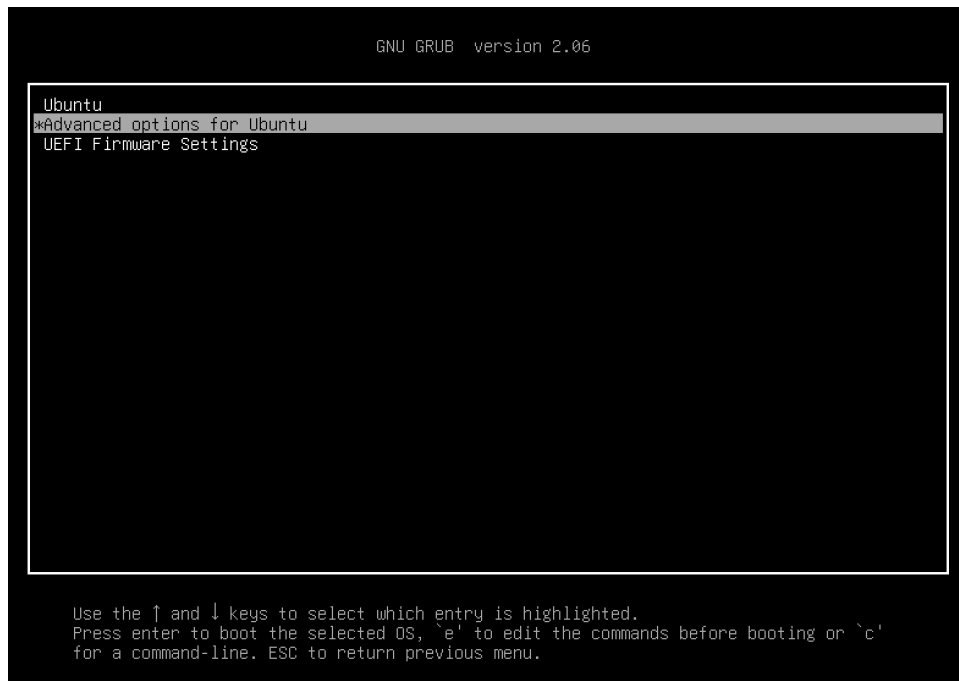
Reboot to the new kernel:

```
$ sudo reboot
```

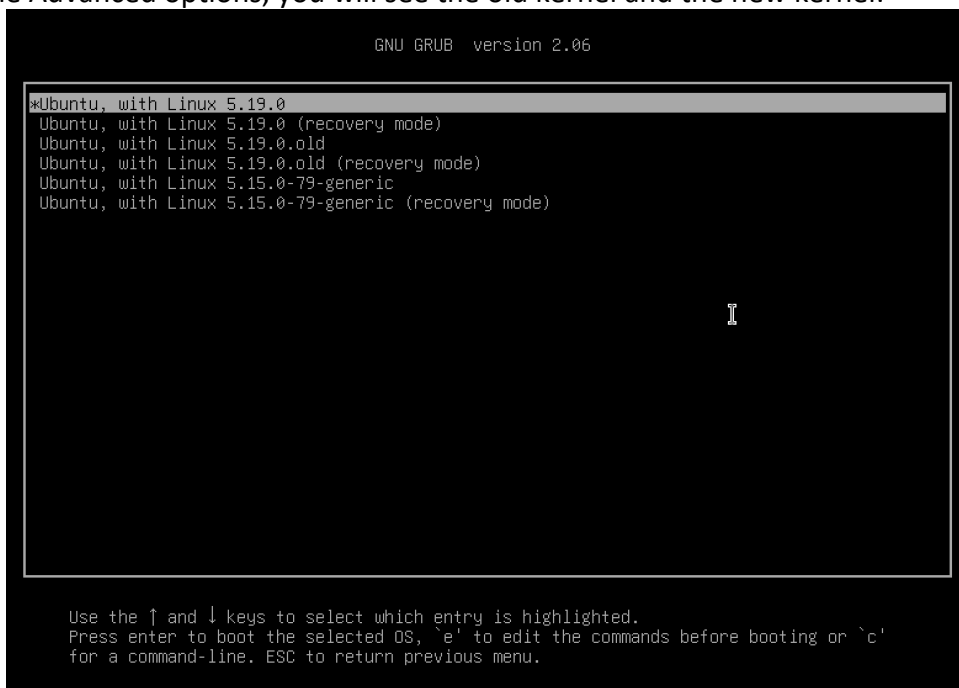
(If you are using university VM, ignore the following steps. It only works for local VM)

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.)

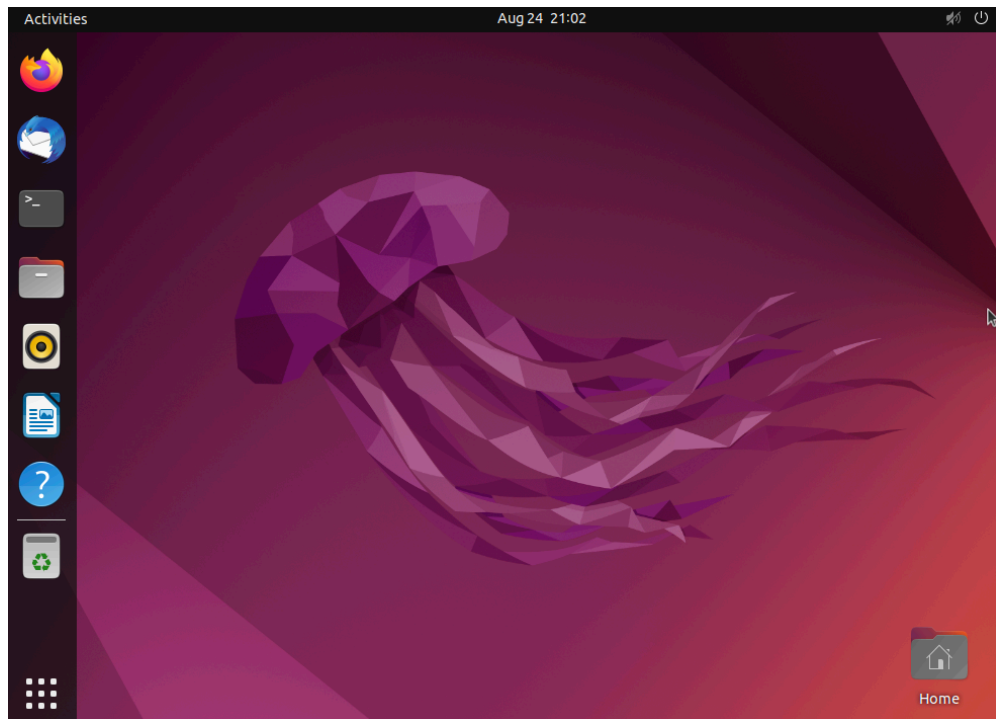
Select the following option:



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:

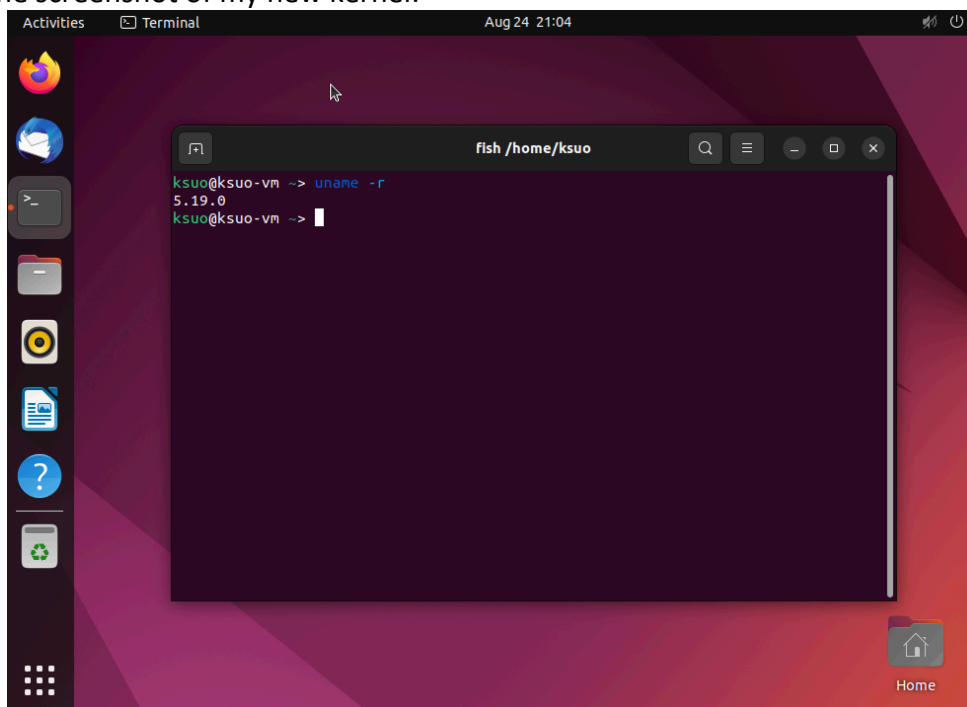


After boot, check if you have the new kernel:

```
$ uname -r
```

5.19.0

Here is the screenshot of my new kernel:



Submission of assignment 0:

Please submit the screenshot of `$uname -r`

Assignment 1: Add a new system call into the Linux kernel (50 points)

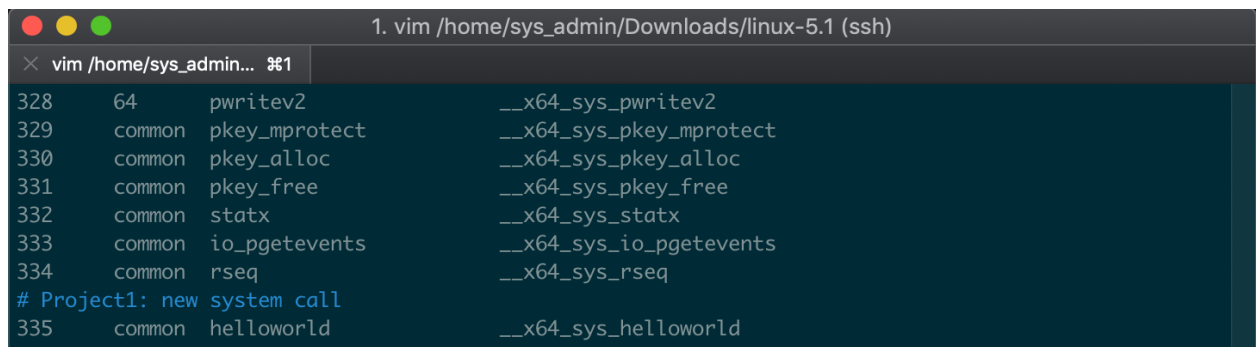
In this assignment, we add a simple system call `helloworld` to the Linux kernel. The system call prints out a hello world message to the `syslog`. You need to implement the system call in the kernel and write a user-level program to test your new system call. Go to the kernel source code folder `linux-5.19`.

```
$ cd linux-5.19
```

(1) If your VM is x86 architecture:

Step 1: register your system call

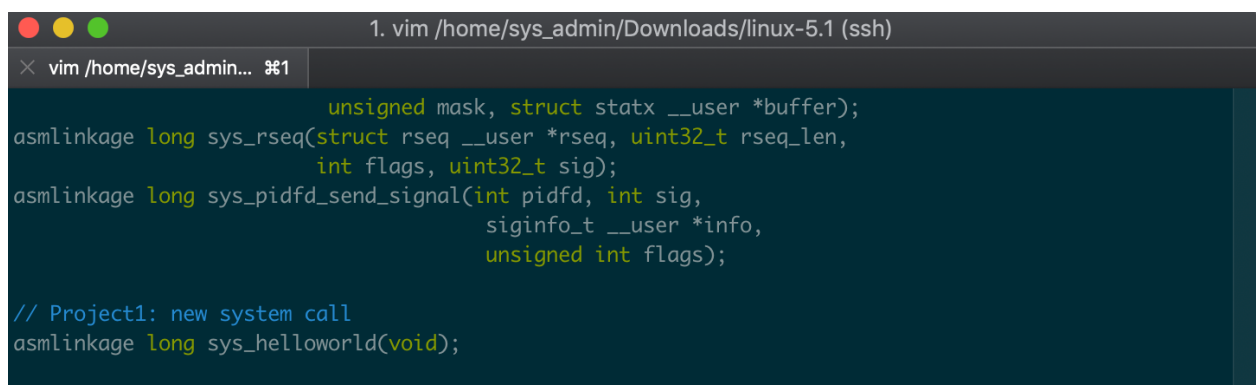
```
$ vim arch/x86/entry/syscalls/syscall_64.tbl
```



```
328 64 pwritev2 __x64_sys_pwritev2
329 common pkey_mprotect __x64_sys_pkey_mprotect
330 common pkey_alloc __x64_sys_pkey_alloc
331 common pkey_free __x64_sys_pkey_free
332 common statx __x64_sys_statx
333 common io_pgetevents __x64_sys_io_pgetevents
334 common rseq __x64_sys_rseq
# Project1: new system call
335 common helloworld __x64_sys_helloworld
```

Step 2: declare your system call in the header file

```
$ vim include/linux/syscalls.h
```



```
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,
                        int flags, uint32_t sig);
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,
                                     signinfo_t __user *info,
                                     unsigned int flags);

// Project1: new system call
asmlinkage long sys_helloworld(void);
```

Step 3: implement your system call

```
$ vim kernel/sys.c
```

```
2. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
vim /home/sys_admin... %1
do_sysinfo(&val);

if (copy_to_user(info, &val, sizeof(struct sysinfo)))
    return -EFAULT;

return 0;
}

// Project1: new system call
SYSCALL_DEFINE0(helloworld)
{
    printk("helloworld");
    return 0;
}
```

Repeat step 3 and 4 in assignment 0 to re-compile the kernel and reboot to the new kernel.

Step 4: write a user-level program to test your system call

Go to your home directory and create a test program test_syscall.c

```
2. vim /home/sys_admin/Downloads/test (ssh)
vim /home/sys_admin... %1
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define __NR_helloworld 335

int main(int argc, char *argv[])
{
    syscall(__NR_helloworld);
    return 0;
}
```

Compile the user level program:

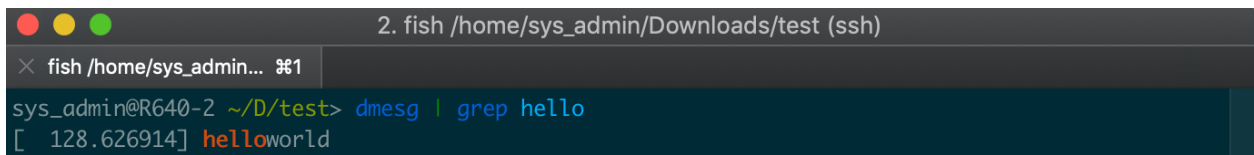
```
$ gcc test_syscall.c -o test_syscall
```

Test the new system call by running:

```
$ sudo ./test_syscall
```

The test program will call the new system call and output a helloworld message at the tail of the output of dmesg.


```
$ dmesg | grep hello
```



```
2. fish /home/sys_admin/Downloads/test (ssh)
fish /home/sys_admin... %1
sys_admin@R640-2 ~/D/test> dmesg | grep hello
[ 128.626914] helloworld
```

- (2) If your VM is arm architecture (i.e., Macbook with Apple Silicon), the tutorial is currently unavailable due to limited hardware resources and support. Please Google search how to do it step by step.

Submission of assignment 1:

Please have two copies of kernel source code: 1) the original kernel source code without any modification; 2) the kernel source code you modified. You can define the folder name based on your need. Here I use *linux-5.19* as the original source code without modification and *linux-5.19-modified* as the source code I worked on.

Instruction:

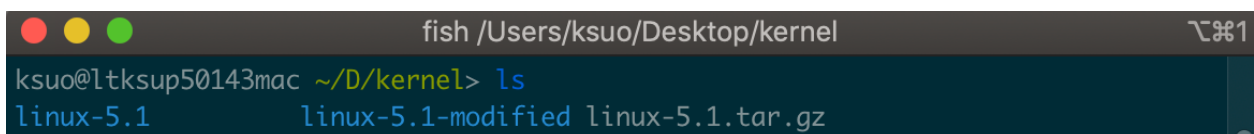
Change your linux-5.19 folder name into linux-5.19-modified.

```
$ mv linux-5.19 linux-5.19-modified
```

Unzip the source code again

```
$ tar xvfz linux-5.19.tar.gz
```

Then you should have two folders of the Linux source code.

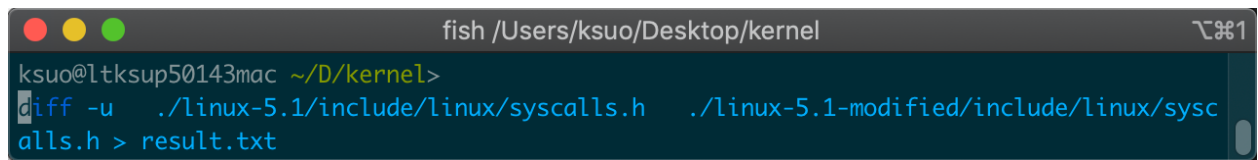


```
fish /Users/ksuo/Desktop/kernel
ksuo@ltsup50143mac ~/D/kernel> ls
linux-5.1      linux-5.1-modified linux-5.1.tar.gz
```

Please use diff command to highlight your modification (Here the original_file.c refers the file or file path of the original file source code; the modified_file.c refers the file or file path of the file source code you have modified):

```
$ diff -u original_file.c modified_file.c > result.txt
```

For example, to show the difference between file include/linux/syscalls.h, just use the command below:

A terminal window with a dark background and light text. The title bar at the top reads 'fish /Users/ksuo/Desktop/kernel'. The terminal content shows a prompt 'ksuo@ltkup50143mac ~/D/kernel>' followed by the command 'diff -u ./linux-5.1/include/linux/syscalls.h ./linux-5.1-modified/include/linux/syscalls.h > result.txt'.

```
fish /Users/ksuo/Desktop/kernel
ksuo@ltkup50143mac ~/D/kernel>
diff -u ./linux-5.1/include/linux/syscalls.h ./linux-5.1-modified/include/linux/syscalls.h > result.txt
```

For the kernel code, please do not add the entire kernel source code. Just add your modification code, e.g., result1.txt, result2.txt, result3.txt, ...

Please submit the screenshot of `$ dmesg | grep hello` and `result1.txt`, `result2.txt`, `result3.txt`, ...