Project 1: Adding a System call to the Linux Kernel

System calls provide an interface to the services made available by an operating system. In this project, we will incorporate a new system call into the kernel, thereby expanding the functionality of the operating system.

Getting Started

First of all, a virtual machine for linux is needed since my operating system is Windows. I choose VMware Workstation for convenience. Then I downloaded ubuntu-16.04.1 for linux, and its kernel version is 4.4.0.

However, the instructions on the book is suitable for linux-2.x. The difference causes many obstacles since files or programs change greatly in new kernel versions. As a result, I found tutorials online for 4.x, which may be a little different according to book.

For preparation, the following instructions are executed to get packages needed in our project.

apt-get install vim
apt-get install libncurses5-dev
apt-get install libssl-dev

Go to the website *www.kernel.org* to download the latest kernel 4.8.1.
Now we can get started.

Adding a System Call to the Kernel

We move to the directory of */home/username/Downloads/linux-4.8.1*. It is where the kernel package are download.

We unzip it by typing the command
tar -xvf linux-4.8.1.tar.xz

We need to access the system call table to add one for our new system call.
vim arch/x86/entry/syscalls/syscall_64.tbl

As is seen in Figure 1, we add a function called *sys_hello* which number is 329. Notice that the original system calls shouldn't be substituted in case errors occur.

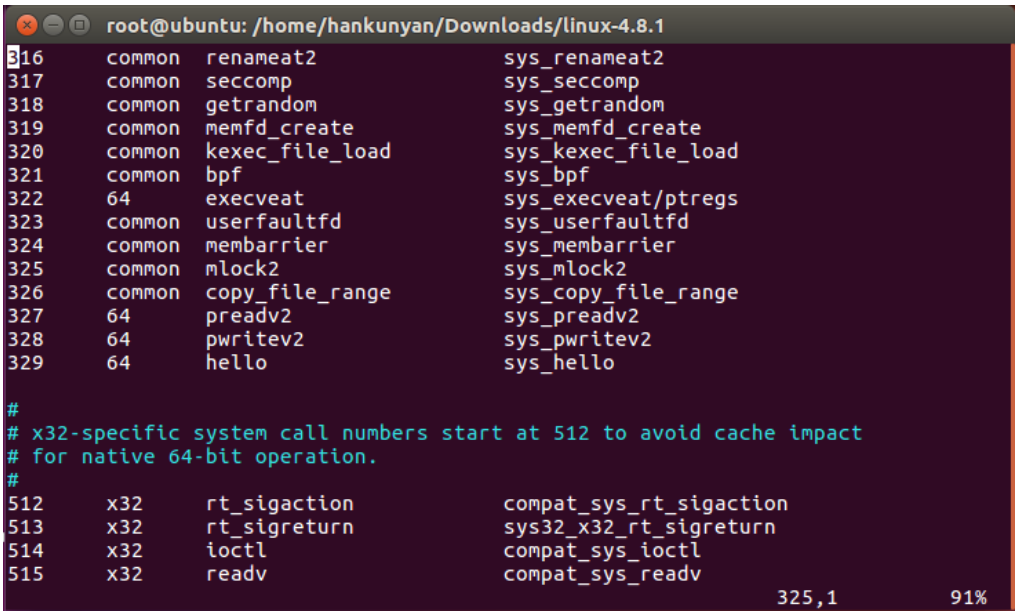

Figure 1: Add a system call number

Then we should make our new system call function declaration.
vim include/linux/syscalls.h
We add our system call function *sys_hello* in the end. It is shown in Figure 2.

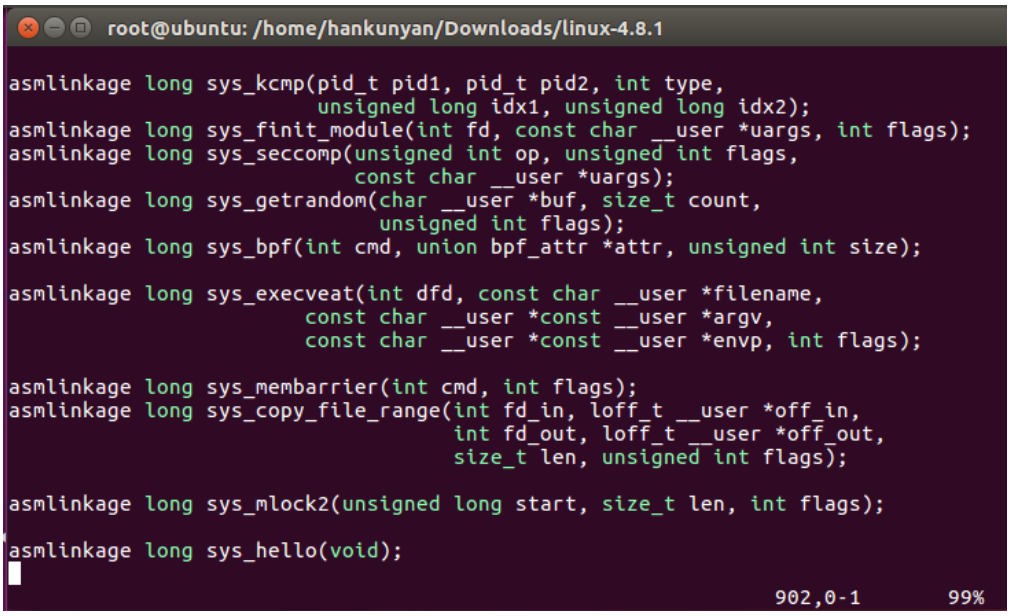

Figure 2: Add the function declaration of the added syscall

Now the only task left is function implementation. We have two choices. The first method is creating a new directory to add the new file about the function under the kernel. However, the Makefile should also be modified accordingly. The second solution is relatively easier. We can add the function in *sys.c* in the kernel directory. Makefile needn't to be changed.

I take the second method for the sake of convenience.

vim kernel/sys.c

Similarly, we append the function *sys_hello* in the end. You can check it in Figure 3.

Figure 3: Add the function implementation of the added syscall

Till now, we have modified the kernel and add our own system call *sys_hello* in it.

Building a New Kernel

Then the following steps are quite easy, but really take time.

sudo make menuconfig
sudo make
sudo make modules_install
sudo make install

We execute above instructions one by one. Notice that the first instruction can be replaced by *sudo make localmodconfig*. This instruction will help *make* process only compile a little part of the kernel which has been changed. It will cut the compiling time from 2.5 hours to only half an hour and it really works.

After that, we reboot the system to check our system call.

Using the System Call From a User Program

Since the modified kernel has already installed, it will support the newly defined system call. Now we can write a simple C program to check it.

```c
#include <unistd.h>

int main()
{
        syscall(329);
        return 0;
}
```

Figure 4: Test function

Then we run this program and use *dmesg* instruction to review system logs to see whether the system call has been executed.

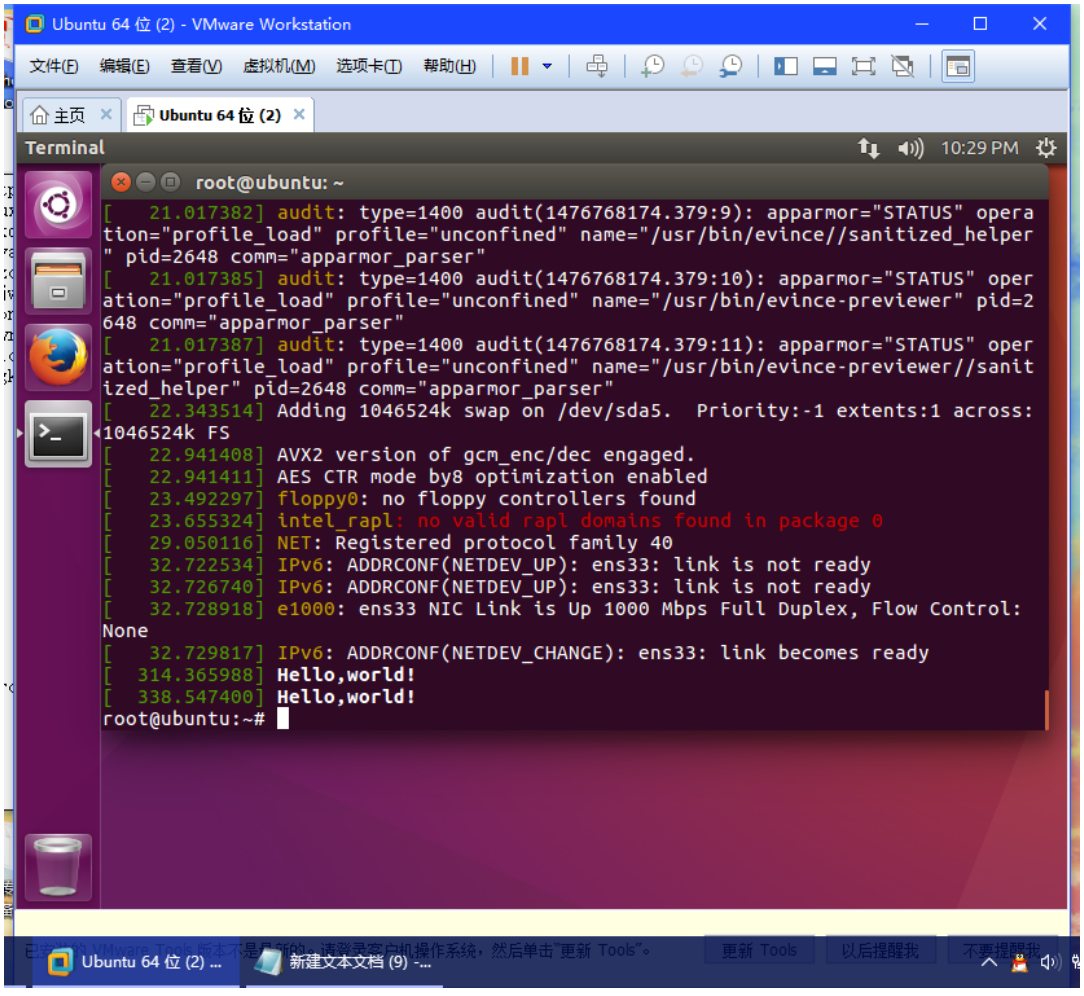

Figure 5: The result

At first the message won't appear and I wonder why. After searching the Internet I finally find the reason. Since I didn't add KERN_EMERG when I used printk function. As a consequence, the message will be in the buffer and won't appear. That's why after running the program more than once, the message will appear together.

Now that we have add a new system call in the kernel, we can expand the functionality of the system call if needed.

Student name: Han Kunyan
Student No.:5140309534