**Kennesaw State University**

**CSE 3502 Operating Systems – Spring 2020**

**Project 1 - System call**

Instructor: Kun Suo

Points Possible: 100

**Assignments**

**Assignment 0: Build the Linux kernel (25 points)**

Create a virtual machine using VirtualBox on your machine. As the kernel compiling is pretty large, please make sure your VM has at least 4GB memory and 40GB storage.

**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

5.0

Then, download kernel 5.1 and extract the source:

$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.1.tar.gz

$ tar xvzf linux-5.1.tar.gz

We will refer LINUX\_SOURCE to the top directory of the kernel source. Go to the linux source code folder:

$ cd linux-5.1

**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 menuconfig

No changes to the default configuration are needed at this time. Press SAVE and OK, and then exit the configuration menu and a default config file will be generated. You can check .config using the following command under kernel folder. (<https://youtu.be/UyOGF4UOoR0>)

$ ls -al

**Step 3: Compile the kernel**

In LINUX\_SOURCE, compile to create a compressed kernel image:

$ make

You can 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-5.1.0

$ sudo update-initramfs -c -k 5.1.0

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

Then, update the grub entry:

$ sudo update-grub2

**Step 6: Reboot your VM**

Reboot to the new kernel:

$ sudo reboot

After boot, check if you have the new kernel:

$ uname -r

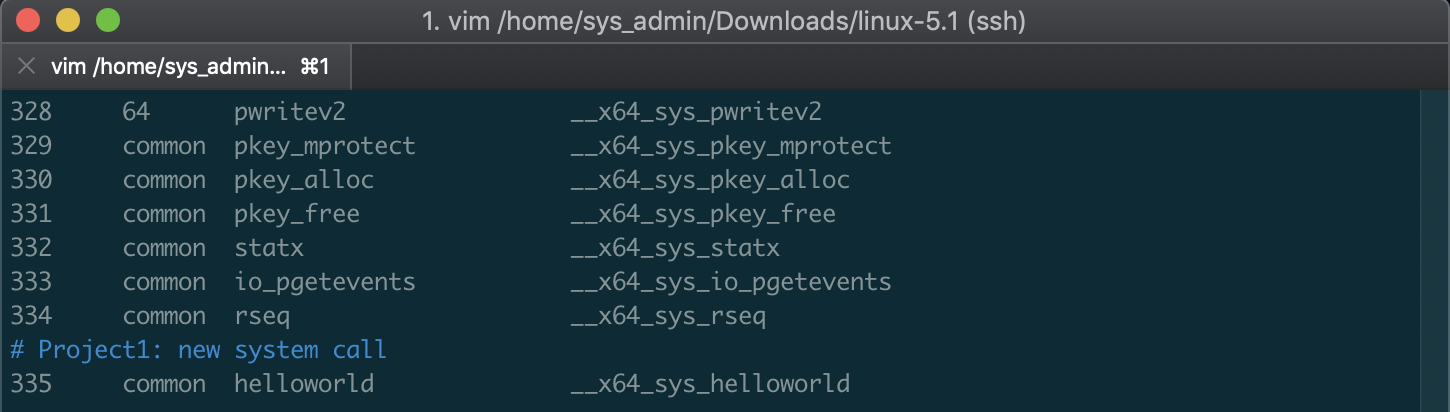
5.1.0

**Assignment 1: Add a new system call into the Linux kernel (25 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.

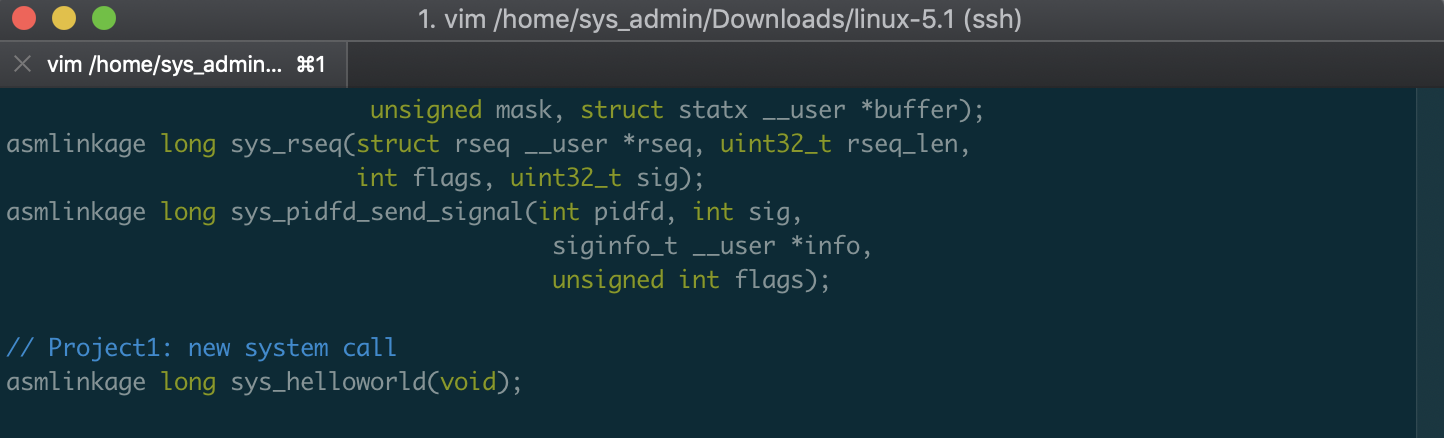
**Step 1: register your system call**

arch/x86/entry/syscalls/syscall\_64.tbl



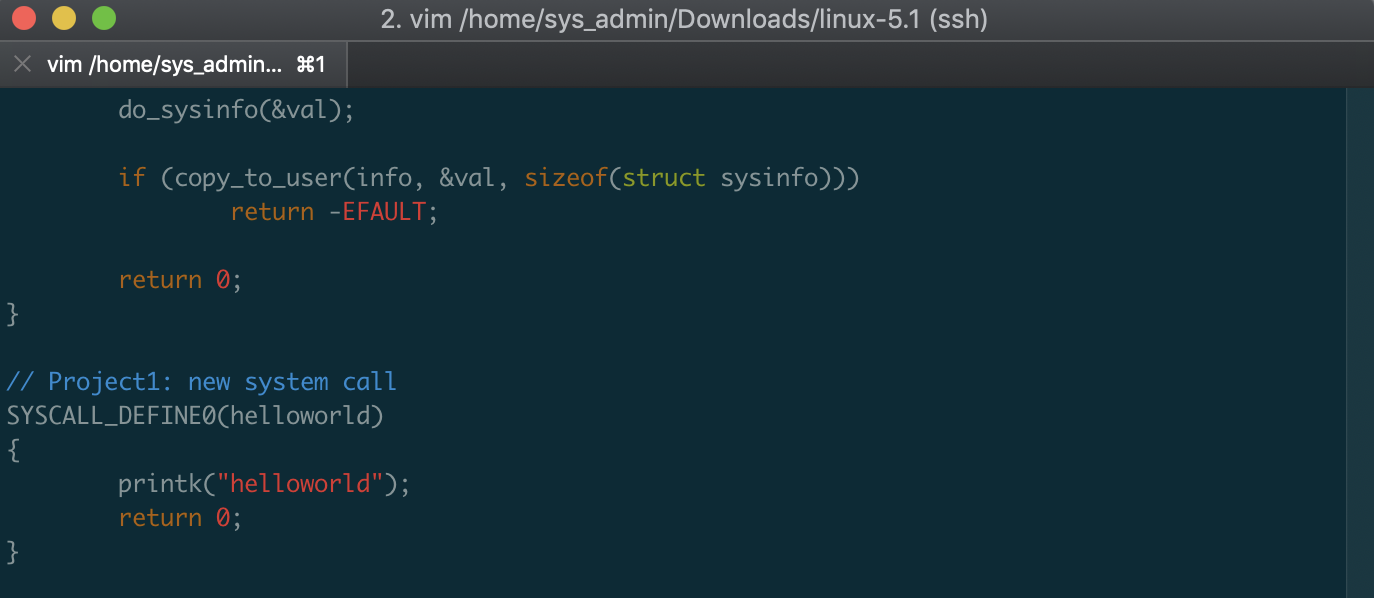
**Step 2: declare your system call in the header file**

include/linux/syscalls.h



**Step 3: implement your system call**

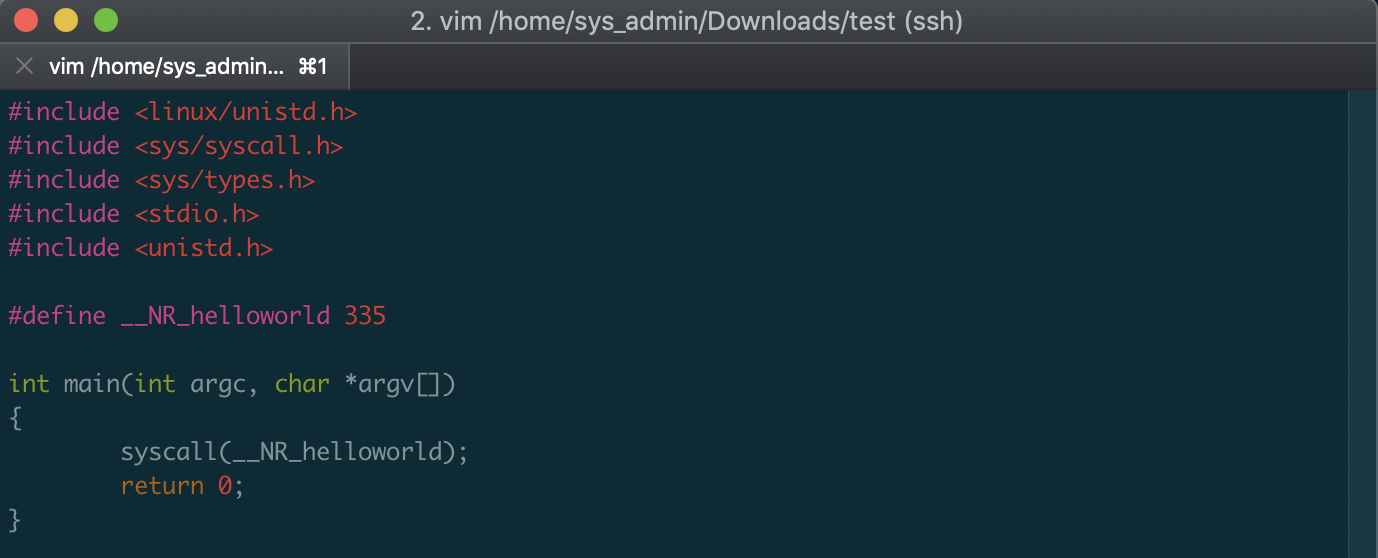
kernel/sys.c



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



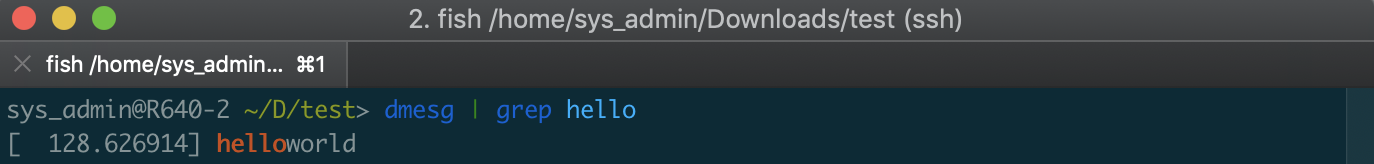
Compile the user level program:

$ gcc test\_syscall.c -o test\_syscall

Test the new system call by running:

$ ./test\_syscall

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



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-5.1* as the original source code without modification and *linux-5.1-modified* as the source code I worked on.



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 screen shot of a social media post

Description automatically generated

**Assignment 2: Extend your new system call to print out the calling process’s information (25 points)**

Follow the instructions we discussed above and implement another system call print\_self. This system call identifies the calling process at the user-level and print out various information of the process.

Implement the print\_self system call and print out the following information of the calling process:

• Process id, running state, and program name

• Start time and virtual runtime

• Its parent processes until init (first system process)

HINT: The macro ***current*** returns a pointer to the task\_struct of the current running process. The virtual runtime (vruntime) is located at the sched\_entity of task\_struct.

task\_struct data structure in the kernel: <https://elixir.bootlin.com/linux/v5.1/source/include/linux/sched.h#L585>

Please use diff command to highlight your modification:

$ diff -u original\_file.c modified\_file.c > result.txt

**Assignment 3: Extend your new system call to print out the information of an arbitrary process identified by its PID (25 points)**

Implement another system call print\_other to print the information for an arbitrary process. The system call takes a process pid as its argument and outputs the above information of this process.

HINT: You can start from the init process and iterate over all the processes. For each process, compare its pid with the target pid. If there is a match, return the pointer to this task\_struct.

A better approach is to use the pidhash table to look up the process in the process table. Linux provides many functions to find a task by its pid.

Please use diff command to highlight your modification:

$ diff -u original\_file.c modified\_file.c > result.txt

**Submitting Assignment**

Submit your assignment zip file through D2L using the appropriate assignment link.

Format: create three folders:

Assignment 1

Assignment 2

Assignment 3

In each folder, add the ***user space source code, kernel space source code, and screenshot of output*** inside.

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, …

In order to use diff command and get the modification file result.txt. Please have two copies of kernel source code: 1) the original kernel source code without any modification; 2) the kernel source code you modified, as the below figure shows:



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 screen shot of a social media post

Description automatically generated

Zip all the files and folders together into one zip file and name it as CS3502\_[your D2L user name]. Such as, **CS3502\_mahmed29.zip**.