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
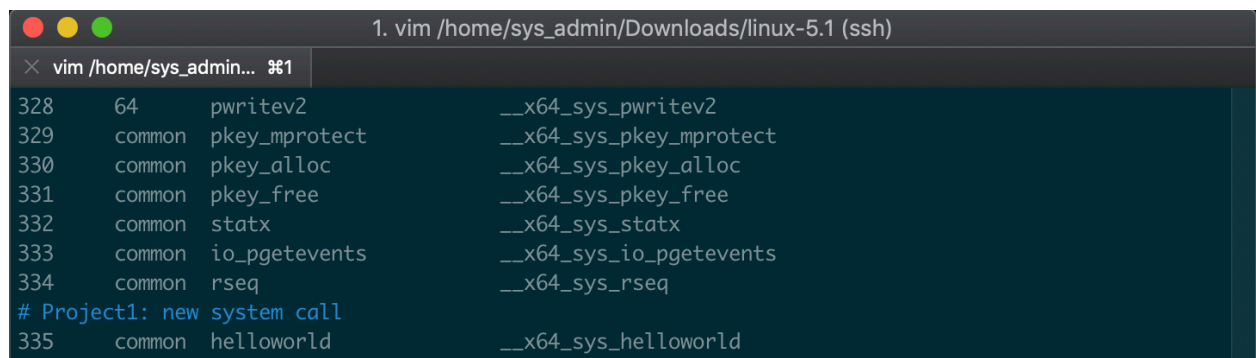
After boot, check if you have the new kernel:

$ 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.

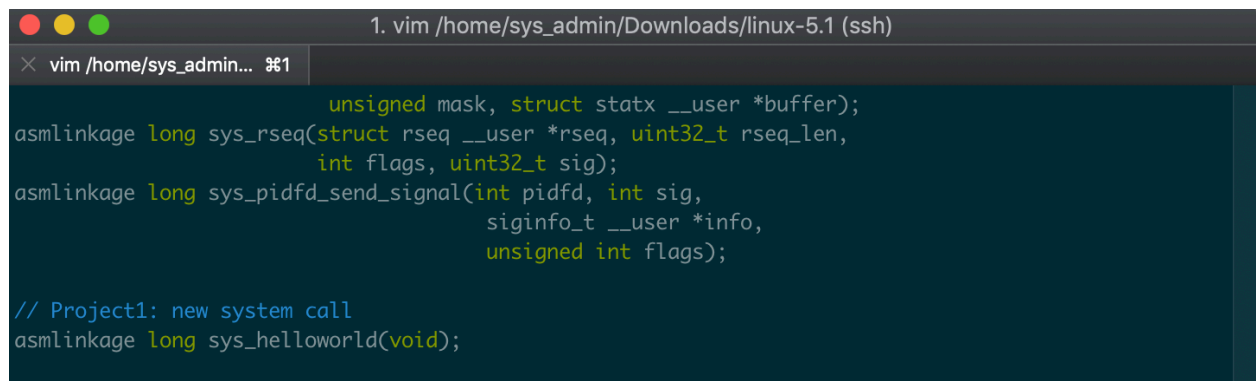**Step 1: register your system call**

arch/x86/entry/syscalls/syscall_64.tbl

```
                      1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
  ✕  vim /home/sys_admin...  ⌘1
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

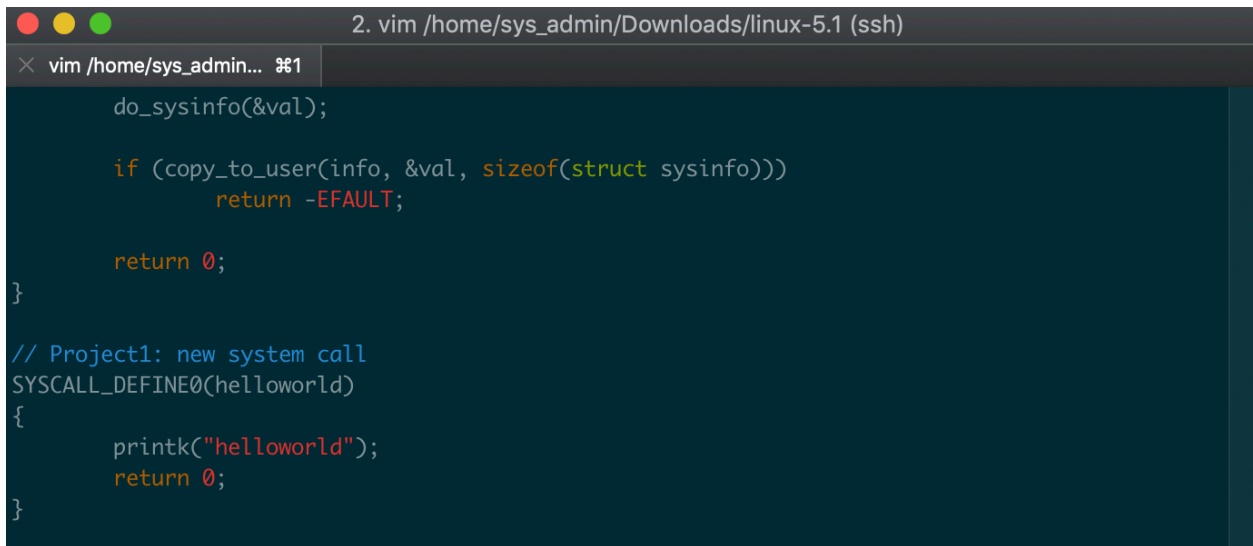include/linux/syscalls.h

```
                      1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
  ✕  vim /home/sys_admin...  ⌘1
                  unsigned mask, struct statx __user *buffer);
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,
                              siginfo_t __user *info,
                              unsigned int flags);

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

Step 3: implement your system call

kernel/sys.c

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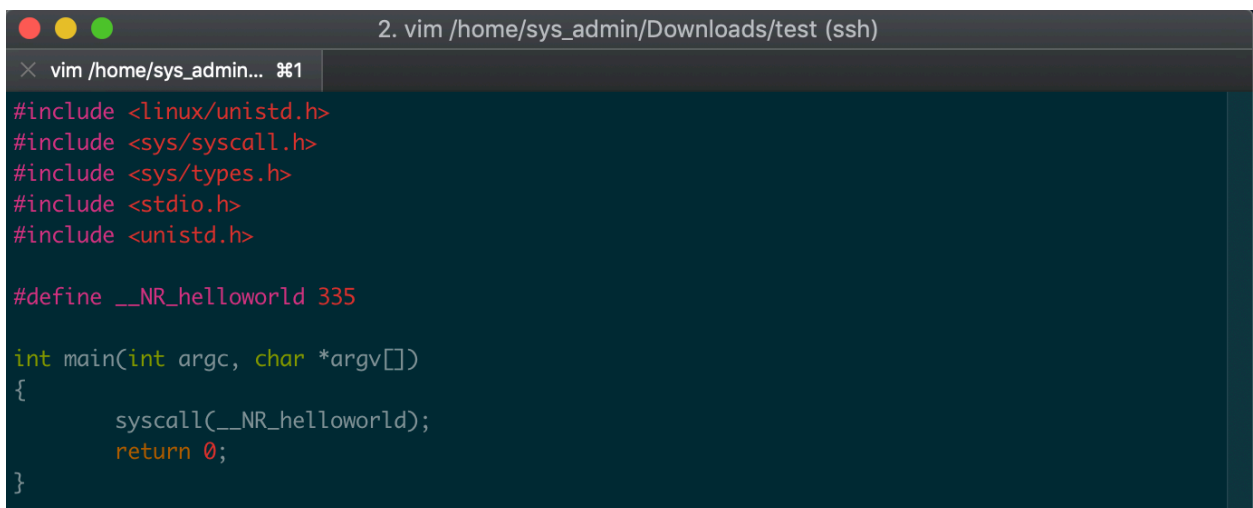
Repeat step 3 and 4 in assignment 0 to re-compile the kernel and reboot to the new kernel.

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

```
#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;
}
```
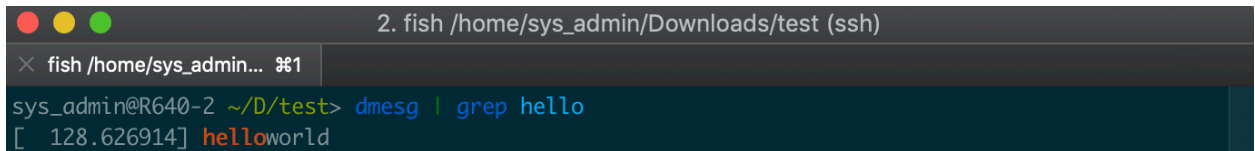
Compile the user level program:
$ gcc test_syscall.c -o test_syscall

Test the new system call by running:

$ sudo ./test_syscall

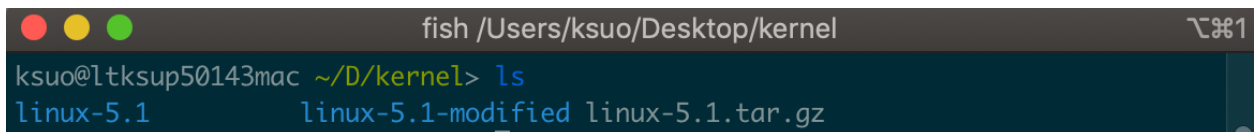The test program will call the new system call and output a helloworld message at the tail of the output of dmesg.

$ 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
```

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.
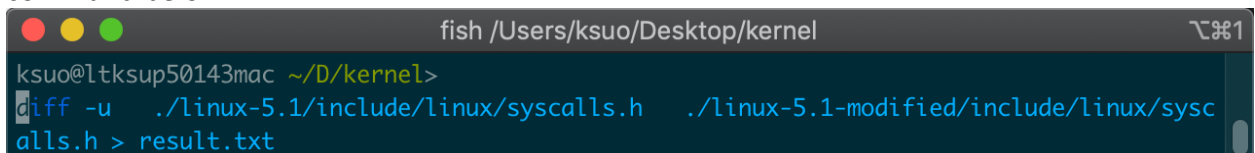
```
● ● ●                         fish /Users/ksuo/Desktop/kernel                    ⌥⌘1
ksuo@ltksup50143mac ~/D/kernel> ls
linux-5.1          linux-5.1-modified linux-5.1.tar.gz
```

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:

```
● ● ●                         fish /Users/ksuo/Desktop/kernel                    ⌥⌘1
ksuo@ltksup50143mac ~/D/kernel>
diff -u   ./linux-5.1/include/linux/syscalls.h   ./linux-5.1-modified/include/linux/sysc
alls.h > result.txt
```

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.
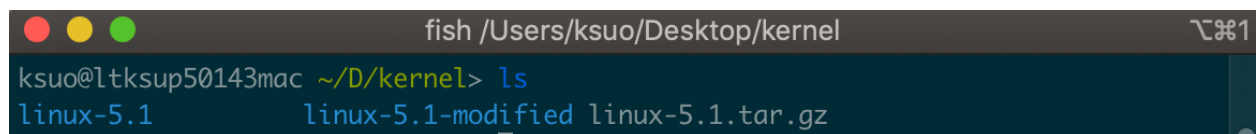
Format: create three folders:
Assignment 1
Assignment 2
Assignment 3

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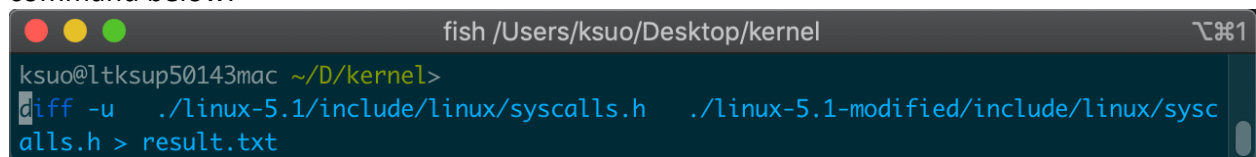
In order to use diff command and get the modification file result.txt. Please have two copies of kernel source code: 1) the original kernel source code without any modification; 2) the kernel source code you modified, as the below figure shows:

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:

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**.