Kennesaw State University

CSE 3502 Operating Systems - Fall 2019

Project 1 - System call

Instructor: Kun Suo
Points Possible: 100
Due date: 11:30 pm, Sep. 21, 2019

Assignments

Assignment 0: Build the Linux kernel (25 points)

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
```

Visit http://kernel.org and download the source code of your current running kernel. To obtain the version of your current kernel, type:

```
$ uname -r
4.9.185
```

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.

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 ESC to exit the configuration menu and a default config file will be generated. You can check .config using the following command under kernel folder.

```
$ 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 machine.

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 machine.

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

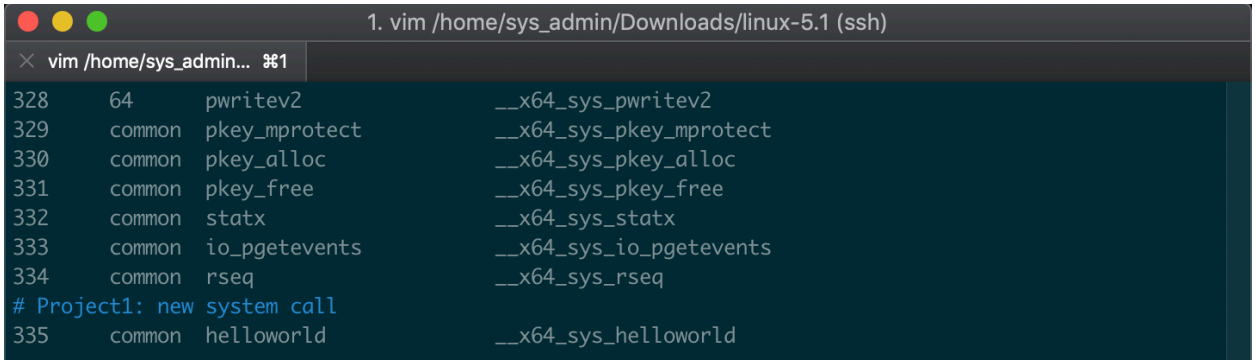
$ 5.1.0

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.

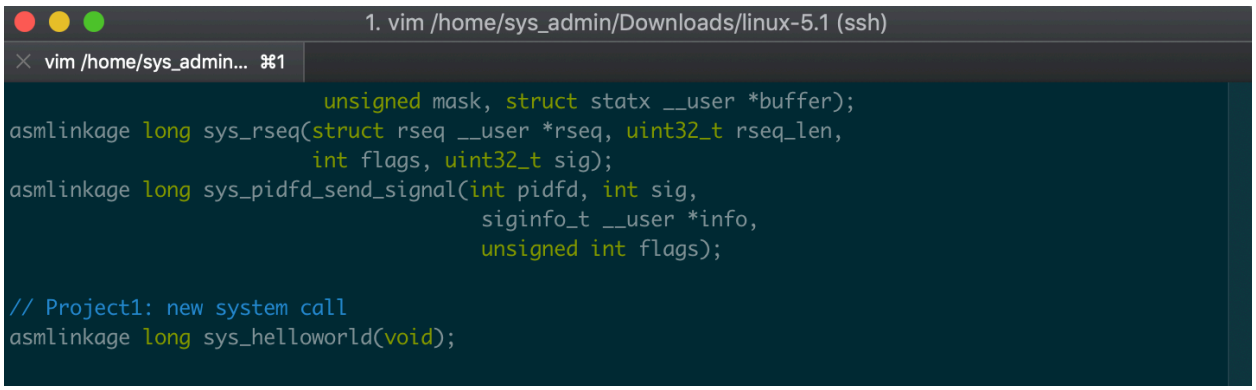
Step 1: register your system call

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

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;
}
```

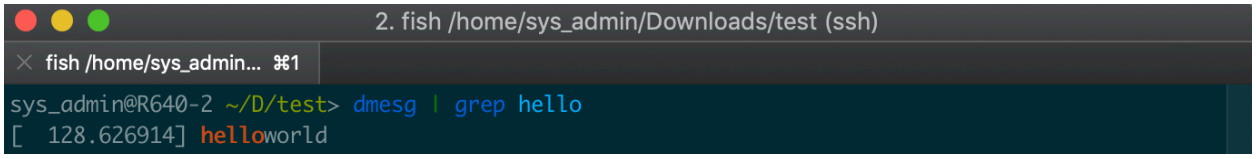
Compile the program:

```
$ 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.



Please use diff command to highlight your modification:
$ diff -u original_file.c modified_file.c > 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.

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 and kernel space source code 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, …

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**.