

CSE Project 1: Adding a System Call to the Linux Kernel

Wang Ziyi
515030910578
wzy_sjtu@foxmail.com

December 17, 2017

Abstract

This is the report for my CSE course project 1: Adding a System Call to the Linux Kernel in SJTU. In this project, I added a new system call to the Linux Kernel 4.14, which was released on Nov.12, 2017. I implemented a system call named *printhello* that print the sentence "hello world no. n " to the screen, where n is the parameter the user gives. The whole report includes the description, the process of implementation, the source code, the bugs fixed and finally a demonstration of a test program.

1 Project Description

A system call is the programmatic way for user programs when the program in user mode requires access to RAM or a hardware resource and it must ask the kernel to provide access to that resource. The situations where system calls perform their function include visiting hardware-related services, creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system.

In this project, we need to add a new system call to the kernel with version higher than 3.0. Here I used the latest Linux Kernel 4.14 released on Nov.12. I added a system call named *printhello* which can print the sentence "hello world no. n ", where n is the parameter the user gives.

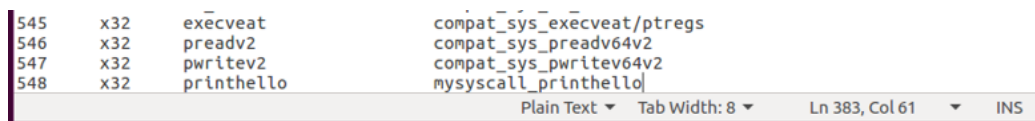
2 Implementation

2.1 Download the Newest Kernel

The project is implemented on the newest Linux Kernel 4.14. To download Linux Kernel, access www.kernel.org, get the newest version and unzip it to a local folder.

2.2 Adding System call Number

In the kernel folder, access `./arch/x86/entry/syscalls/syscall_64.tbl`, and add a new line with a new system call number. We can refer to this system call using this number later. Do not use repeated number, or the kernel may be confused.

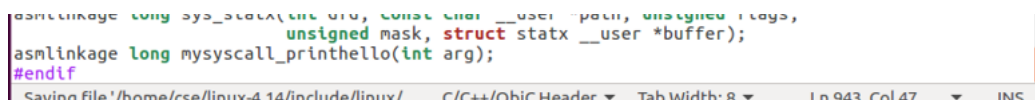


545	x32	execveat	compat_sys_execveat/ptregs
546	x32	preadv2	compat_sys_preadv64v2
547	x32	pwritev2	compat_sys_pwritev64v2
548	x32	printhello	mysyscall_printhello

Figure 1: Add the new system call number into the number table.

2.3 Declare System Call Function

Access `./include/linux/syscalls.h`, in the last of the file, add the declaration of the system call function.



```
asm(
    long sys_execveat(int fd, const char __user *path, unsigned flags,
        unsigned mask, struct statx __user *buffer);
    asm(
    long mysyscall_printhello(int arg);
#endif
```

Figure 2: Add the function declaration.

2.4 Define System Call Function

Access `./kernel/sys.c`, in the last of the file, define the function to print "hello world no. *n*". Notice that we should use *printk* rather than *print* here. See figure 3.

```

        return 0;
    }
    asm linkage long mysyscall_printhello(int arg){
        printk("hello world no. %d\n", arg);
        return 0;
    }
#endif /* CONFIG_COMPAT */

```

Saving file '/home/cse/linux-4.14/kernel/svs.c'...

C Tab Width: 8 Ln 2553, Col 18 INS

Figure 3: Add the function definition.

2.5 Building a New Kernel

Due to different versions, the way of building a new kernel in Linux Kernel 4.x has quite changed from that of 2.x shown in the textbook. The commands used are as follows:

```

1 make oldconfig
2 sudo make -j8
3 sudo make modules_install
4 sudo make install

```

After that, reboot the system. Now when I check the version of the system kernel, it returns 4.14, which indicates my kernel has been successfully updated.

```

cse@cse-VirtualBox: ~
cse@cse-VirtualBox:~$ uname -r
4.14.0
cse@cse-VirtualBox:~$

```

Figure 4: New version of system kernel.

3 Problems and Solutions

When building the kernel, I came across an error message on executing *sudo make* that said "fatal error: openssl/opensslv.h: No such file or directory". I thought this may be resulted from lacking the package OpenSSL. So I tried to install this package before *sudo make*, and it worked. You can refer to the figures to see what happened.

4 Source Code

You can find all the source code in the figures of implementation process.

```
cse@cse-VirtualBox: ~/linux-4.14
CHK    scripts/mod/devicetable-offsets.h
/bin/sh: 1: cannot create scripts/modules.order: Permission denied
scripts/Makefile.build:484: recipe for target 'scripts/modules.order' failed
make[1]: *** [scripts/modules.order] Error 2
Makefile:562: recipe for target 'scripts' failed
make: *** [scripts] Error 2
cse@cse-VirtualBox: ~/linux-4.14$ sudo make
Firefox Web Browser  onfig/kernel.release
CHK    include/generated/uapi/linux/version.h
CHK    include/generated/utsrelease.h
CHK    include/generated/timeconst.h
CHK    include/generated/bounds.h
CHK    include/generated/asm-offsets.h
CALL   scripts/checksyscalls.sh
CHK    scripts/mod/devicetable-offsets.h
HOSTCC scripts/sign-file
scripts/sign-file.c:25:30: fatal error: openssl/opensslv.h: No such file or directory
compilation terminated.
scripts/Makefile.host:102: recipe for target 'scripts/sign-file' failed
make[1]: *** [scripts/sign-file] Error 1
Makefile:562: recipe for target 'scripts' failed
make: *** [scripts] Error 2
cse@cse-VirtualBox: ~/linux-4.14$
```

Figure 5: Fatal error

```
cse@cse-VirtualBox: ~/linux-4.14$ sudo apt-get install libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libssl-doc libssl1.0.0 zlib1g-dev
The following NEW packages will be installed:
  libssl-dev libssl-doc zlib1g-dev
The following packages will be upgraded:
  libssl1.0.0
1 upgraded, 3 newly installed, 0 to remove and 244 not upgraded.
Need to get 3,675 kB of archives.
After this operation, 10.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Figure 6: Install openssl

```
cse@cse-VirtualBox: ~/linux-4.14$ sudo make -j8
CHK    include/config/kernel.release
CHK    include/generated/uapi/linux/version.h
CHK    include/generated/utsrelease.h
CHK    scripts/mod/devicetable-offsets.h
CHK    include/generated/timeconst.h
CHK    include/generated/bounds.h
CHK    include/generated/asm-offsets.h
CALL   scripts/checksyscalls.sh
CC     init/main.o
CC     init/do_mounts.o
CC     init/do_mounts_initrd.o
CHK    include/generated/compile.h
CC     init/do_mounts_md.o
CC     init/initramfs.o
HOSTCC usr/gen_init_cpio
UPD     include/generated/compile.h
CC     init/calibrate.o
CC     arch/x86/crypto/crc32c-intel_glue.o
GEN     usr/initramfs_data.cpio
```

Figure 7: Problem solved

5 Test

I wrote a C file named "hello.c" to call the system function. And use the following command to see the result. The result is the same as expected.

```
1 dmesg | grep "hello"
```

```
#include <unistd.h>
int main(){
    for (int i=0;i<3;i++)
    {
        syscall(548,i);
    }
    return 0;
}
```

Figure 8: User function

```
cse@cse-VirtualBox:~$ dmesg |grep "hello"
[ 536.463312] hello world no. 0
[ 536.463313] hello world no. 1
[ 536.463314] hello world no. 2
```

Figure 9: User function

6 Conclusion

In this project, we practiced the relative knowledge of kernel and system call, additionally familiarized ourselves with the Linux terminate. The difficulty of this project lies in the kernel version. The newest Linux Kernel 4.x is not very commonly used, and there is a lack of document on the Internet. However, once we get familiar with this process, it is in fact pretty clear and simple.