

Project 1: 向 Linux 内核中添加系统调用

516030910259 刘欣鹏

1. 实验目的

以 Linux 内核为例，了解操作系统中系统调用的工作原理。

2. 实验原理

系统调用向用户态程序提供了由操作系统实现的各个功能的接口。在 Linux 系统中，一个系统调用是通过在 EAX 寄存器中存储系统调用号，在其他硬件寄存器中存储系统调用参数，随后执行陷阱指令而完成的。陷阱指令执行后，系统调用号被用于索引相应 handler 的起始地址。进程随后跳跃到该地址，并将进程由用户态转到内核态。在内核态下，进程得以执行在用户态下不能执行的指令。

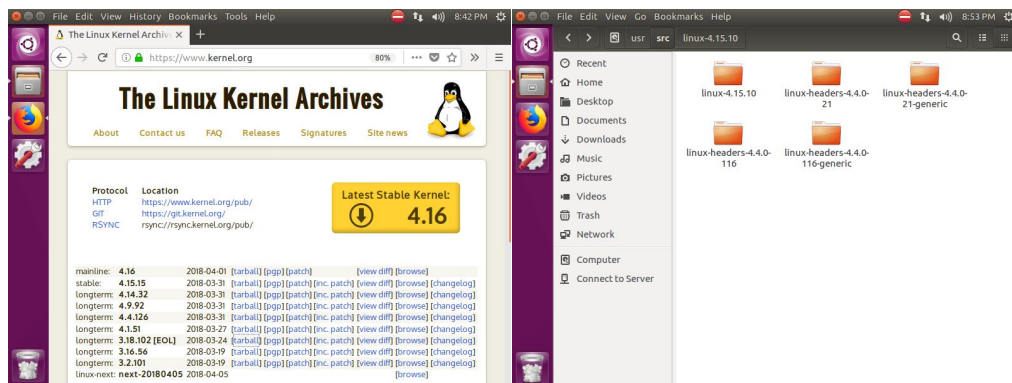
3. 实验步骤

3.1. 下载内核源代码并解压

访问 <https://www.kernel.org/>。在我的实验中下载的版本是 4.15.10。

终端内输入命令：

```
su -
mv linux-4.15.10.tar.xz ./usr/src/linux-4.15.10.tar.xz
cd ./usr/src
tar -xvf linux-4.15.10.tar.xz
```



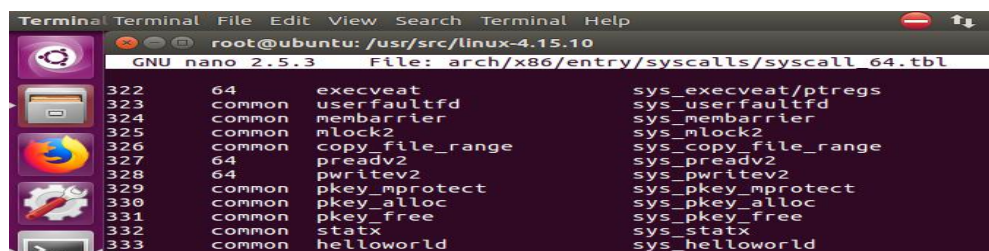
3.2. 添加系统调用号

在该版本的 Linux 内核中，系统调用号存储在内核文件夹下 `./arch/x86/entry/syscalls/syscall_64.tbl` 文件中。

终端内键入命令：

```
cd linux-4.15.10
nano arch/x86/entry/syscalls/syscall_64.tbl
```

在后面添加系统调用函数 `helloworld` 及其相关信息。在我的虚拟机上，其系统调用号为 333。

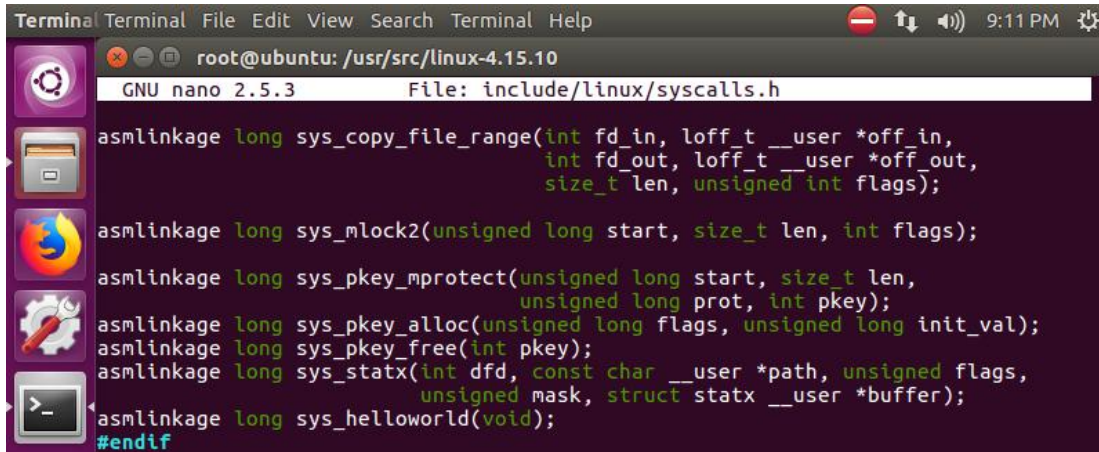


3.3. 声明系统调用函数原型

终端内键入命令：

```
nano include/linux/syscalls.h
```

在后面添加新系统调用的原型。



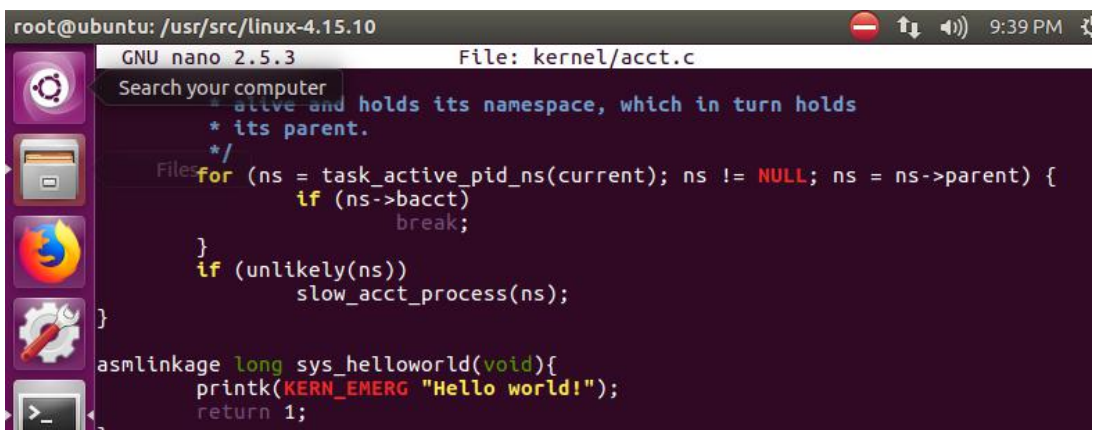
```
Terminal Terminal File Edit View Search Terminal Help
root@ubuntu: /usr/src/linux-4.15.10
GNU nano 2.5.3 File: include/linux/syscalls.h
asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,
                                     int fd_out, loff_t __user *off_out,
                                     size_t len, unsigned int flags);
asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);
asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len,
                                   unsigned long prot, int pkey);
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
                          unsigned mask, struct statx __user *buffer);
asmlinkage long sys_helloworld(void);
#endif
```

3.4. 添加系统调用函数的定义

终端内键入命令：

```
nano kernel/acct.c
```

在最后加入 helloworld 系统调用函数的定义。



```
root@ubuntu: /usr/src/linux-4.15.10
GNU nano 2.5.3 File: kernel/acct.c
Search your computer
/* active and holds its namespace, which in turn holds
 * its parent.
 */
Files for (ns = task_active_pid_ns(current); ns != NULL; ns = ns->parent) {
    if (ns->bacct)
        break;
}
if (unlikely(ns))
    slow_acct_process(ns);
}
asmlinkage long sys_helloworld(void){
    printk(KERN_EMERG "Hello world!");
    return 1;
}
```

3.5. 编译、安装内核

终端内依次键入命令：

```
menuconfig
```

```
make
```

```
make modules_install
```

```
make install
```

等待指令执行完毕，时间较长。

3.6. 重启进入新内核



```
Last login: Wed Mar 28 01:29:21 PDT 2018 on tty2
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.15.10 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

30 packages can be updated.
0 updates are security updates.
```

如画蓝线处所示，当前使用的内核即是添加了新系统调用的 4.15.10 版本内核。

3.7. 测试系统调用

```
ubuntu@ubuntu:~$ less hello.c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main()
{
    long amma=syscall(333);
    printf("System call returns %d\n",amma);
    return 0;
}
ubuntu@ubuntu:~$ gcc -o hello hello.c
hello.c: In function 'main':
hello.c:9:9: warning: format '%d' expects argument of type 'int', but argument 2 has
[-Wformat=]
    printf("System call returns %d\n",amma);
    ^
ubuntu@ubuntu:~$ ./hello
System call returns 1
ubuntu@ubuntu:~$ ./hello
[ 361.614600] Hello world!
System call returns 1
ubuntu@ubuntu:~$ ./hello
[ 367.070071] Hello world!
System call returns 1
ubuntu@ubuntu:~$ _
```

创建 hello.c 文件，内容如上。用 gcc 编译并执行 hello.c。输出结果如上。亦可通过键入 dmesg 查看系统日志。

```
[ 361.614600] Hello world!
[ 367.070071] Hello world!
[ 378.812131] Hello world!
[ 533.027933] Hello world!
ubuntu@ubuntu:~$
```

4. 心得与体会

在实验过程中，遇到的主要问题是教科书上实验指导与实际操作的不符。教科书上使用的内核版本为 2.x，而现在主流的 Linux 内核已经达到了 4.x 版本，教科书上给出的许多路径和指导已经不再适用了。因此，在实验过程中我主要是通过上网查询相关资料而最终完成的。

通过本次实验，我初步了解了如何在 Linux 内核中添加简单的系统调用，如何在编译并安装新的 Linux 内核，以及如何用 C 语言程序进行系统调用，在实验中加深了对 Linux 系统使用的了解。