《操作系统》课第四次实验报告

学院:	软件学院
姓名:	杨万里
学号:	2013774
邮箱:	2013774@mail.nankai.edu.cn
时间:	2022/10/7

0. 开篇感言

对操作系统内核做操作实在是一件令人胆战心惊的事。经历了安装 Ubuntu 和更换新内核的困难过程（容易出错的配置，需要等待非常久的编译）之后，更加害怕内核出问题需要重新编译安装。

本次实验其实整体并不困难，理清思路之后按部就班地做（当然要跳出老师预先设计的坑）就能完成。

但是做的时候还是特别谨慎，生怕哪里做错了最后发现不了要重装内核。

尤其是在验证阶段，因为有一些坑会导致没有预期效果，这时候很容易第一时间想到我是不是前面做错了，不会要重装内核吧……好在最后顺利完成实验。

实验任务不难，但是对内心的挑战倒是不小。

1. 实验题目

添加一个新的系统调用

2. 实验目标

（1） 向 Linux 内核添加一个新的系统调用

（2） 在用户模式测试新的系统调用

3. 原理方法

本次实验的目标是向 Linux 内核中增加一个我们定义的系统调用，并且能够实现用户模式下使用该系统调用。

为了保护计算机和操作系统的核心内容不被用户访问和修改，出现了内核态与用户态。用户态只能使用非特权指令，访问有限的内存。而内核态既可以使用非特权指令又可以使用特权指令，同时访问全部的内存。

因此，如果用户模式下涉及到保护的内存等，就需要系统调用，也就是借助操作系统来完成一些动作。系统调用实际上就是操作系统提供给用户的接口，由操作系统代理完成用户想要完成的工作。

在操作系统的内核增加系统调用，就需要先在操作系统中“注册”这一需要添加的系统调用（类似函数声明），接着需要在内核中实现该系统调用的具体内容，比如本次实验的打印“Hello new system call schello!**Your ID(2013774)**\n”。

添加了系统调用的操作系统内核实际上已经改变，因此需要重新编译安装，从而用新的内核驱动 Ubuntu。

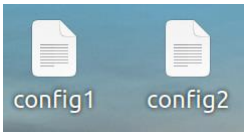
用户态测试新的系统调用只需要写一个简单的程序调用该系统调用，运行之后检查相关功能是否实现即可。

4. 具体步骤

（1） 安装 GCC 软件包。

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/桌面$ sudo apt-get install build-essential
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
build-essential 已经是最新版 (12.9ubuntu3)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 1 个软件包未被升级。
```

（2） 新建两个自定义的内核配置文件 config1 和 config2（备份）。



（3） 在 usr/src/linux/include/linux 文件夹下，打开 syscalls.h 文件。

在#endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */之前，添加一行:

asmlinkage long sys_schello(void);

如下图所示。

```
 * CONFIG_ARCH_HAS_SYSCALL_WRAPPER is enabled.
 */
#include <asm/syscall_wrapper.h>
asmlinkage long sys_schello(void);
#endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
```

（4） 在 usr/src/linux/kernel 文件夹下，打开 sys.c 文件。

在 SYSCALL_DEFINE0(gettid)函数之后，添加一段代码:

SYSCALL_DEFINE0(schello) { printk("Hello new system call schello!**2013774**\n"); return 0; } **这里的 2013774 就是我的学号。**

如下页的图所示。

```
SYSCALL_DEFINE0(gettid)
{
        return task_pid_vnr(current);
}

SYSCALL_DEFINE0(schello)
{
printk("Hello new system call schello!2013774\n");
return 0;
}
```

(5) 针对 64 位 OS，在/usr/src/linux/arch/x86/entry/syscalls 文件夹下，打开 syscall_64.tbl 文件，在 447 common memfd_secret sys_memfd_secret 行之后，添加 448 common schello sys_schello，同时要修改之后的行号。如下图所示。(这个 448 之后验证要用)

```
446     common  landlock_restrict_self  sys_landlock_restrict_self
447     common  memfd_secret            sys_memfd_secret
448     common  schello                 sys_schello
449     common  process_mrelease        sys_process_mrelease
450     common  futex_waitv             sys_futex_waitv
451     common  set_mempolicy_home_node sys_set_mempolicy_home_node
```

(6) 开始编译和安装修改后的内核。

```
wanliyang2013774@wanliyang2013774-virtual-machine:/usr/src/linux$ make clean
  CLEAN   arch/x86/boot/compressed
  CLEAN   arch/x86/boot
  CLEAN   arch/x86/crypto
  CLEAN   arch/x86/entry/vdso
  CLEAN   arch/x86/kernel/cpu
  CLEAN   arch/x86/kernel
  CLEAN   arch/x86/purgatory
  CLEAN   arch/x86/realmode/rm
  CLEAN   arch/x86/tools
  CLEAN   arch/x86/lib
```

```
wanliyang2013774@wanliyang2013774-virtual-machine:/usr/src/linux$ make -j5
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_32.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_64.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_x32.h
  SYSTBL  arch/x86/include/generated/asm/syscalls_32.h
  HOSTCC  scripts/basic/fixdep
  SYSHDR  arch/x86/include/generated/asm/unistd_32_ia32.h
  SYSHDR  arch/x86/include/generated/asm/unistd_64_x32.h
  SYSTBL  arch/x86/include/generated/asm/syscalls_64.h
  HYPERCALLS arch/x86/include/generated/asm/xen-hypercalls.h
  DESCEND objtool
```

……

```
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#2)
wanliyang2013774@wanliyang2013774-virtual-machine:/usr/src/linux$
```

```
wanliyang2013774@wanliyang2013774-virtual-machine:/usr/src/linux$ sudo make modules_install
[sudo] wanliyang2013774 的密码:
  INSTALL /lib/modules/5.19.10/kernel/arch/x86/crypto/aesni-intel.ko
  SIGN    /lib/modules/5.19.10/kernel/arch/x86/crypto/aesni-intel.ko
  INSTALL /lib/modules/5.19.10/kernel/arch/x86/crypto/crc32-pclmul.ko
  SIGN    /lib/modules/5.19.10/kernel/arch/x86/crypto/crc32-pclmul.ko
  INSTALL /lib/modules/5.19.10/kernel/arch/x86/crypto/crct10dif-pclmul.ko
  SIGN    /lib/modules/5.19.10/kernel/arch/x86/crypto/crct10dif-pclmul.ko
  INSTALL /lib/modules/5.19.10/kernel/arch/x86/crypto/ghash-clmulni-intel.ko
  SIGN    /lib/modules/5.19.10/kernel/arch/x86/crypto/ghash-clmulni-intel.ko
```

```
wanliyang2013774@wanliyang2013774-virtual-machine:/usr/src/linux$ sudo make install
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.19.10 /boot/vmlinuz-5.19.10
update-initramfs: Generating /boot/initrd.img-5.19.10
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.19.10 /boot/vmlinuz-5.19.10
```

……

```
Check GRUB_DISABLE_OS_PROBER documentation entry.
done
wanliyang2013774@wanliyang2013774-virtual-machine:/usr/src/linux$
```

（7） 重启并确认新内核是否成功运行。

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/桌面$ uname -a
Linux wanliyang2013774-virtual-machine 5.19.10 #2 SMP PREEMPT_DYNAMIC Fri Oct 7 15:36:34 CST 2022 x86_64 x86_64 x86_64 GNU/Linux
```

新内核成功运行。

（8） 编写用户态测试程序 testschello.c。

这里老师挖了三个坑，第一个是第 5 行 define 后面__要改成_

第二个是 print 要用 printf。第三个是第五行最后那串数字应该是我们在第五步添加行的行号，我这里是 448，要把 336 改成 448，否则没有输出。

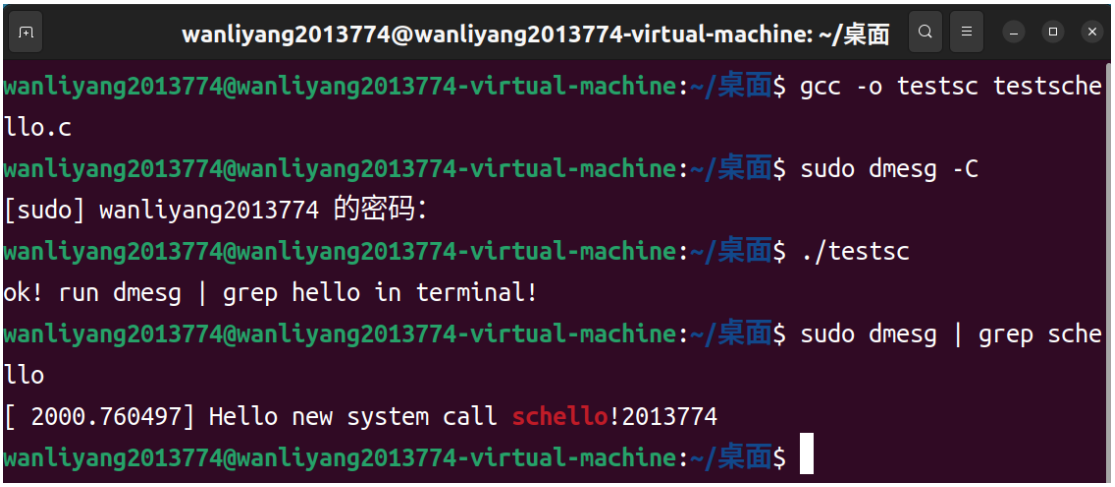
打开(O) *testschello.c ~/桌面

```c
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#define _NR_schello 448
int main(int argc, char *argv[])
{
        syscall(_NR_schello);
        printf("ok! run dmesg | grep hello in terminal!\n");
        return 0;
}

```

(9) 编译并执行用户态应用程序 testschello.c 以验证成果。

这里也有一个小坑，dmesg 前要用 sudo，否则会读取内核缓冲区失败。



到这里说明实验成功。(**2013774 是我的学号**)

5. 总结心得

(1) 区分内核态和用户态是十分必要的，如果没有这层阻碍，用户很有可能在有意或者无意的情况下修改了重要的文件导致计算机或者操作系统出现难以修复的异常。

(2) 虽然有内核态和用户态之分，但是为了保证用户的需求得到满足，操作系统需要像用户提高系统调用，既保证安全又满足用户需求。

(3) 操作系统的内核也不是完全不可达的，本次实验就向内核中添加了系统调

用。不过本次添加的系统调用比较简单，也不涉及重要内容，只是打印一些内容。如果涉及到重要数据的修改也许就没这么容易了。

（4） 对待操作系统还是要小心谨慎。

6. 参考资料

实验指导书《Lab03LinuxKernelCompile_README.pdf》