**通过内核模块法增加一个系统调用**

## 1、一些基本概念

1）内核模块法相较于直接添加系统调用所耗费的时间更短，更加灵活。

2）内核模块的目标代码一旦被链接到内核，它的作用和静态链接的内核目标代码完全等价。

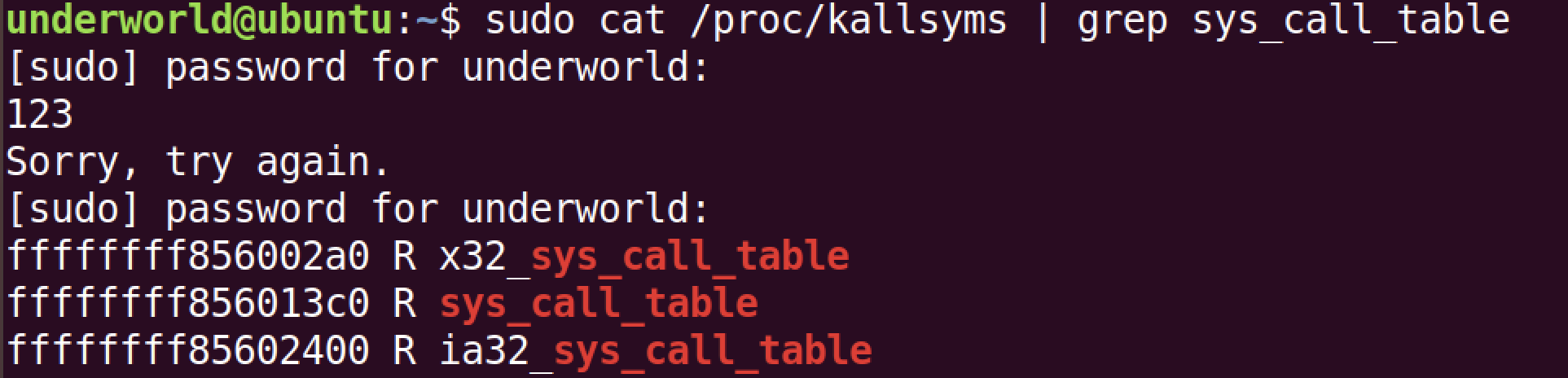
3）修改内核时，不必全部重新编译整个内核，可节省不少时间，避免人工操作的错误。系统中如果需要使用新模块，只要编译相应的模块然后使用特定用户空间的程序将模块插入即可。

## 2、实验过程

**1、准备工作**

（1）获取当前系统syscall\_table的地址

可以使用如下指令：



这里查到的 R sys\_call\_table即是我们需要找到的系统调用表的地址。

（2）查询系统预留的系统调用号

可以在系统对应的内核文件中找到系统调用表，一般系统调用表中都会存在一定的预留系统调用号，可以供给用户自定义系统调用模块。（在实验环境下预留系统调用号在335-390之间），因此在下面的模块源文件中，采用335作为定义的系统调用号。

**2、创建模块对应的源文件simple.c**

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/unistd.h>

#include <linux/sched.h>

#define SYS\_CALL\_TABLE\_ADDRESS 0xffffffff856013c0 //sys\_call\_table对应的地址

#define NUM 335 //系统调用号为335

int orig\_cr0; //用来存储cr0寄存器原来的值

unsigned long \*sys\_call\_table\_my=0;

static int(\*anything\_saved)(void); //定义一个函数指针，用来保存一个系统调用

static int clear\_cr0(void) //使cr0寄存器的第17位设置为0（内核空间可写）

{

unsigned int cr0=0;

unsigned int ret;

asm volatile("movq %%cr0,%%rax":"=a"(cr0));//将cr0寄存器的值移动到eax寄存器中，同时输出到cr0变量中

ret=cr0;

cr0&=0xfffffffffffeffff;//将cr0变量值中的第17位清0,将修改后的值写入cr0寄存器

asm volatile("movq %%rax,%%cr0"::"a"(cr0));//将cr0变量的值作为输入，输入到寄存器eax中，同时移动到寄存器cr0中

return ret;

}

static void setback\_cr0(int val) //使cr0寄存器设置为内核不可写

{

asm volatile("movq %%rax,%%cr0"::"a"(val));

}

asmlinkage long sys\_mycall(void) //定义自己的系统调用

{

printk("system call:当前pid：%d，当前comm:%s\n",current->pid,current->comm);

printk("hello, underworld!\n");

return current->pid;

}

/\* This function is called when the module is loaded. \*/

int simple\_init(void)

{

printk(KERN\_INFO "Loading Module\n");

sys\_call\_table\_my=(unsigned long\*)(SYS\_CALL\_TABLE\_ADDRESS);

//printk("call\_init......\n");

anything\_saved=(int(\*)(void))(sys\_call\_table\_my[NUM]);//保存系统调用表中的NUM位置上的系统调用

orig\_cr0=clear\_cr0();//使内核地址空间可写

sys\_call\_table\_my[NUM]=(unsigned long) &sys\_mycall;//用自己的系统调用替换NUM位置上的系统调用

setback\_cr0(orig\_cr0);//使内核地址空间不可写

return 0;

}

/\* This function is called when the module is removed. \*/

void simple\_exit(void) {

printk(KERN\_INFO "Removing Module\n");

orig\_cr0=clear\_cr0();

sys\_call\_table\_my[NUM]=(unsigned long)anything\_saved;//将系统调用恢复

setback\_cr0(orig\_cr0);

}

/\* Macros for registering module entry and exit points. \*/

module\_init( simple\_init );

module\_exit( simple\_exit );

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("Simple Module");

MODULE\_AUTHOR("SGG");

该模块功能主要是返回调用该模块的进程的pid及运行的可执行文件的文件名。同时调用系统调用printk输出pid并打印“hello，underworld”（可用dmesg查看）。

同时，该模块定义了自己的系统调用，并将定义的系统调用指向了系统的系统调用表，这样可以使用syscall(int num)函数调用模块中定义的系统调用。

**3、编写对应的makefile文件**

obj-m:=simple.o

CURRENT\_PATH:=$(shell pwd)

LINUX\_KERNEL\_PATH:=/usr/src/linux-headers-5.4.0-42-generic

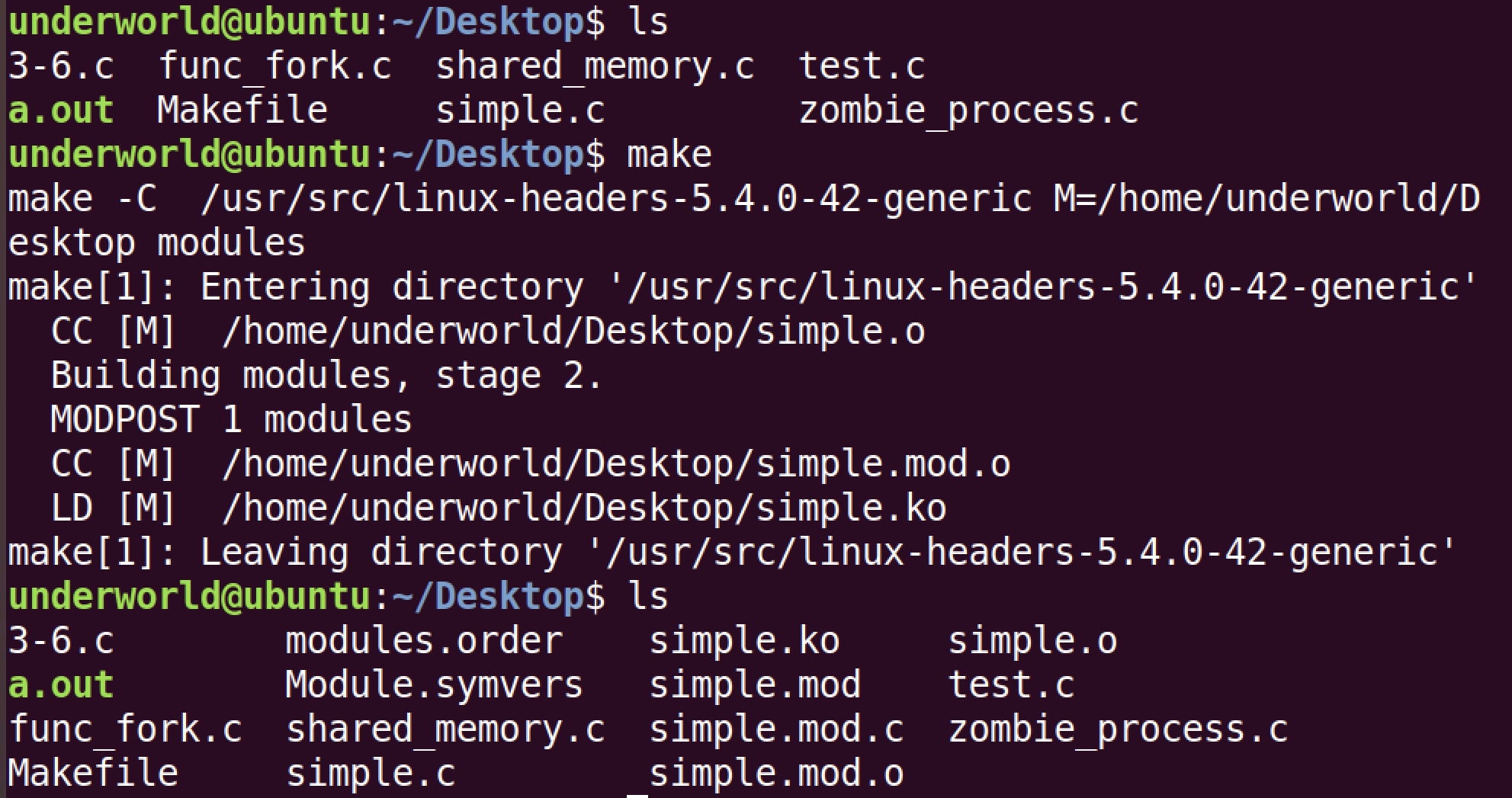
all:

make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) modules

clean:

make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) clean

**4、执行make指令**

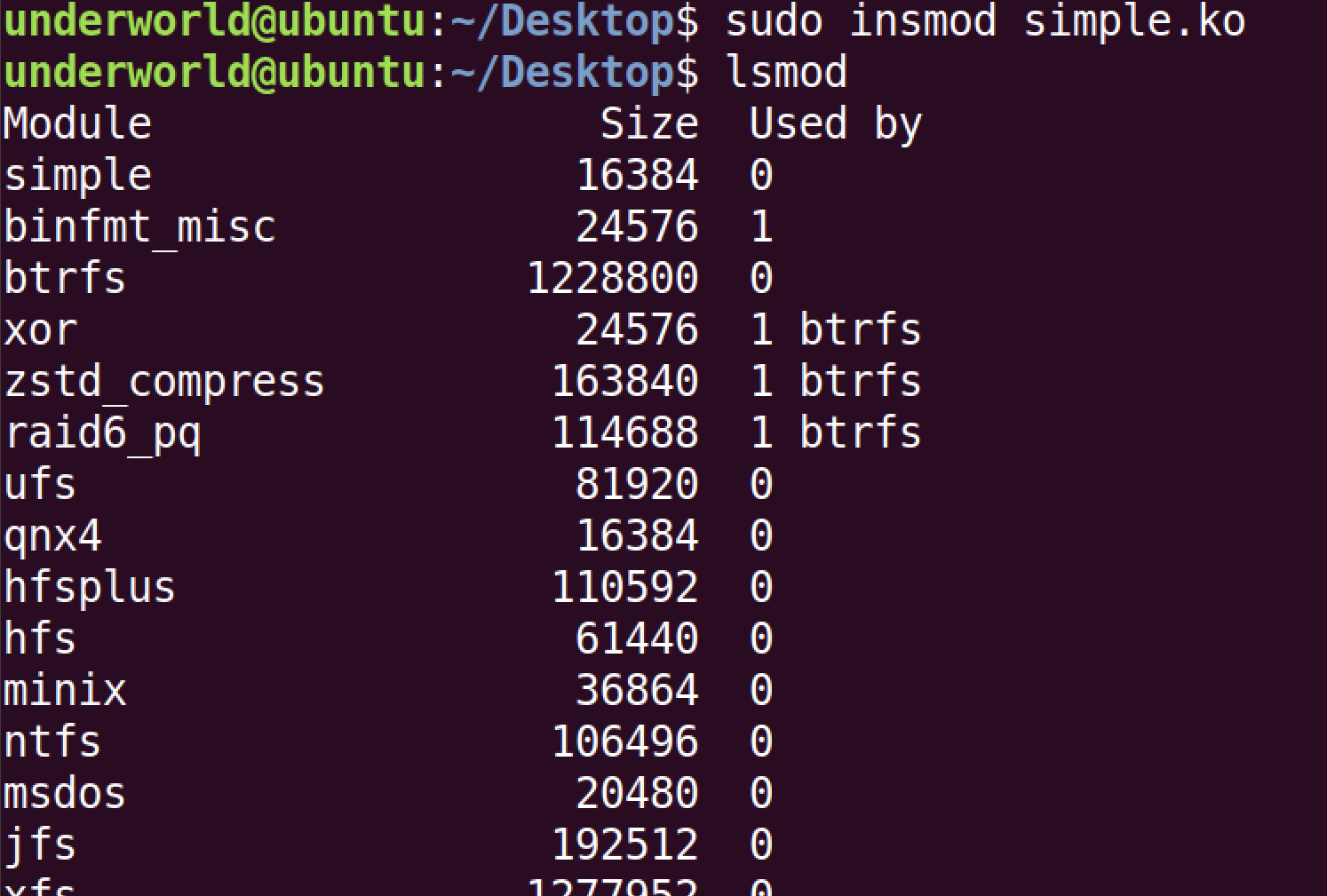


此时，看到的simple.ko即为可以载入系统内核的模块。

**5、载入模块**

通过insmod指令，可以将可链接的模块载入系统内核。

这里即是使用 sudo insmod simple.ko（需要管理员权限）



通过lsmod指令可以在终端输出当前内核载入的模块，可以看到定义的simple模块成功载入。

**6、编写测试函数test.c**

#include<stdio.h>

#include<stdlib.h>

#include<linux/kernel.h>

#include<sys/syscall.h>

#include<unistd.h>

int main()

{

unsigned long x = 0;

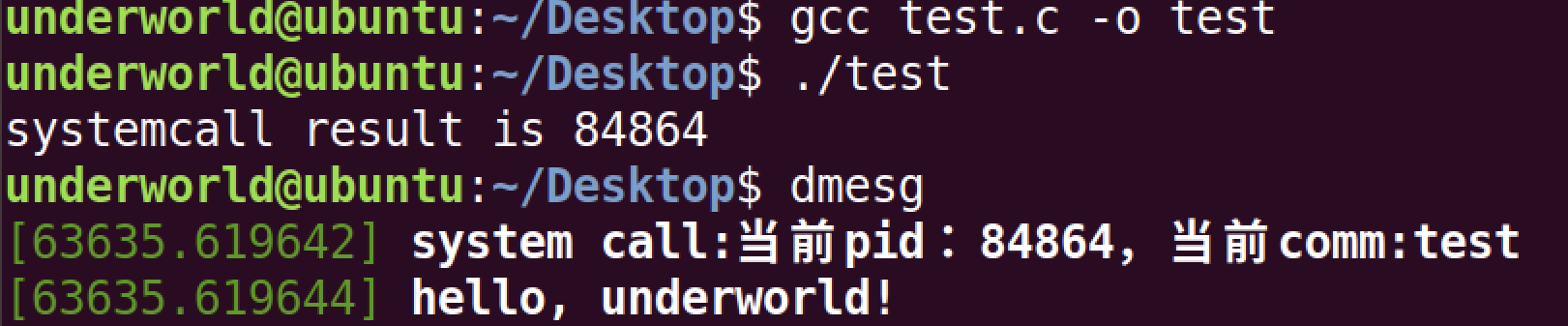
x = syscall(335); //测试335号系统调用

printf("systemcall result is %ld\n", x);

return 0;

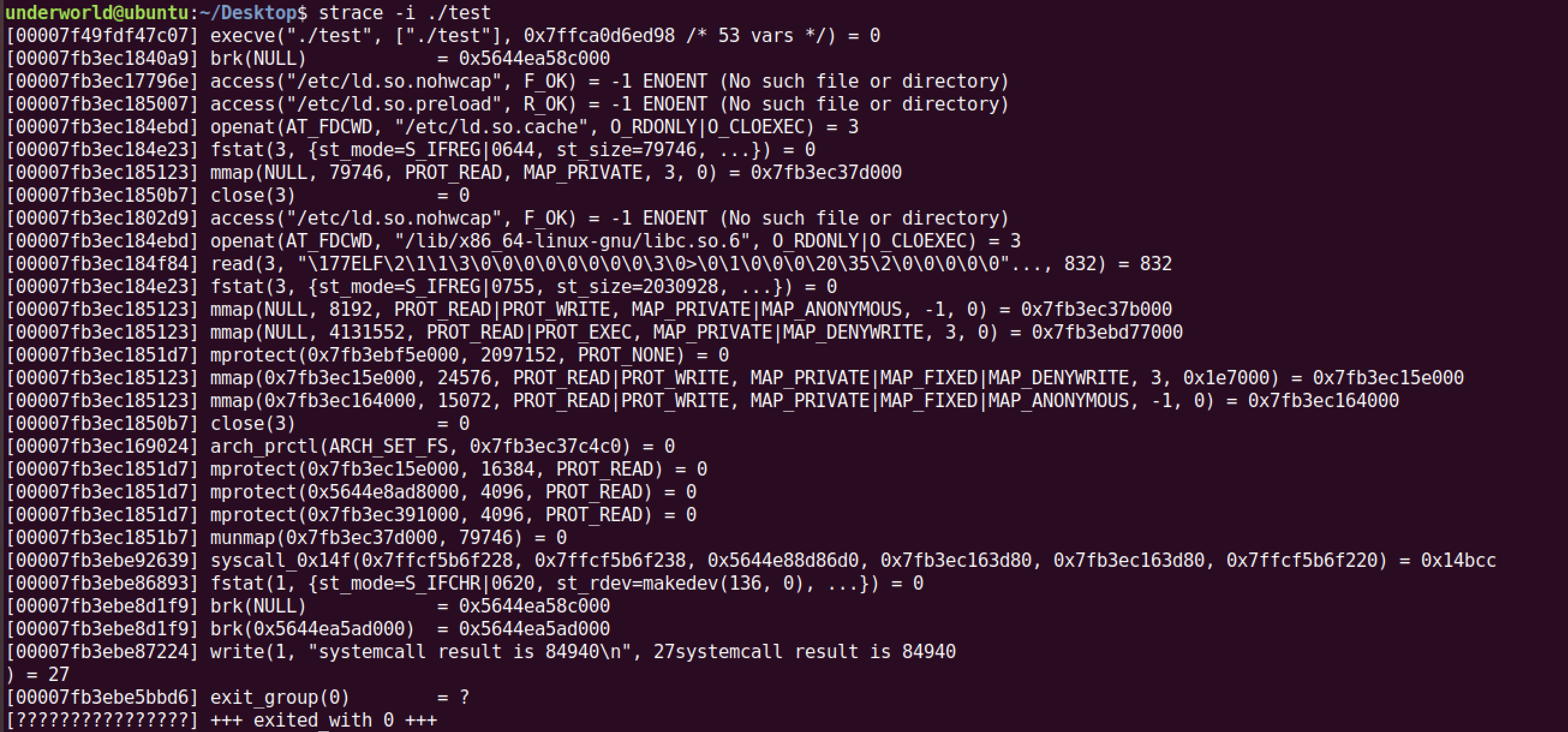
}

这里通过输出对这个系统调用的返回值（即当前进程的pid）来测试，如果返回值为-1，则表示调用失败。同时，可以通过dmesg指令检测系统调用是否正常输出定义的字符串和pid&comm.,如下图。

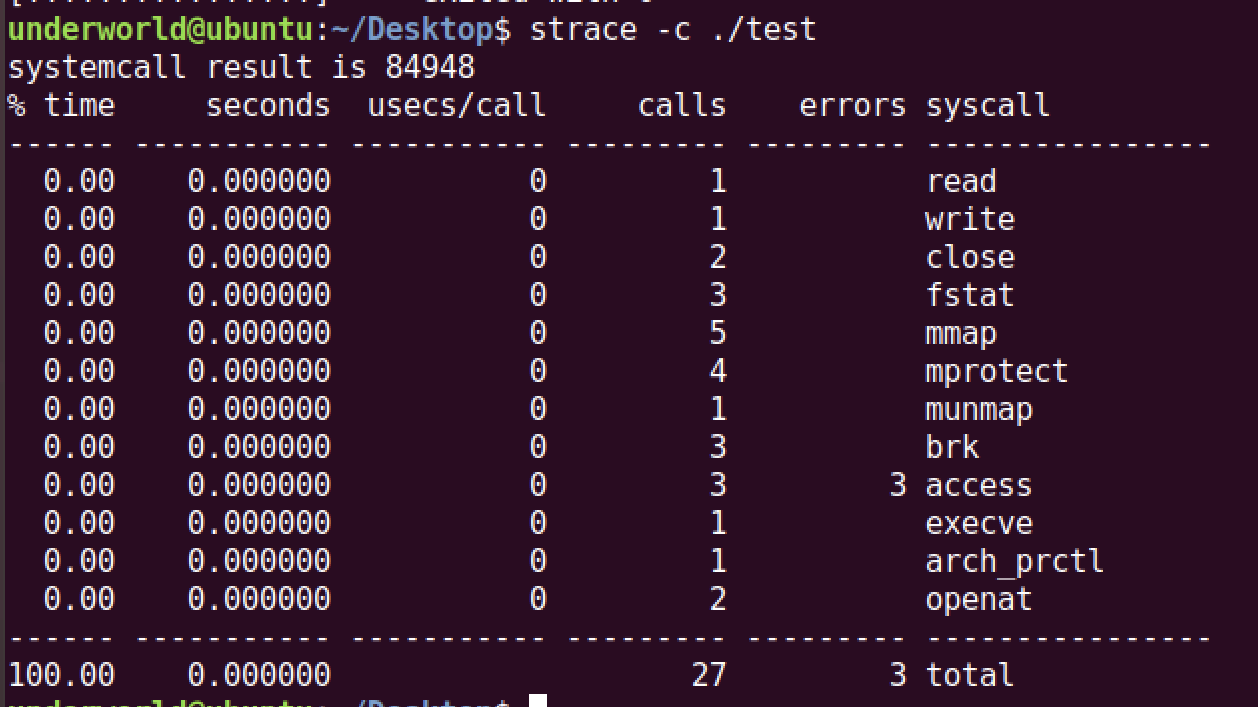


**7、strace追踪测试函数**

strace-i 即可显示程序在哪个地址进行了系统调用，在后续代码调试的部分可将该地址作为断点，再利用GDB进一步定位问题



strace -c 统计每一系统调用的所执行的时间,次数和出错的次数等



**8、移除模块**

实验结束后，应将链接的测试模块移出内核，以保证内核不被污染。用rmmod指令即可，最后可以用lsmod检查是否移除成功。

## 3、实验遇到的问题&解决措施

**1、最初实验不知道怎么通过模块来修改内核的系统调用表的地址**

解决方法：查找一些博客，主要是参考博客，得知可以通过更改寄存器cr0的值来更改内核地址的读写权限。最后成功的将模块的系统调用载入了内核的系统调用。

**2、实验测试之初，定义的系统调用总是返回-1，不能正确的返回pid**

问题原因：上网搜索相关博客后，发现syscall返回-1是没有调用成功的结果。后来发现最初找的系统调用表不是makefile文件中对应的内核的系统调用表。（因为下载了好几个内核，所以用混了），最初使用的预留系统调用号其实超过了内核的系统调用号。找到对应的内核的系统调用表后，使用了正确的预留系统调用号，运行则没有问题。

## 4、参考文献

参考书目：《linux设备驱动开发详解》、《Operating System》（9th）

参考博客链接：<https://blog.csdn.net/thugkd/article/details/50117125>