## 1. 实验目的：

掌握用户程序如何利用系统调用与操作系统内核实现通信的方法，加深对系统调用机制的理解。

## 2. 实验目标：

本次实验主要是修改linux内核增加一个系统调用，打印出所有的进程信息，包括系统函数的实现、系统调用表、系统调用入口的修改，对新的内核编译安装，调试新的系统调用。

## 3. 实验原理：

### 3.1系统调用

系统调用，顾名思义，说的是操作系统提供给用户程序调用的一组“特殊”接口。用户程序可以通过这组“特殊”接口来获得操作系统内核提供的服务，比如用户可以通过文件系统相关的调用请求系统打开文件、关闭文件或读写文件，可以通过时钟相关的系统调用获得系统时间或设置定时器等。

从逻辑上来说，系统调用可被看成是一个内核与用户态程序交互的接口——它好比一个中间人，把用户进程的请求传达给内核，待内核把请求处理完毕后再将处理结果送回给用户态。

系统服务之所以需要通过系统调用来提供给用户态的根本原因是为了对系统进行“保护”，因为我们知道Linux的运行空间分为内核态与用户态，它们各自运行在不同的级别中，逻辑上相互隔离。所以用户进程在通常情况下不允许访问内核数据，也无法使用内核函数，它们只能在用户态操作用户数据，调用用户态函数。比如我们熟悉的“hello world”程序（执行时）就是标准的用户态进程，它使用的打印函数printf就属于用户态函数，打印的字符“hello word”字符串也属于用户态数据。

但是很多情况下，用户进程需要获得系统服务（调用系统程序），这时就必须利用系统提供给用户的“特殊接口”——系统调用了，它的特殊性主要在于规定了用户进程进入内核的具体位置；换句话说，用户访问内核的路径是事先规定好的，只能从规定位置进入内核，而不准许肆意跳入内核。有了这样的陷入内核的统一访问路径限制才能保证内核安全无虞。

### 3.2 内核态中获取进程信息

Linux内核通过一个被称为进程描述符的task\_struct结构体来管理进程，这个结构体包含了一个进程所需的所有信息，比如pid，进程亲属关系等。

## 4. 实验方案：

实验安装的Linux发行版为Ubuntu 17.10，实验前手动升级内核到目前stable分支的最新版4.14.8，实验时使用的内核源代码也是这个版本。

### 4.1 增加系统调用

#### 步骤1：编写程序获取cpu使用率

本次实验添加的两个系统调用，第一个是接收用户态传送的字符串参数并输出，此系统调用较简单，只有输出函数，以此来验证在新内核中添加系统调用是否成功。第二个是打印出所有的进程信息，是本次实验的目标。系统调用使用宏来定义。其中宏是在include/linux/syscalls.h头文件中定义的，宏函数名末尾代表着参数数量，系统调用中最多可以有6个参数。

题目中的代码添加方法是有严重缺陷的，新版本内核中如果在内核态直接访问用户态指针，会导致系统卡死。因此我对代码进行了修改。同时，“printk无法输出中文”也是无稽之谈。

在sys.c文件中加入以下函数：

SYSCALL\_DEFINE1(hello, char \_\_user \* , buf) {

char\* kernel\_buf;

kernel\_buf=(char\*)malloc(sizeof(buf));

copy\_from\_user(&buf , kernel\_buf , sizeof(buf) );

printk("====hello\n");

printk("===%s====", kernel\_buf);

return 1;

}

SYSCALL\_DEFINE0(output\_processes) {

struct task\_struct \*p;

printk("------------输出所有进程------------\n");

printk("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printk("%-20s %-6s %-6s %-20s\n","Name","pid","state","ParentName");

for(p = &init\_task; (p = next\_task(p)) != &init\_task;)

printk("%-20s %-6d %-6d %-20s\n",p->comm , p->pid, p->state, p->parent->comm);

return 1;

}



#### 步骤2：添加系统调用号

#### 在arch/x86/entry/syscalls/syscall\_64.tbl里加入:



在include/linux/syscalls.h的最后加入：



#### 步骤3：编译内核，使用新内核重启

# 与上次实验没有区别

make mrproper

make menuconfig

sudo make install -j15

sudo make modules\_install

sudo make install

sudo update-grub

#### 步骤4：编写程序，执行系统调用

#### 步骤4：查看内核输出信息

成功实现了输出所有进程

## 5. 实验结论：

本次实验主要分为三个步骤：下载linux内核，修改内核，添加新的函数，修改系统调用表，添加系统调用入口；修改完内核，对内核进行重新编译；编译成功后使用新的内核启动，在用户态编程使用新的系统调用，验证整个是否添加成功。主要使学生掌握用户态如何利用系统调用与操作系统内核实现通信的方法，加深对系统调用机制的理解。