**《操作系统》实验报告**

徽标, 公司名称

描述已自动生成

**题目: 添加系统调用**

**学院 计算机学院（国家示范性软件学院）**

**班级 2021211301**

**学号 2021213586**

**姓名 郭栩源**

**2024年 5月**

**目录**

[**第一章 实验概述** 1](#_Toc167576000)

[1.1 实验内容 1](#_Toc167576001)

[1.1.1 添加系统调用 1](#_Toc167576002)

[1.1.2编程调用系统调用 1](#_Toc167576003)

[**第二章 实验步骤** 1](#_Toc167576004)

[2.1 添加系统调用 1](#_Toc167576005)

[2.1.1添加源代码 1](#_Toc167576006)

[2.1.2添加系统调用声明 2](#_Toc167576007)

[2.1.3添加系统调用号 3](#_Toc167576008)

[2.1.4 重新编译安装Linux内核和模块 3](#_Toc167576009)

[2.2 编程调用系统调用 4](#_Toc167576010)

[2.2.1编程调用系统调用fork 4](#_Toc167576011)

[2.2.2编程调用创建的系统调用foo 5](#_Toc167576012)

[**第三章 实验总结** 5](#_Toc167576013)

# **第一章 实验概述**

## 1.1 实验内容

### 1.1.1 添加系统调用

系统调用是操作系统为程序设计人员提供的接口服务。通过使用系统调用，程序员可以更充分地利用计算机资源，使编写的程序更加灵活，功能更加强大。程序员在对系统充分了解的情况下还可以自己定做系统调用，实现一些非专业程序员所难以实现的功能。

本次实验将在Linux内核中添加一个新的系统调用foo，具体实验内容如下：

1.修改内核源码，添加新的系统调用：编辑系统调用表，在适当的位置添加新系统调用的信息，并在内核源码中相应的位置添加新系统调用的实现代码；

2.编译和安装新内核：配置、编译和安装新内核，并更新引导加载程序（如GRUB），以使系统能够启动新内核。

### 1.1.2编程调用系统调用

完成以下两个实验：

1.编程调用系统调用fork：编写一个C程序，调用系统调用 fork 创建一个新的进程，实现进程的创建和父子进程的基本交互；

2.编程调用创建的系统调用foo：编写一个C程序，调用自定义的系统调用 foo，并在内核中实现该系统调用的功能，实现一些非标准的功能或服务。

# **第二章 实验步骤**

## 2.1 添加系统调用

### 2.1.1添加源代码

本次实验的第一个任务是编写添加到内核中的源程序,即添加到内核文件中的一个函数。该函数的名称应该是在新的系统调用名称之前加上sys\_标志。假设新加的系统调用为foo(),功能为原值返回输人的整型数。

在/usr/src/linux/kernel/sys.c文件中添加源代码:

asmlinkage int \_\_x64\_sys\_foo(int x){

printk("%d\n",x);

return x;

}

注意，此处使用printk()而非printf()，printk()和printf()功能类似，但printf不是内核的函数。想要在内核实现在控制台显示字符串的功能，就必须以内核提供的printk()函数来替代。函数printk()不同于printf()的地方是，printk()会把输出的结果送到内核缓冲区里面，最终调用控制台的写函数将其打印出来。

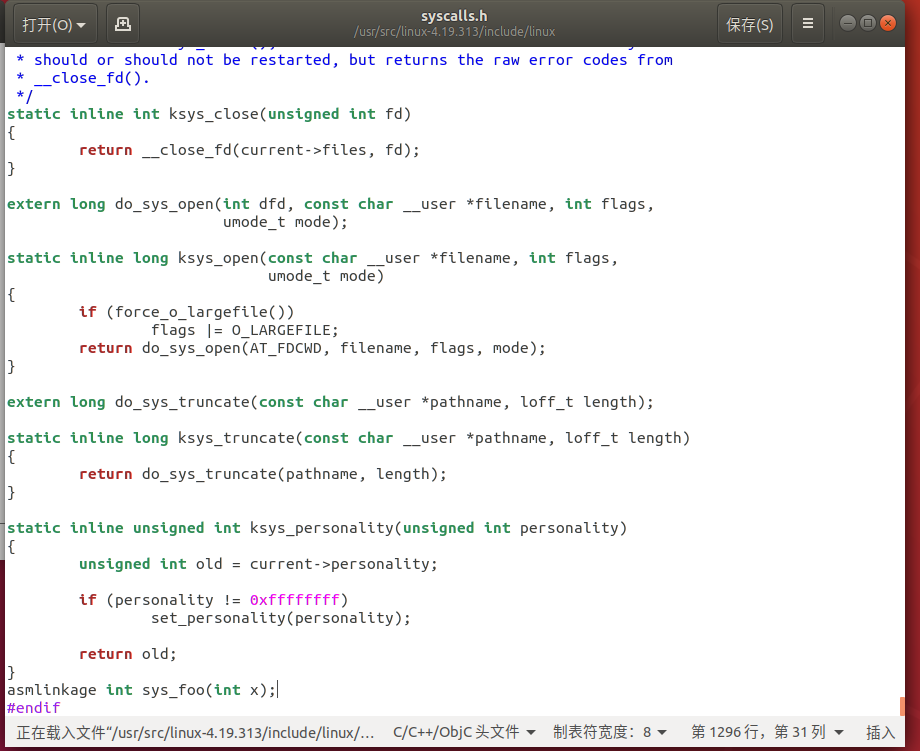
如下所示：



### 2.1.2添加系统调用声明

在源代码中添加系统调用的实现后，我们还需增加系统调用声明。查看系统调用头文件/usr/src/linux/include/linux/syscalls.h，并在文件中添加系统调用声明：

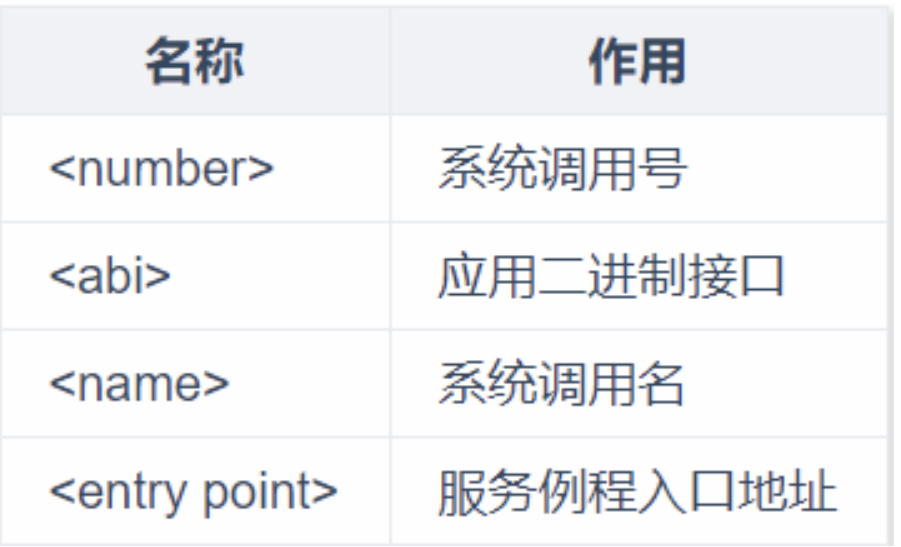
asmlinkage int \_\_x64\_sys\_foo(int x);



### 2.1.3添加系统调用号

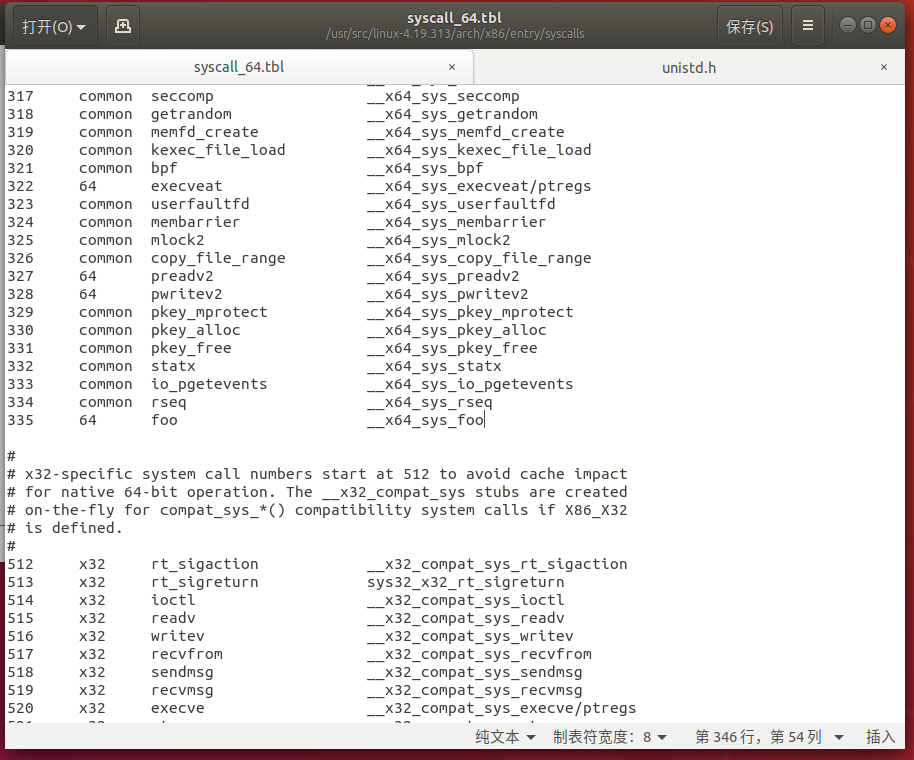
添加新的系统调用之后,下一个任务是让Linux内核的其余部分知道该程序的存在为了给已有的内核程序中增加新函数的链接,需要进行下面的操作。

进入目录/usr/src/linux-4.19.25/arch/x86/entry/syscalls,打开文件/syscall\_64.tbl。这个文件包含了系统调用表,用来给每个系统调用分配一个唯一的号码，格式如下:



在该文件中为刚才的系统调用添加系统调用号如下：

335 64 foo \_\_x64\_sys\_foo



### 2.1.4 重新编译安装Linux内核和模块

安装步骤具体可见实验1.1.1的实验报告，下面简述以下新内核的安装步骤：

首先使用sudo make mrproper和sudo make clean清除之前编译残留的文件。

然后使用sudo make menuconfig添加config文件。

随后开始编译Linux内核，j4表示使用4线程进行编译，2> error.log表示将错误信息重定向到error.log文件下，便于后续查看：

# sudo make -j4 2> error.log

开始编译后，只需等待Linux内核编译完成即可，这个过程是很漫长的，本人在Ubuntu 18.04的虚拟机中编译需要半小时左右。

通过上述步骤我们已经编译好了Linux内核，下面需要进行内核的安装：

# sudo make modules\_install install

通过以上步骤，系统调用正式添加完毕。

## 2.2 编程调用系统调用

### 2.2.1编程调用系统调用fork

在应用程序中调用系统调用fork()非常简单,下面的程序可以很清楚地显示出由于fork()系统调用生成了子进程而产生的分叉作用:

#include<stdio.h>

int main(){

int iUid;

iUid = fork();

if(iUid == 0)

for(;;){

printf("this is parent.\n" );

sleep(1);

}

if(iUid>0)

for(;;){

printf("this is child.\n" );

sleep(1);

}

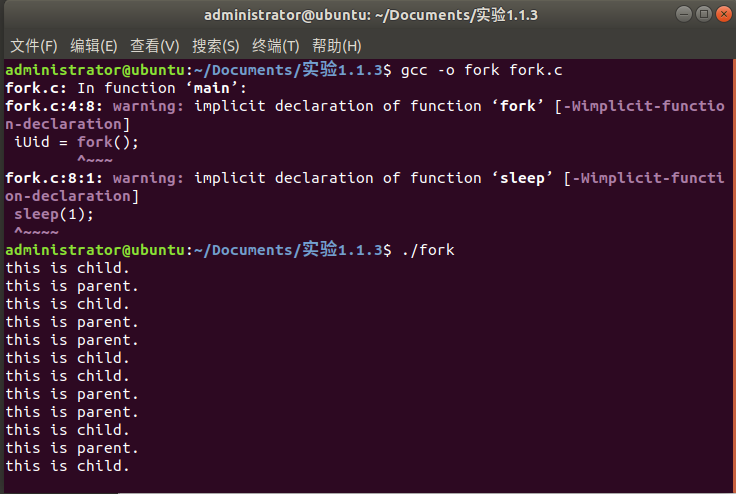
if( iUid< 0)

printf("can not using system call.\n");

return 0;

}

程序运行结果如下，可以看到，fork系统调用生成了一个与父进程一起运行的子进程：



### 2.2.2编程调用创建的系统调用foo

在上文中，我们已经给新编译的Linux内核添加了系统调用foo，下编写程序验证foo能否正常运作：

#include<stdio.h>

#include<linux/unistd.h>

#include<syscall.h>

int main(){

int I,J;

I=100;

J=0;

J=syscall(335,I);

printf("This is the result of new kernel\n" );

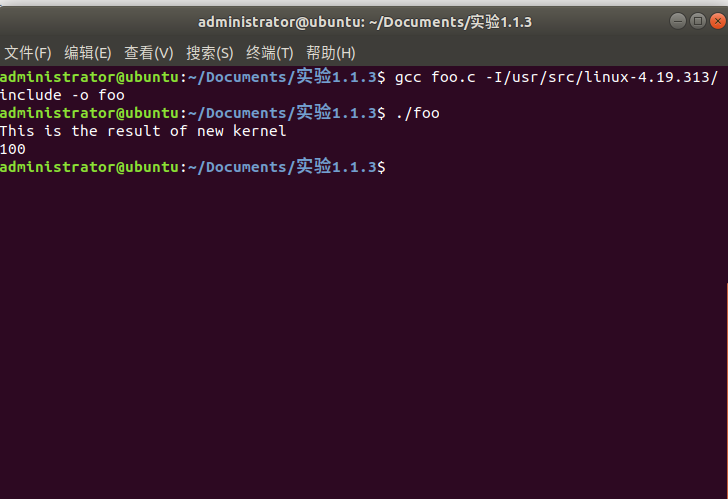
printf("%d", J);

}

其中，int syscall(int number, ...) 函数是一个系统调用的接口，它允许用户空间程序直接调用内核中的系统调用。这个函数接受一个整数参数 number，代表要调用的系统调用的编号，以及可变数量的参数。

注意，由于使用了内核头文件unist.h，不能简单使用普通的编译命令，而应该使用gcc -o foo -I/usr/src/linux-4.19.313/include foo.c。

运行结果如下所示，可见，系统调用被正确添加：



# **第三章 实验总结**

本次实验通过在Linux内核中添加自定义系统调用foo，并使用编程调用系统调用fork和自定义的foo，实现了对系统调用机制的深入理解和实践。

通过本次实验，我不仅加深了对系统调用概念的理解，还掌握了在Linux内核中添加和使用自定义系统调用的方法。实验过程涵盖了内核源码修改、内核编译和安装、应用程序编写等多个环节，全面提升了我对Linux内核操作的熟练度和实际动手能力。

总的来说，本次实验是一次极具挑战性和收获的学习经历。通过动手实践，不仅巩固了理论知识，还提高了实际操作能力，为今后的学习和工作奠定了坚实的基础。希望在未来的学习中，能够继续深入探索Linux内核和操作系统的更多奥秘。