osLab10_2013747_张怡桢

《操作系统》课第10次实验报告

学院:	软件学院
姓名:	张怡桢
学号:	2013747
邮箱:	2662765987@qq.com
时间:	11/24/2022

1. 开篇感言

老师教程X86，奈何吾之arm，摸石头过河成长。

2. 实验题目

一个新系统调用实现列举进程和文件内核拷贝等多项功能，并返回结果到用户程序。

3. 实验目标

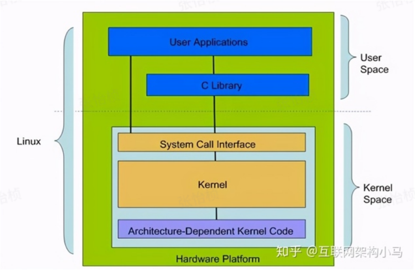
1. Add a new system call with arguments into the linux kernel
2. The new system call will return all processes information to user mode
3. 实现内核中文件拷贝 (见pdf文档)

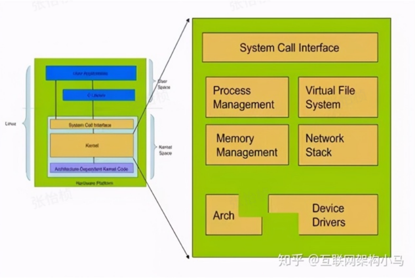
4. 原理方法

3.1 Linux 内核概述：

Linux 内核是 Linux 操作系统（OS）的主要组件，也是计算机硬件与其进程之间的核心接口。它负责两者之间的通信，还要尽可能高效地管理资源。

之所以称为内核，是因为它在操作系统中就像果实硬壳中的种子一样，并且控制着硬件（无论是电话、笔记本电脑、服务器，还是任何其他类型的计算机）的所有主要功能。





5. 具体步骤

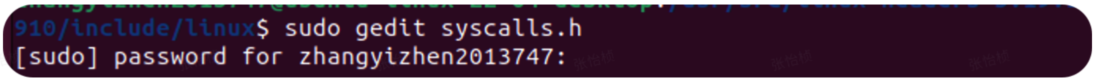
Step1 (Linux kernel 5.19)

include/linux/syscalls.h

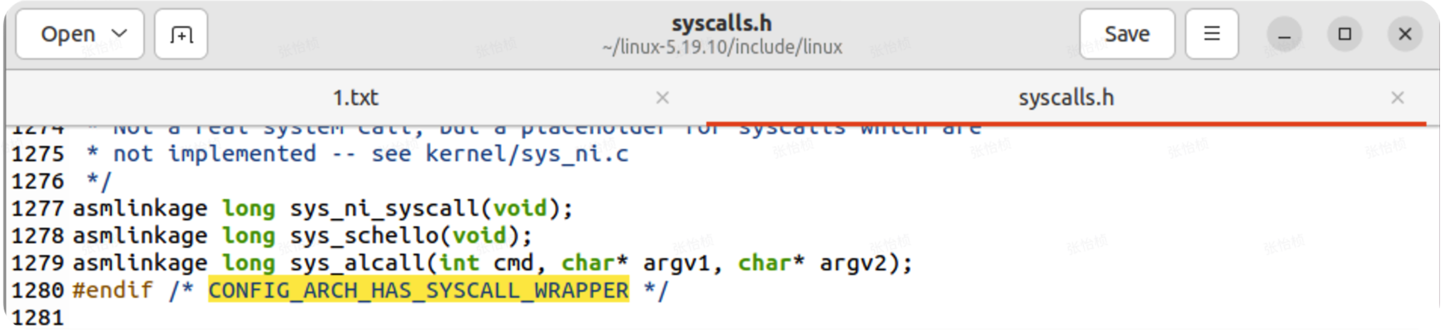
在文件(No. 1279)

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

asmlinkage long sys_alcall(int cmd, char* argv1, char* argv2);



```
910/include/linux$ sudo gedit syscalls.h
[sudo] password for zhangyizhen2013747:
```

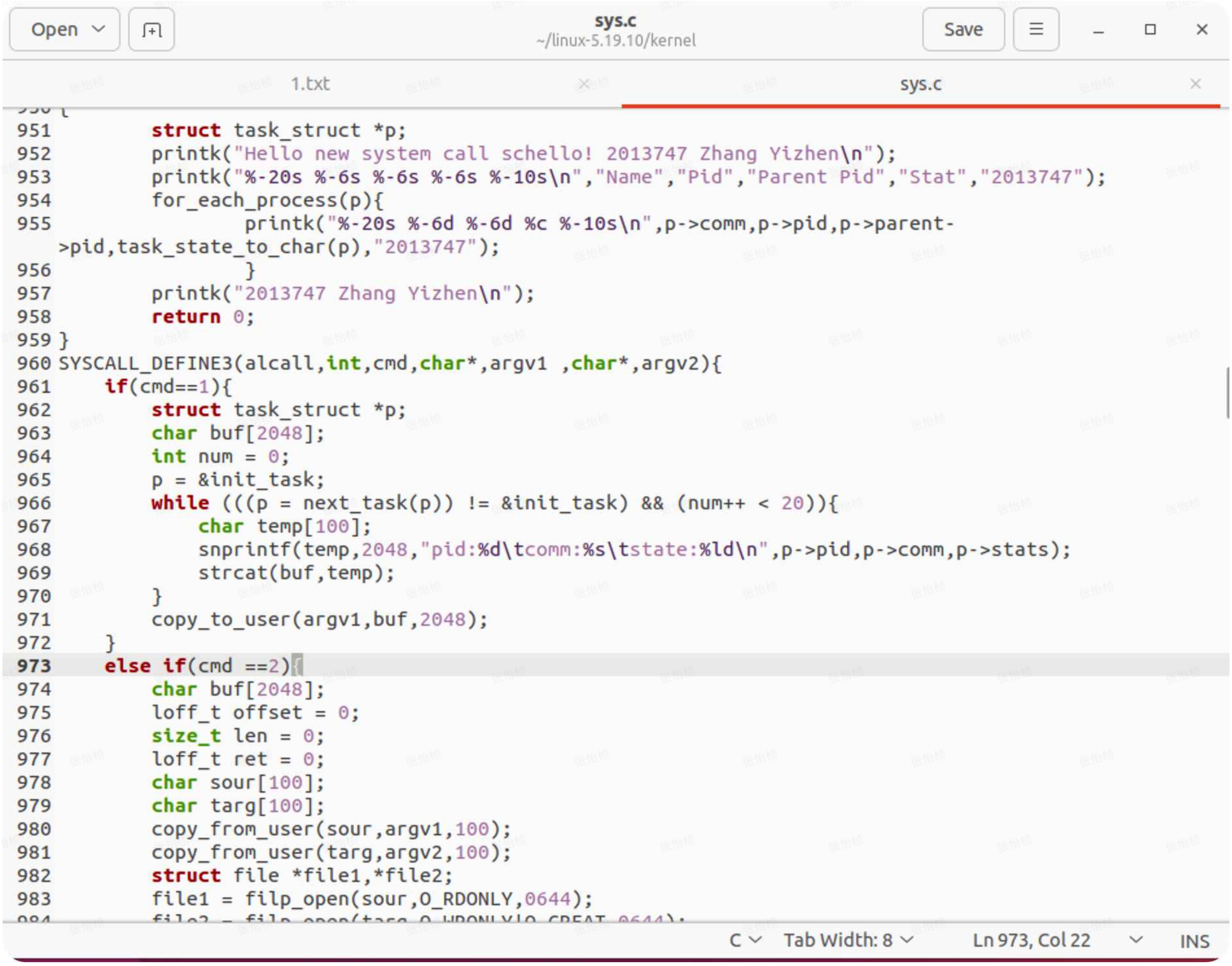


```
1274  * Not a real system call, but a placeholder for syscalls which are
1275  * not implemented -- see kernel/sys_ni.c
1276  */
1277 asmlinkage long sys_ni_syscall(void);
1278 asmlinkage long sys_schello(void);
1279 asmlinkage long sys_alcall(int cmd, char* argv1, char* argv2);
1280 #endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
1281
```

Step2 (Linux kernel 5.19)

kernel/sys.c

在文件SYSCALL_DEFINE0(gettid)函数之后（No. 959），添加如下行:



```
        struct task_struct *p;
        printk("Hello new system call schello! 2013747 Zhang Yizhen\n");
        printk("%-20s %-6s %-6s %-6s %-10s\n","Name","Pid","Parent Pid","Stat","2013747");
        for_each_process(p){
                printk("%-20s %-6d %-6d %c %-10s\n",p->comm,p->pid,p->parent->pid,task_state_to_char(p),"2013747");
                }
        printk("2013747 Zhang Yizhen\n");
        return 0;
}
SYSCALL_DEFINE3(alcall,int,cmd,char*,argv1 ,char*,argv2){
    if(cmd==1){
        struct task_struct *p;
        char buf[2048];
        int num = 0;
        p = &init_task;
        while (((p = next_task(p)) != &init_task) && (num++ < 20)){
            char temp[100];
            snprintf(temp,2048,"pid:%d\tcomm:%s\tstate:%ld\n",p->pid,p->comm,p->stats);
            strcat(buf,temp);
        }
        copy_to_user(argv1,buf,2048);
    }
    else if(cmd ==2){
        char buf[2048];
        loff_t offset = 0;
        size_t len = 0;
        loff_t ret = 0;
        char sour[100];
        char targ[100];
        copy_from_user(sour,argv1,100);
        copy_from_user(targ,argv2,100);
        struct file *file1,*file2;
        file1 = filp_open(sour,O_RDONLY,0644);
        file2 = filp_open(targ,O_WRONLY|O_CREAT,0644);
```

```
SYSCALL_DEFINE3(alcall,int,cmd,char*,argv1 ,char*,argv2){
    if(cmd==1){
        struct task_struct *p;
        char buf[2048];

        int num = 0;
        p = &init_task;
        while (((p = next_task(p)) != &init_task) && (num++ < 20)){
            char temp[100];
            snprintf(temp,2048,"pid:%d\tcomm:%s\tstate:%ld\n",p->pid,p->comm,p->
            strcat(buf,temp);
        }
        copy_to_user(argv1,buf,2048);
    }
    else if(cmd ==2){
        char buf[2048];
        loff_t offset = 0;
        size_t len = 0;
        loff_t ret = 0;
        char sour[100];
        char targ[100];
        copy_from_user(sour,argv1,100);
        copy_from_user(targ,argv2,100);
        struct file *file1,*file2;
        file1 = filp_open(sour,O_RDONLY,0644);
        file2 = filp_open(targ,O_WRONLY|O_CREAT,0644);
        while (1)
        {
            ret = offset;
            len = kernel_read(file1,buf,(size_t)2048,&offset);
            if(len == 0){
                break;
            }
            kernel_write(file2,buf,len,&ret);
            strcpy(buf,"");
        }
        filp_close(file1,NULL);
        filp_close(file2,NULL);
    }
    return 0;
}
```

Step3 (Linux kernel 5.19)

由于我的电脑不是X86架构的，所以我按着老师给的教程的思路修改了arm/tools下的syscall.tbl，但是编译之后并不能生效，在最后的输出依然失败，询问同学，查询网站，找到解决办法【https://blog.csdn.net/m0_51683653/article/details/124133370】

include/uapi/asm-generic/unistd.h

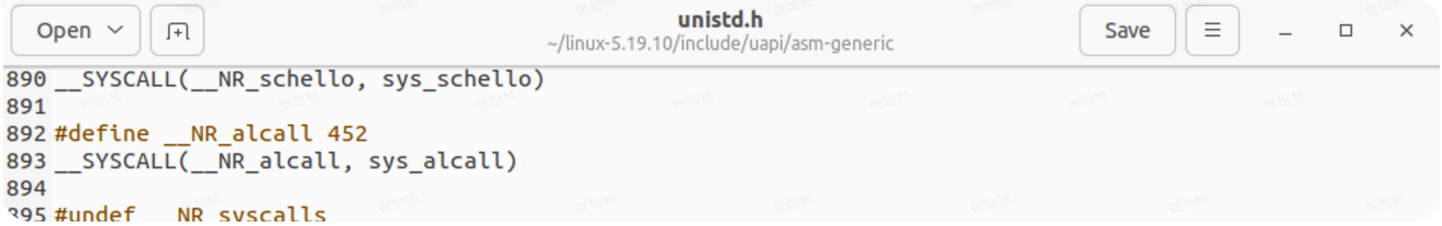
修改及添加系统调用号（注意系统调用号不能随意加，只能一次加1）

在892行加入

#define __NR_alcall 452

__SYSCALL(__NR_alcall, sys_alcall)

并顺次把下一个syscalls序号加1改成453即可。



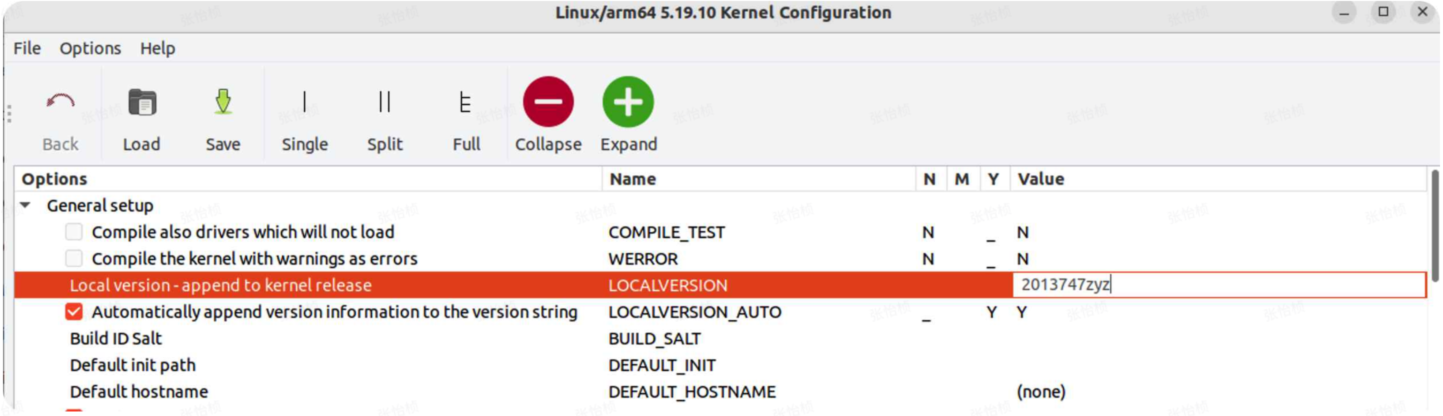
```
890 __SYSCALL(__NR_schello, sys_schello)
891
892 #define __NR_alcall 452
893 __SYSCALL(__NR_alcall, sys_alcall)
894
895 #undef __NR_syscalls
```

Step4

make oldconfig

make gconfig

在这一步我在版本名上把原来的zyz2013747改成了2013747zyz以区别更改之后的新版本

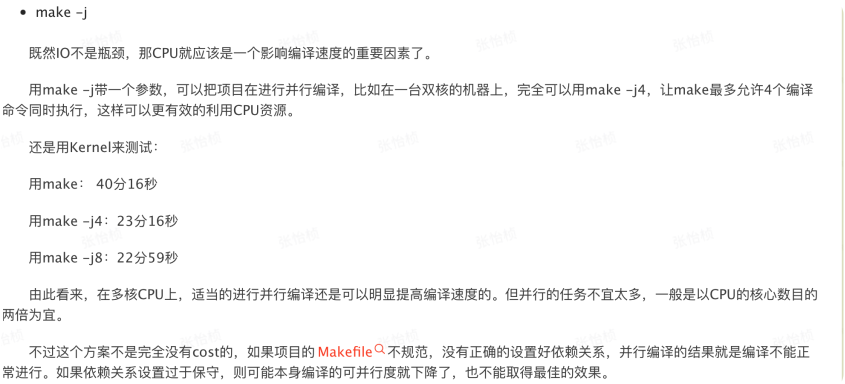


Options	Name	N	M	Y	Value
General setup					
□ Compile also drivers which will not load	COMPILE_TEST	N		_	N
□ Compile the kernel with warnings as errors	WERROR	N		_	N
Local version - append to kernel release	LOCALVERSION				2013747zyz
☑ Automatically append version information to the version string	LOCALVERSION_AUTO	_		Y	Y
Build ID Salt	BUILD_SALT				
Default init path	DEFAULT_INIT				
Default hostname	DEFAULT_HOSTNAME				(none)

make clean

make -j5

// make -jx（x越大，编译的时候用的线程越多，就越快！！！！！）



- make –j

既然IO不是瓶颈，那CPU就应该是一个影响编译速度的重要因素了。

用make –j带一个参数，可以把项目在进行并行编译，比如在一台双核的机器上，完全可以用make –j4，让make最多允许4个编译命令同时执行，这样可以更有效的利用CPU资源。

还是用Kernel来测试：

用make： 40分16秒

用make –j4：23分16秒

用make –j8：22分59秒

由此看来，在多核CPU上，适当的进行并行编译还是可以明显提高编译速度的。但并行的任务不宜太多，一般是以CPU的核心数目的两倍为宜。

不过这个方案不是完全没有cost的，如果项目的Makefile不规范，没有正确的设置好依赖关系，并行编译的结果就是编译不能正常进行。如果依赖关系设置过于保守，则可能本身编译的可并行度就下降了，也不能取得最佳的效果。

sudo make

sudo make modules

sudo make modules_install

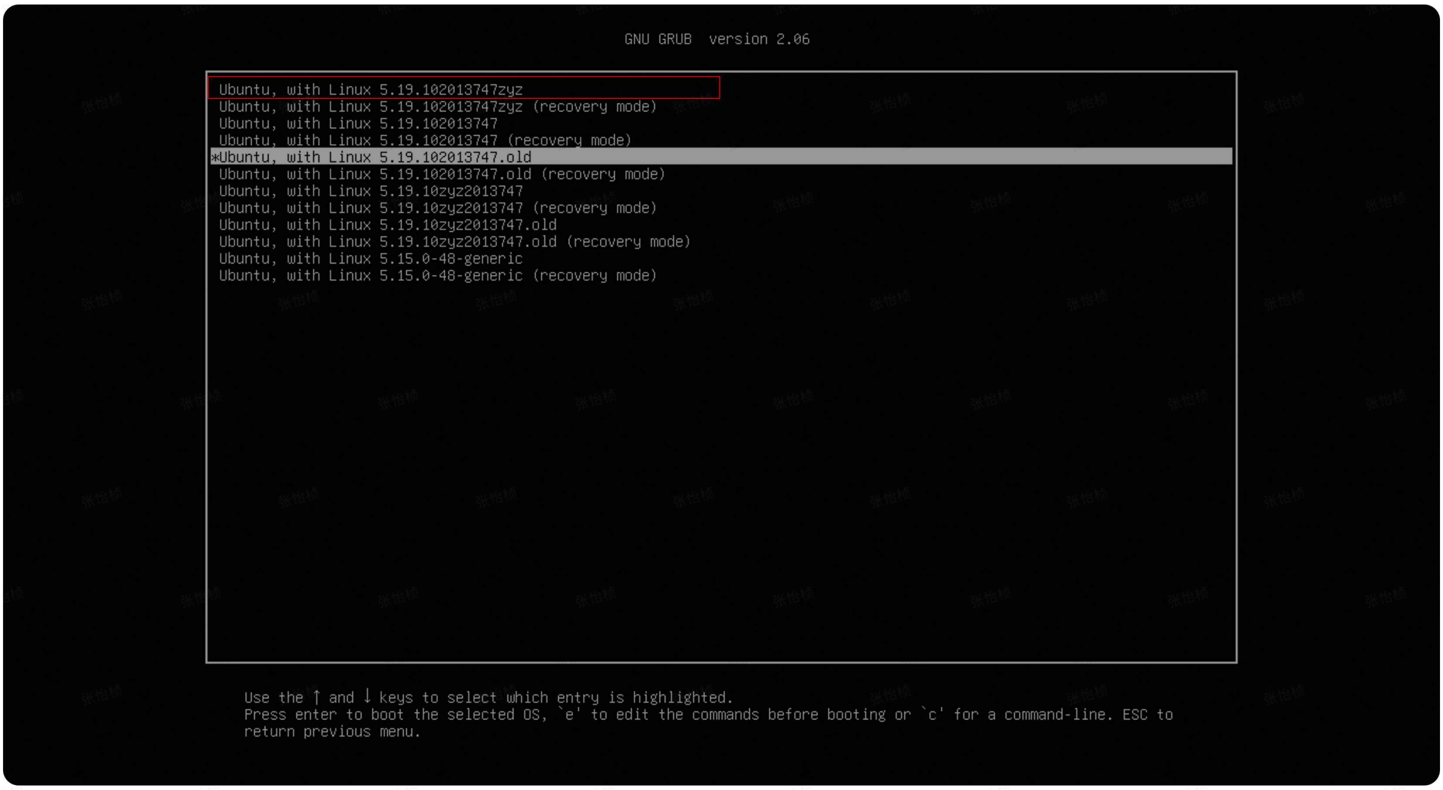
sudo make install

Step 5

重新启动:

sudo reboot

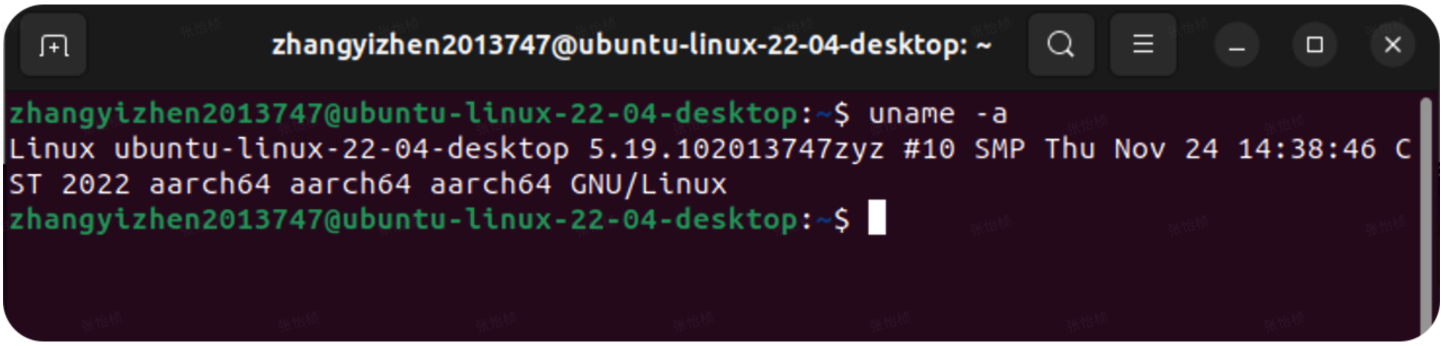
可以看到有2013747zyz后缀的新版本，选择进入即可



```
GNU GRUB  version 2.06

 Ubuntu, with Linux 5.19.102013747zyz
 Ubuntu, with Linux 5.19.102013747zyz (recovery mode)
 Ubuntu, with Linux 5.19.102013747
 Ubuntu, with Linux 5.19.102013747 (recovery mode)
*Ubuntu, with Linux 5.19.102013747.old
 Ubuntu, with Linux 5.19.102013747.old (recovery mode)
 Ubuntu, with Linux 5.19.10zyz2013747
 Ubuntu, with Linux 5.19.10zyz2013747 (recovery mode)
 Ubuntu, with Linux 5.19.10zyz2013747.old
 Ubuntu, with Linux 5.19.10zyz2013747.old (recovery mode)
 Ubuntu, with Linux 5.15.0-48-generic
 Ubuntu, with Linux 5.15.0-48-generic (recovery mode)

    Use the ↑ and ↓ keys to select which entry is highlighted.
    Press enter to boot the selected OS, `e' to edit the commands before booting or `c' for a command-line. ESC to
    return previous menu.
```

确认新内核是否成功运行：

uname -a



```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ uname -a
Linux ubuntu-linux-22-04-desktop 5.19.102013747zyz #10 SMP Thu Nov 24 14:38:46 C
ST 2022 aarch64 aarch64 aarch64 GNU/Linux
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ 
```

Step 6

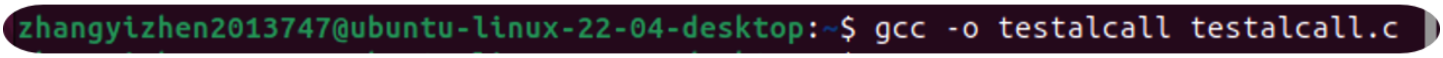
编写用户态测试程序testschello.c

```
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define _NR_alcall 452
int main(int argc, char **argv)
{
    if(strcmp(argv[1],"1")==0){
        char* temp = malloc(1024);
        char* temp2 = malloc(1024);
        syscall(_NR_alcall, 1,temp,temp2);
        printf("%s\n",temp);
    }
    else if(strcmp(argv[1],"2")==0){
        printf("%s\n",argv[2]);
        printf("%s\n",argv[3]);
        syscall(_NR_alcall, 2,argv[2],argv[3]);
    }

    return 0;
}

```

Step 7

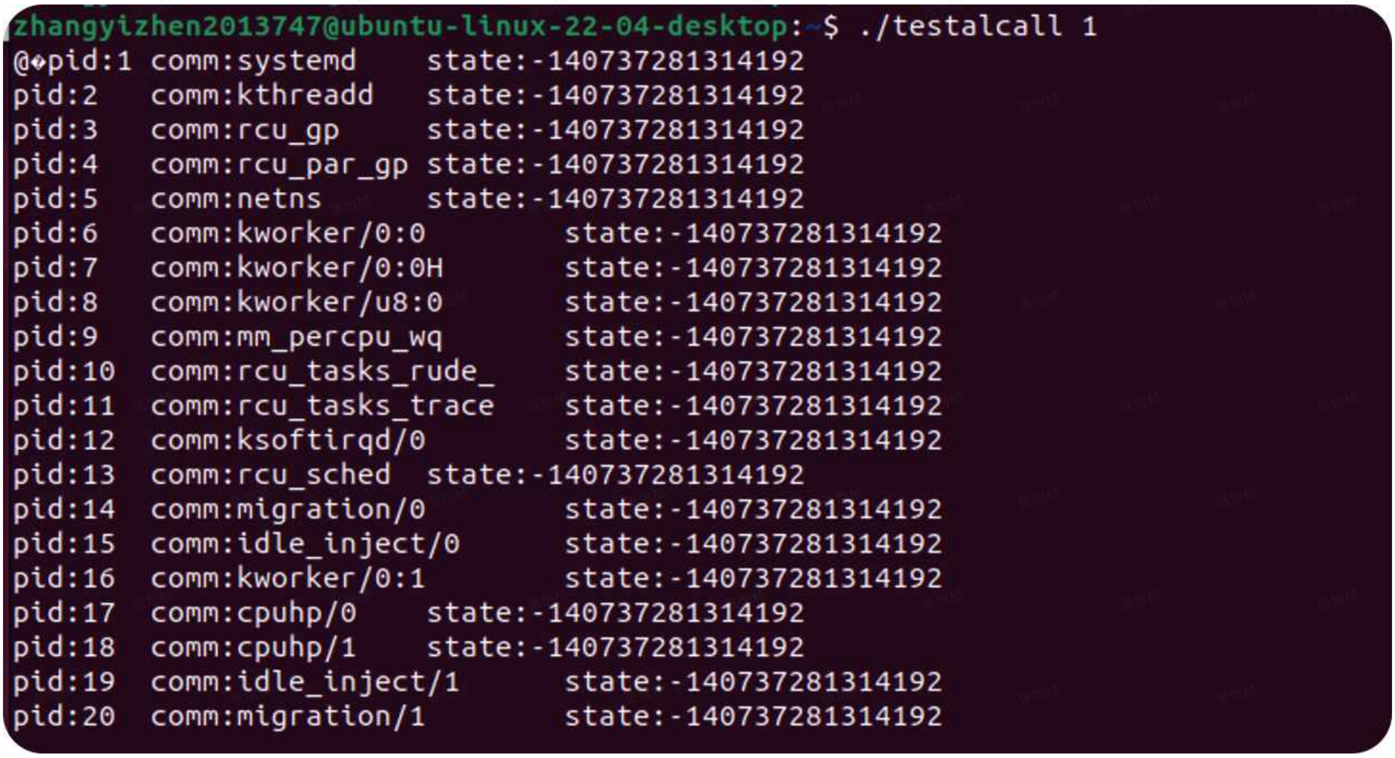
编译用户态测试程序testalcall.c，并执行

gcc -o testalcall testalcall.c



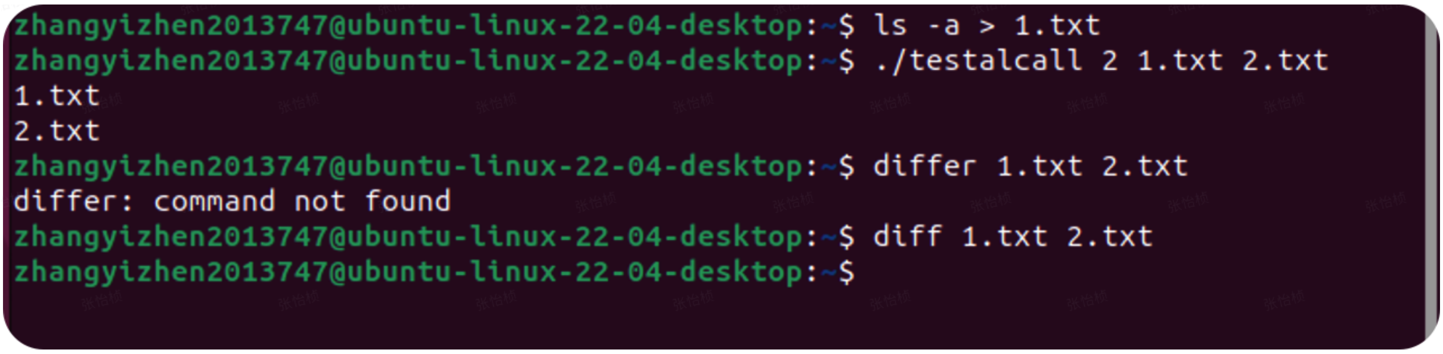
```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ gcc -o testalcall testalcall.c
```

1. 实现列举进程



```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ ./testalcall 1
@◆pid:1 comm:systemd    state:-140737281314192
pid:2   comm:kthreadd   state:-140737281314192
pid:3   comm:rcu_gp     state:-140737281314192
pid:4   comm:rcu_par_gp state:-140737281314192
pid:5   comm:netns      state:-140737281314192
pid:6   comm:kworker/0:0        state:-140737281314192
pid:7   comm:kworker/0:0H       state:-140737281314192
pid:8   comm:kworker/u8:0       state:-140737281314192
pid:9   comm:mm_percpu_wq       state:-140737281314192
pid:10  comm:rcu_tasks_rude_    state:-140737281314192
pid:11  comm:rcu_tasks_trace    state:-140737281314192
pid:12  comm:ksoftirqd/0        state:-140737281314192
pid:13  comm:rcu_sched  state:-140737281314192
pid:14  comm:migration/0        state:-140737281314192
pid:15  comm:idle_inject/0      state:-140737281314192
pid:16  comm:kworker/0:1        state:-140737281314192
pid:17  comm:cpuhp/0    state:-140737281314192
pid:18  comm:cpuhp/1    state:-140737281314192
pid:19  comm:idle_inject/1      state:-140737281314192
pid:20  comm:migration/1        state:-140737281314192
```

2. 文件内核拷贝



```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ ls -a > 1.txt
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ ./testalcall 2 1.txt 2.txt
1.txt
2.txt
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ differ 1.txt 2.txt
differ: command not found
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$ diff 1.txt 2.txt
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~$
```

6. 总结心得

更新编译linux内核的过程让我学会了很多东西，遇到了不少问题，查询了很多相关的资料，对于linux的内核有了进一步的了解。

再一次重新编译的过程中，我将原来的版本后缀zyz2013747改成了2013747zyz，可以看到，修改成功，并且得到了最后的新系统调用结果。

7. 参考资料

1. ARM架构增加新的系统调用