《操作系统》课第 10 次实验报告

学院:	软件学院
姓名:	郁万祥
学号:	2013852
邮箱:	yuwanxiang0114@163.com
时间:	2022.11.18

0. 开篇感言

此次实验，一部分是对于之前实验的总结，另一部分，是对以前实验的扩展，对于文件目录的拷贝和进程信息的获取方式，和之前的实验核心相同，不过，之前的系统调用，是通过测试程序进行系统调用后，使用命令

```
$dmesg | grep schello
```

过去系统调用打印的信息，而此次实验，是需要通过传递参数，将系统调用获取的进程信息传递到用户程序当中，然后进行输出。当然，这也不是很困难，然后，对于文件拷贝这一任务，比较容易，只是将以往实验中通过外部程序的方式改变成通过系统调用的方式即可。

1. 实验题目

Add a New System Call with args to list all processes

2. 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.

4、系统调用的结果是通过参数返回到用户程序中。

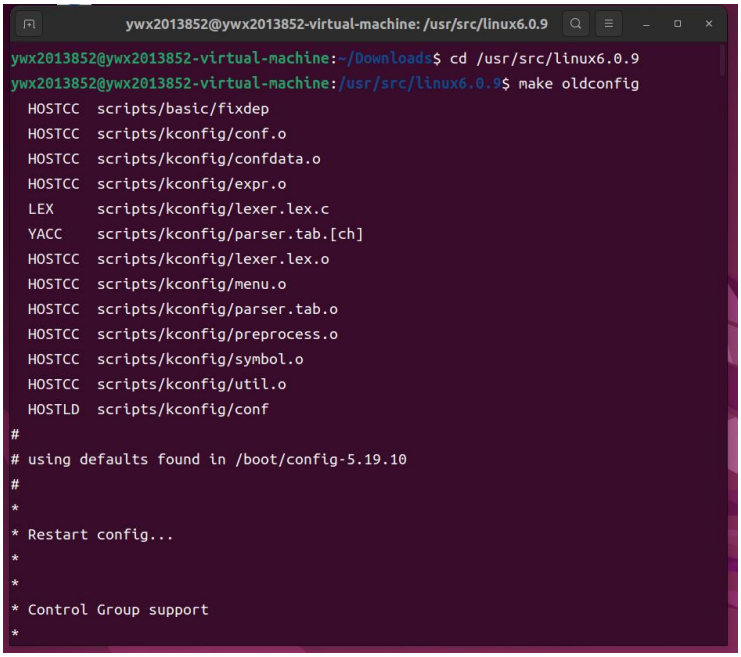
3. 原理方法

对于系统调用和文件拷贝的原理方法不再赘述，在 lab2 和 lab4 中已经有过详细的介绍。

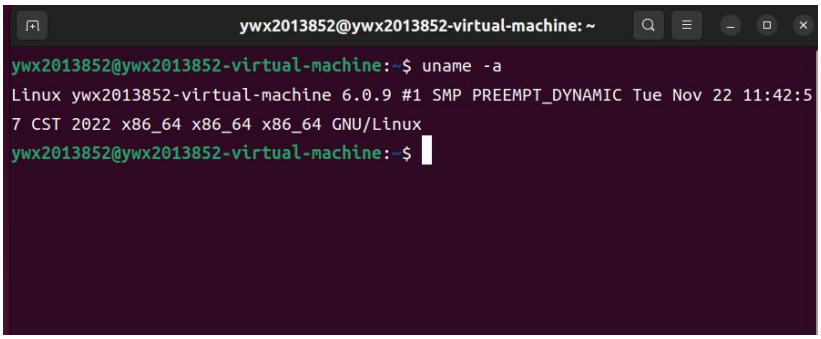
那么，此次需要将系统调用后获得的信息传递到用户程序，那么如何将系统调用的信息传递到用户程序呢？这其实可以在用户程序当中声明一个字符指针，然后在编写系统调用的时候，传入一个字符之指针参数，然后在用户程序访问系统调用时将字符指针进行传递，此时字符指针就可以指到系统调用产生的信息，将得到的信息返回到本用户程序中，因此就可以在用户程序中进行输出等操作。具体实现见具体步骤中。

4. 具体步骤

1、进行最新内核的编译：



最新版本 6.0.9:



2、添加新的系统调用，实现文件拷贝功能和系统进程信息的展示。

原理：拷贝文件：将源文件地址和目标文件地址作为参数传递到自定义的系统调用中，从而通过系统调用实现文件目录拷贝。

获取进程信息：将缓冲区 buffer 作为参数，传递到自定义的系统调用函数当中，在系统调用被使用的过程中，buffer 指向系统调用产生的进程信息，从而实现了系统调用的结果是通过参数返回到用户程序中这一目标。

打开(O) *syscalls.h ~/Documents/linux-5.19.10/include/linux 保存(S)

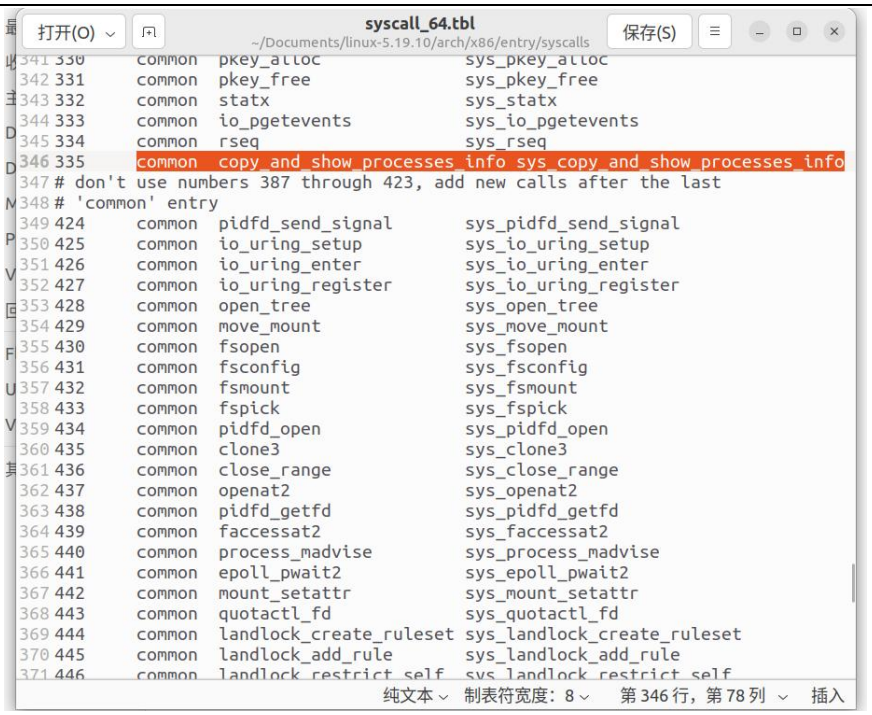
```
1260 #endif
1261
1262 /* obsolete: ipc */
1263 asmlinkage long sys_ipc(unsigned int call, int first, unsigned long second,
1264                         unsigned long third, void __user *ptr, long fifth);
1265
1266 /* obsolete: mm/ */
1267 asmlinkage long sys_mmap_pgoff(unsigned long addr, unsigned long len,
1268                                unsigned long prot, unsigned long flags,
1269                                unsigned long fd, unsigned long pgoff);
1270 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
1271
1272
1273 /*
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
1279 asmlinkage long
    copy_and_show_processes_info(char*,buf,char*,source,char*,target);
1280 #endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
1281
1282
1283 /*
1284  * Kernel code should not call syscalls (i.e., sys_xyzyyz()) directly.
1285  * Instead, use one of the functions which work equivalently, such as
1286  * the ksys_xyzyyz() functions prototyped below.
1287  */
1288 ssize_t ksys_write(unsigned int fd, const char __user *buf, size_t count);
```

C/C++/ObjC 头文件 制表符宽度：8 第 1279 行，第 1 列 插入

打开(O) sys.c ~/Documents/linux-5.19.10/kernel 保存(S)

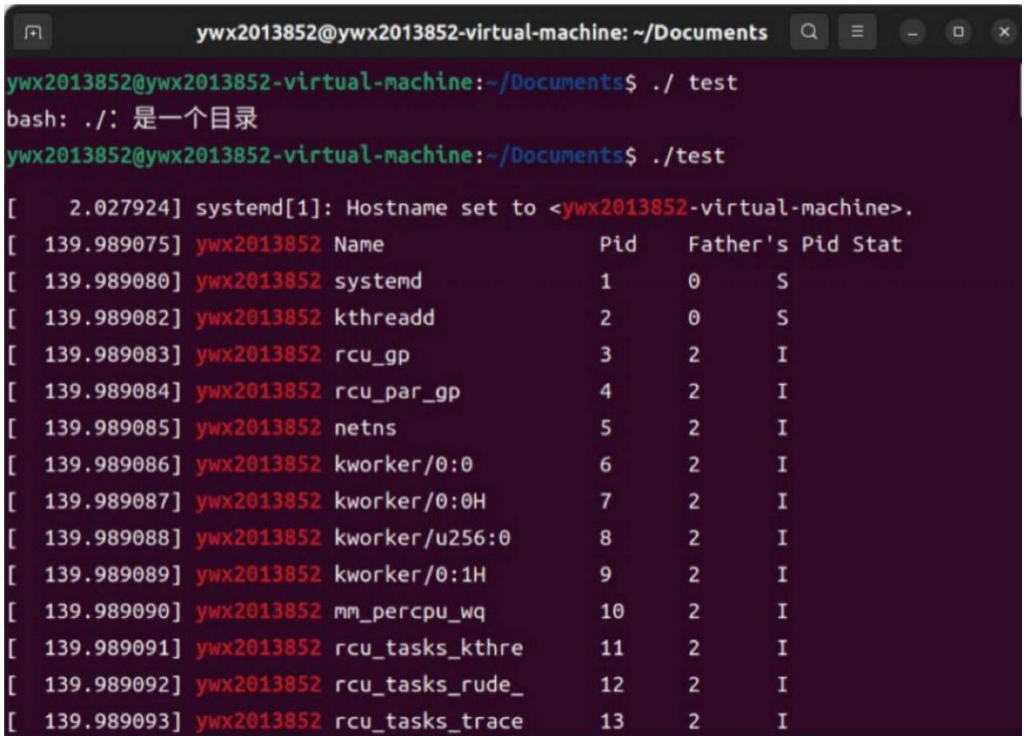
```
948 }
949
950
    SYSCALL_DEFINE0(copy_and_show_processes_info,char*,buf,char*,source,char*,tar
951 {
952
953     struct task_struct *p;    // 进程结构体
954     struct file* src_file = NULL;    //源地址
955     struct file* tgt_file = NULL;    // 目的地址
956     int sum = 0;    // 总进程数
957     int value;
958     int count = 0;
959     loff_t src_pos;
960     loff_t tgt_pos;
961     char buffer[128];    // 字符指针，存储进程信息
962     char* BUF = kmalloc(sizeof(char)*18000,GFP_ATOMIC);
963     char* src = kmalloc(sizeof(char)*30,GFP_ATOMIC);
964     char* tgt = kmalloc(sizeof(char)*30,GFP_ATOMIC);
965     char temp[45];
966     int offset = 0;
967     int i;
968     p = &init_task;
969        // 获取进程信息到缓冲区buffer中
970     for_each_process(p)
971     {
972         memset(temp,'\0',45);
973         sprintf(temp, "%-20s %6d %4c 2013774\n",p->comm,p-
```

C 制表符宽度：8 第 972 行，第 33 列 插入

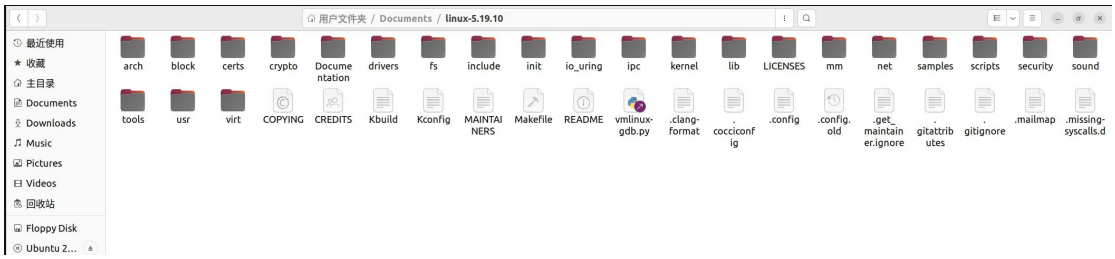


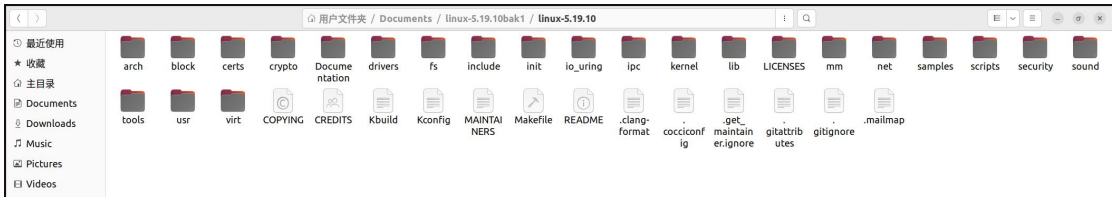
3、编写调用程序，实现文件拷贝和进程信息的展示

原理：char* buffer 缓冲区，作为参数，传递到自定义的系统调用中，最终指向进程信息保存的内存块中，将进程信息返回到了用户程序，进行输出。



4、查看拷贝正确性：





5. 总结心得

此次实验，实现了将系统调用获取的信息通过参数的方式传入到了用户程序区，提供了系统调用与用户程序在内存等方面上面的联系桥梁 。并且，此时的系统调用就像是一个用户自己定义的一个函数没，在用户程序中，可以灵活的调用该"函数"，达到用户想要的效果。因此，我认为，系统调用是一个比较方便，而且是一个比较灵活的工具，在使用自己的 OS 时，可以通过添加多种的系统调用，灵活的封装各种需要的功能，在编写自己的用户程序时，将系统调用与用户程序相结合。

6. 参考资料

源代码：

系统调用：

```
SYSCALL_DEFINE0(copy_and_show_processes_info,char*,buf,char*,source,char*,target)
{

    struct task_struct *p;      // 进程结构体
```

```
        struct file* src_file = NULL;       //源地址
        struct file* tgt_file = NULL;       // 目的地址
        int sum = 0;       // 总进程数
        int value;
        int count = 0;
        loff_t src_pos;
        loff_t tgt_pos;
        char buffer[128];       // 字符指针，存储进程信息
        char* BUF = kmalloc(sizeof(char)*18000,GFP_ATOMIC);
        char* src = kmalloc(sizeof(char)*30,GFP_ATOMIC);
        char* tgt = kmalloc(sizeof(char)*30,GFP_ATOMIC);
        char temp[45];
        int offset = 0;
        int i;
        p = &init_task;
       // 获取进程信息到缓冲区 buffer 中
        for_each_process(p)
        {
                memset(temp,'\0',45);
                sprintf(temp,
"%-10s %-20s %-6d %-6d %-6c\n","ywx2013852",p->comm,p->pid,p->parent->pi
d,task_state_to_char(p)
);
                i = 0;
                while(temp[i]!='\0'){
                        BUF[offset]=temp[i];
                        i++;
                        offset++;
                }
                sum += 1;
        }
        memset(temp,'\0',45);
        sprintf(temp, "the number of processes is %d 2013774\n", sum);
        i = 0;
        while(temp[i]!='\0'){
                BUF[offset]=temp[i];
                i++;
                offset++;
        }
        // 进行拷贝到用户空间
        value = copy_to_user(buf,BUF,offset);
        value = copy_from_user(src,source,30);
        value = copy_from_user(tgt,target,30);
        printk("%s to %s 2013852",src,tgt);
```

```
    // 进行文件拷贝
     src_file = filp_open(src, O_RDWR | O_APPEND | O_CREAT, 0644);
     tgt_file = filp_open(tgt, O_RDWR | O_APPEND | O_CREAT, 0644);
     if (IS_ERR(src_file)) {
          printk("fail to open file 2013774");
          return 0;
     }
     if (IS_ERR(tgt_file)) {
          printk("fail to open file 2013774");
          return 0;
     }
     src_pos = src_file->f_pos;
     tgt_pos = tgt_file->f_pos;
     while((count = kernel_read(src_file,buffer,128,&src_pos))>0)
     {
          kernel_write(tgt_file,buffer,count,&tgt_pos);
     }
     filp_close(src_file,NULL);
     filp_close(tgt_file,NULL);
     kfree(BUF);
     return 0;
}
```

测试程序:

```
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#define _NR_copy_and_show_processes_info 460
int main(int argc, char *argv[])
{
    char *buf;        // 缓冲区存放进程信息
    char *source = "/home/ywx2013852/Documents/linux-5.19.10";    //源地址
    char *target = "/home/ywx2013852/Documents/linux-5.19.10bak";   // 目的地址
    syscall(_NR_copy_and_show_processes_info(buf,source,target));    //系统调用
    printf("%-10s %-20s %-6s %-6s %-6s\n","ywx2013852","Name","Pid","Father's Pid","Stat");
    printf(buf);
    return 0;
}
```