《操作系统综合实验》

实验报告

01 系统基础调用

姓 名　　王 栗 政

班 级　2018039

学 号　20179100018

教 师　冯鹏斌

实验日期　 2022.4.2

一、实验题目

```
  In Section 2.3, we described a program that copies the contents of one file to a destination file. This program works by first prompting the user for the name of the source and destination files. Write this program using either the Windows or POSIX API. Be sure to include all necessary error checking, including ensuring that the source file exists. Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. Linux systems provide the strace utility, and Solaris and Mac OS X systems use the dtrace command. As Windows systems do not provide such features, you will have to trace through the Windows version of this program using a debugger.
```

二、相关原理与知识

1. **系统调用相关知识**

```
#include<sys/types.h>
  一般在/usr/include/sys路径下，是Unix/Linux系统的基本系统数据类型的头文件，含有size_t,time_t,pid_t等类型，Linux编程中常用到的头文件。
  较为常见的几种类型:
    clock_t 表示系统时间（以时钟周期为单位）。
    dev_t 用于设备号。
    off_t 用于文件大小和偏移量。
    ptrdiff_t 是一种带符号整型，用于对两个指针执行减法运算后所得的结果。
    size_t 反映内存中对象的大小（以字节为单位）。
    ssize_t 供返回字节计数或错误提示的函数使用。
    time_t 以秒为单位计时。

   所有这些类型在 ILP32 编译环境中保持为 32 位值，并会在 LP64 编译环境中增长为 64 位值。

#include<sys/stat.h>
  主要函数有:
    int stat(const char *restrict pathname,struct stat *restrict buf);
    int fstat(int fields,struct stat *buf);
    int lstat(const char *restrict pathname,struct stat *restrict buf);
  可以用于获取一个结构体，里面包括文件的全部属性
    struct stat {
       dev_t st_dev; // 文件所在设备ID
       ino_t st_ino; // 结点(inode)编号
       mode_t st_mode; // 保护模式
       nlink_t st_nlink; // 硬链接个数
       uid_t st_uid; // 所有者用户ID
       gid_t st_gid; // 所有者组ID
       dev_t st_rdev; // 设备ID(如果是特殊文件)
       off_t st_size;// 总体尺寸，以字节为单位
       blksize_t st_blksize; // 文件系统 I/O 块大小
```

```
        blkcnt_t st_blocks; // 已分配 512B 块个数
        time_t st_atime; // 上次访问时间
        time_t st_mtime; // 上次更新时间
        time_t st_ctime; // 上次状态更改时间
    };

#include<fcntl.h>
  用于文件操作的系统调用,主要函数如下:
    打开文件  int open( const char * pathname, int flags);
              int open( const char * pathname, int flags);
    关闭文件  int close(int fd);
    读取数据  ssize_t read(int fd,void * buf ,size_t count);
    写入数据  ssize_t write (int fd,const void * buf,size_t count);

#include<unistd.h>
  是POSIX标准定义的unix类系统定义符号常量的头文件，包含了许多UNIX系统服务的函数原型，例如
read函数、write函数和getpid函数
  常用的函数:
    ssize_t      read(int, void *, size_t);
    int          unlink(const char *);
    ssize_t      write(int, const void *, size_t);
    int          usleep(useconds_t);
    unsigned     sleep(unsigned);
    int          access(const char *, int);
    unsigned     alarm(unsigned);
    int          chdir(const char *);
    int          chown(const char *, uid_t, gid_t);
    int          close(int);
    size_t       confstr(int, char *, size_t);
    void         _exit(int);
    pid_t        fork(void);
```

2. **strace命令相关知识**

```
NAME
       strace - trace system calls and signals

SYNOPSIS
       strace [-ACdffhikqqrtttTvVwxxyyzZ] [-I n] [-b execve] [-e expr]...
              [-O overhead] [-S sortby] [-U columns] [-a column] [-o file]
              [-s strsize] [-X format] [-P path]... [-p pid]...
              [--seccomp-bpf] { -p pid | [-DDD] [-E var[=val]]...
              [-u username] command [args] }

       strace -c [-dfwzZ] [-I n] [-b execve] [-e expr]... [-O overhead]
              [-S sortby] [-U columns] [-P path]... [-p pid]...
              [--seccomp-bpf] { -p pid | [-DDD] [-E var[=val]]...
              [-u username] command [args] }
```

三、实验过程

为了模拟cp命令，我们主要通过命令行来传递参数，对于参数数量不符合的输入直接终止程序并给出正确使用方法

```
    if(argc!=3){
        puts("usage: ./mycp source_file_path destination_file_path\n");
        return -1;
    }
```

对于需要拷贝的文件，我们以只读模式打开，并使用fstat()函数获取文件的相关信息并将其存入一个stat结构体中，通过其st_size成员获取其长度，创建一个缓存区，之后通过read函数获取文件内容，在过程中给出提示（报错信息）

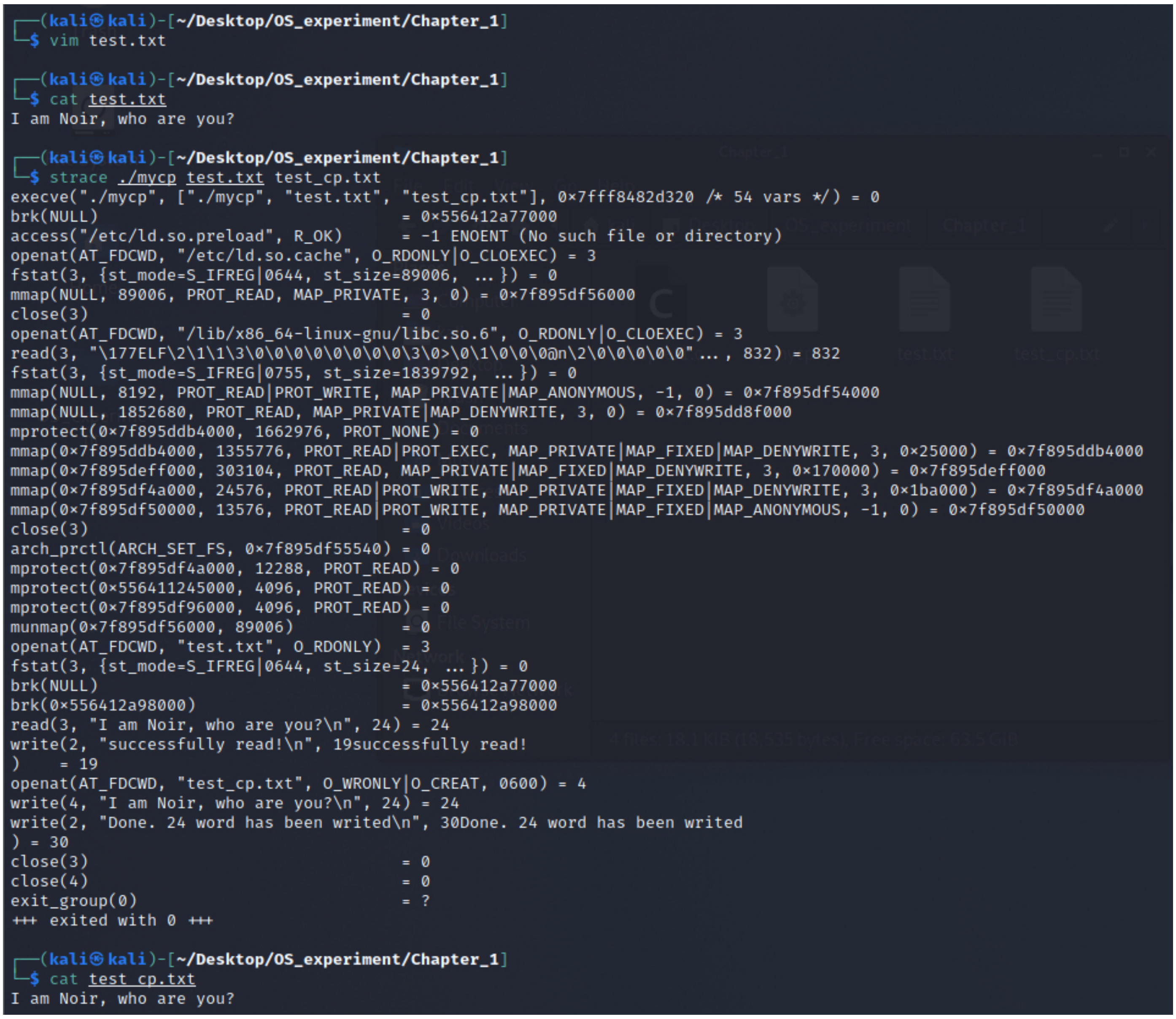
```
    src_file = open(argv[1],O_RDONLY);
    if(src_file == -1){
        printf("Fail to open %s. Please check! \n",argv[1]);
        return -5;
    }
    fstat(src_file,&s); //获取文件属性
    buff = (char*) malloc(sizeof(char) * s.st_size);
    if (buff == NULL)
      {
        puts("Malloc error!");
        return -3;
      }
    ssize_t r = read(src_file, buff, s.st_size);

    if(r == 0 ){
        puts("Nothing in dst_file!\n");
    }else if(r == -1){
        puts("A error happened!\n");
        return -4;
    }else{
        puts("successfully read!\n");
    }

```

对于要写入的文件，则以O_CREAT|O_WRONLY模式打开，（表示以只写模式打开，若没有该文件则创建一个），通过write函数将缓存区buff中的内容写入文件，并给出相关提示信息，最终关闭文件描述符。

```
    dst_file = open(argv[2],O_CREAT|O_WRONLY,0600);
    if(dst_file == -1){
        printf("Can't find %s.\n",argv[2]);
        return -5;
    }

```

```c

    ssize_t w = write(dst_file, buff,s.st_size);
    if(w == -1 ){
        puts("A error happened!\n");
        return -6;
    }

    printf("Done. %d word has been writed\n",s.st_size);
    close(src_file);
    close(dst_file);
```

四、实验结果与分析



```
┌──(kali㉿kali)-[~/Desktop/OS_experiment/Chapter_1]
└─$ vim test.txt

┌──(kali㉿kali)-[~/Desktop/OS_experiment/Chapter_1]
└─$ cat test.txt
I am Noir, who are you?

┌──(kali㉿kali)-[~/Desktop/OS_experiment/Chapter_1]
└─$ strace ./mycp test.txt test_cp.txt
execve("./mycp", ["./mycp", "test.txt", "test_cp.txt"], 0×7fff8482d320 /* 54 vars */) = 0
brk(NULL)                               = 0×556412a77000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=89006, ...}) = 0
mmap(NULL, 89006, PROT_READ, MAP_PRIVATE, 3, 0) = 0×7f895df56000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0@n\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839792, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0×7f895df54000
mmap(NULL, 1852680, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0×7f895dd8f000
mprotect(0×7f895ddb4000, 1662976, PROT_NONE) = 0
mmap(0×7f895ddb4000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0×25000) = 0×7f895ddb4000
mmap(0×7f895deff000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0×170000) = 0×7f895deff000
mmap(0×7f895df4a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0×1ba000) = 0×7f895df4a000
mmap(0×7f895df50000, 13576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0×7f895df50000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0×7f895df55540) = 0
mprotect(0×7f895df4a000, 12288, PROT_READ) = 0
mprotect(0×556411245000, 4096, PROT_READ) = 0
mprotect(0×7f895df96000, 4096, PROT_READ) = 0
munmap(0×7f895df56000, 89006)           = 0
openat(AT_FDCWD, "test.txt", O_RDONLY)  = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=24, ...}) = 0
brk(NULL)                               = 0×556412a77000
brk(0×556412a98000)                     = 0×556412a98000
read(3, "I am Noir, who are you?\n", 24) = 24
write(2, "successfully read!\n", 19successfully read!
)    = 19
openat(AT_FDCWD, "test_cp.txt", O_WRONLY|O_CREAT, 0600) = 4
write(4, "I am Noir, who are you?\n", 24) = 24
write(2, "Done. 24 word has been writed\n", 30Done. 24 word has been writed
) = 30
close(3)                                = 0
close(4)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++

┌──(kali㉿kali)-[~/Desktop/OS_experiment/Chapter_1]
└─$ cat test_cp.txt
I am Noir, who are you?
```

由图中可知虽然我们使用open()函数来打开文件“test.txt”，但是其使用的系统调用为openat()，这是由于我们传入的路径为相对路径而非绝对路径导致的，获取到的文件描述符为3。（0、1、2在创建进程用已经使用，分别为标准输入、标准输出、标准错误）。

打开文件后通过fstat()获取文件属性，后通过read()调用文件描述符3读入文件内容，write()调用文件描述符2（标准错误）输出提示信息。

后又通过openat()打开写入文件“test_cp.txt”获取文件描述符为4，通过write()将内容写入文件，后将完成信息和copy的字符数输出到标准错误。

最终通过close()系统调用将文件描述符关闭。

五、问题总结

在完成实验的过程中，对于很多系统调用不熟悉，只对C的文件操作函数有一定的了解，第一次完成时仅使用了C的文件操作函数，后来通过在网上查阅资料的方式，学到了C的很多系统调用，对代码进行了改进。

六、源代码

1. **Chapter1.c**

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>

int main(int argc, char **argv){
    int src_file, dst_file;
    struct stat s;
    char *buff;
    if(argc!=3){
        puts("usage: ./mycp source_file_path destination_file_path\n");
        return -1;
    }

    src_file = open(argv[1],O_RDONLY);
    if(src_file == -1){
        printf("Fail to open %s. Please check! \n",argv[1]);
        return -5;
    }
    fstat(src_file,&s); //获取文件属性
    buff = (char*) malloc(sizeof(char) * s.st_size);
    if (buff == NULL)
      {
        puts("Malloc error!");
        return -3;
      }
    ssize_t r = read(src_file, buff, s.st_size);

    if(r == 0 ){
        puts("Nothing in dst_file!\n");
    }else if(r == -1){
        puts("A error happened!\n");
        return -4;
    }else{
        puts("successfully read!\n");
    }

    dst_file = open(argv[2],O_CREAT|O_WRONLY,0600);
    if(dst_file == -1){
        printf("Can't find %s.\n",argv[2]);
        return -5;
    }


    ssize_t w = write(dst_file, buff,s.st_size);
    if(w == -1 ){
        puts("A error happened!\n");
        return -6;
    }

    printf("Done. %d word has been writed\n",s.st_size);
    close(src_file);
    close(dst_file);
    return 0;
}
```

2. test.txt

```
I am Noir, who are you?
```