

实验 4 文件复制

班级： 07112002 学号： 1120200822 姓名： 郑子帆

一、实验目的

独立设计并实现一个文件复制命令，熟悉 Linux 文件系统提供的有关文件操作的系统调用，加深对文件系统实现功能的理解。

二、实验内容

1. 在 Linux 系统下实现目录复制命令，新实现的命令命名为"mycopy"。
2. 使用 mycopy 命令能够支持多级目录（子目录）的复制，支持 Linux 下的 soft link 复制。
3. 要求使用命令行参数接受参与文件复制的源文件和目标文件，mycopy 命令的使用方法为： mycopy src dest，其中 src 为源文件，dest 为目标文件。
4. 示例：若源目录 src 结构如图 (a)所示，则运行“mycopy Group dest”后，目录 dest 的结构应如图(b)所示。

Group	dest
__ Paper	__ Paper
__ 1.pdf	__ 1.pdf
__ 2.docx	__ 2.docx
__ Slide	__ Slide
__ x.pptx	__ x.pptx
__ Project	__ Project
__ source code	__ source code
__ code.rar	__ code.rar
__ documents	__ documents
__ doc.zip	__ doc.zip
(a)	(b)

5. 在 Linux 系统下可使用 mkdir、opendir、readdir、symlink、readlink 等函数。

三、实验步骤

3.1 实验环境

1. 硬件配置: Intel(R)Core(TM)i7-10750H CPU @ 2.60GHz, 6 个内核
2. 宿主机操作系统: Microsoft Windows 10 家庭中文版
3. 虚拟机操作系统: Ubuntu 22.04 LTS
4. 代码编辑器: Text Editor
5. 编译器: MinGW64 gcc

3.2 实验思路

编写程序 `mycopy.cpp` 完成 linux 系统下的文件复制任务。

首先, 程序中要判断源文件夹是否存在, 若不存在则报告错误, 终止程序; 其次, 判断目标文件夹是否在源文件夹中, 若在, 则文件复制时可能会导致无限循环, 报告错误, 终止程序。

复制时, 先判断, 如果是文件, 则如果是目录, 则新建文件夹, 递归调用文件复制。

最后, 自定义 `mycopy` 命令, 通过更改环境变量使得 `mycopy` 命令可以在终端使用。

3.3 数据结构介绍

3.3.1 文件系统中文件属性结构体

具体定义如下:

```
struct stat {  
    mode_t    st_mode;        // 文件对应的模式, 文件, 目录等  
    ino_t     st_ino;         // inode 节点号  
    dev_t     st_dev;         // 设备号码  
    dev_t     st_rdev;        // 特殊设备号码  
    nlink_t   st_nlink;       // 文件的连接数  
    uid_t     st_uid;         // 文件所有者  
    gid_t     st_gid;         // 文件所有者对应的组  
    off_t     st_size;        // 普通文件, 对应的文件字节数  
    time_t    st_atime;       // 文件最后被访问的时间  
    time_t    st_mtime;       // 文件内容最后被修改的时间  
    time_t    st_ctime;       // 文件状态改变时间
```

```

    blksize_t  st_blksize;    // 文件内容对应的块大小
    blkcnt_t   st_blocks;    // 文件内容对应的块数量
};

```

其中部分成员介绍如下：

`st_mode`: 文件对应的模式。

`st_dev`: 该文件所存放在的设备。

`st_rdev`: 该文件的大小（以字节为单位）。

`st_blocks`: 给该文件分配的块的数量。

`st_blksize`: 该字段提供倾向的块的大小，为了使得该文件的 I/O 可以高效进行。

`st_atime`: 文件最后被访问的时间，该字段在该文件被访问时被修改。

`st_mtime`: 文件内容最后被修改的时间，该字段在文件被修改时会发生改变。

`st_ctime`: 文件状态改变时间，该字段在写入或设置 `inode` 节点信息时发生改变。

一些用到 `st_mode` 的宏：

`S_ISLNK(st_mode)`: 判断是否是一个链接

`S_ISREG(st_mode)`: 判断是否是一个常规文件

`S_ISDIR(st_mode)`: 判断是否是一个目录

`S_ISCHR(st_mode)`: 判断是否是一个字符设备

`S_ISBLK(st_mode)`: 判断是否是一个块设备

`S_ISFIFO(st_mode)`: 判断是否是一个 FIFO 文件

`S_ISSOCK(st_mode)`: 判断是否是一个 SOCKET 文件

3.3.2 timeval 结构体

`timeval` 是 Linux 系统中定义的用于保存时间的结构体，具体定义如下：

```

struct timeval{
    __time_t tv_sec;
    __suseconds_t tv_usec;
};

```

其中成员介绍如下：

`tv_sec`: Epoch 到创建 `struct timeval` 时的秒数。

`tv_usec`: 微秒数，即秒后面的零头。

3.3.3 utimbuf 结构体

`utimbuf` 结构体中存放了文件的修改时间，它是 `utime` 函数的参数，而上面的 `timeval` 是 `utimes` 中的参数，具体定义如下：

```
struct utimbuf{
    time_t actime;
    time_t modtime;
};
```

3.3.4 DIR 结构体

DIR 结构体类似于 FILE，是一个内部结构。DIR 结构体定义如下：

```
struct __dirstream{
    void *__fd;
    char *__data;
    int __entry_data;
    char *__ptr;
    int __entry_ptr;
    size_t __allocation;
    size_t __size;
    __libc_lock_define(, __lock)
};
typedef struct __dirstream DIR;
```

3.3.5 dirent 结构体

dirent 结构体包含了其他文件的名称以及指向与这些文件有关的信息的指针，具体定义如下：

```
struct dirent{
    ino_t d_ino;
    off_t d_off;
    signed short int d_reclen;
    unsigned char d_type;
    char d_name[256];
};
```

其中成员介绍如下：

d_ino: 该文件进入点的索引节点号 inode。
d_off: 目录文件开头至此文件进入点的位移。
d_reclen: 文件名长度
d_type: 文件类型
d_name: 文件名。

3.4 系统调用函数介绍

3.4.1 access

`access` 函数可以判断用户是否具有访问某个文件的权限（或判断某个文件是否存在），具体定义如下：

```
int access(  
    const char *pathname,  
    int mode  
);
```

参数介绍：

`pathname`: 表示要测试的文件的路径

`mode`: 表示测试的模式可能的值有：

`R_OK`: 是否具有读权限

`W_OK`: 是否具有可写权限

`X_OK`: 是否具有可执行权限

`F_OK`: 文件是否存在

返回值：若测试成功则返回 0， 否则返回 -1。

3.4.2 realpath 函数

`realpath` 函数是将相对路径转换成绝对路径的函数，具体定义如下：

```
char *realpath(  
    const char *path,  
    char *resolved_path  
);
```

参数介绍：

`path`: 相对路径。

`resolved_path`: 绝对路径。

返回值：

若成功则返回指向 `resolved_path` 的指针，失败则返回 `NULL`。

3.4.3 lstat

`lstat` 函数可以获取文件相关的信息，并将这些信息保存在 `stat` 结构体中，具体定义如下：

```
int lstat(  
    const char *path,
```

```
    struct stat *buf  
);
```

参数介绍：

path: 文件路径名。

buf: stat 结构体指针。

返回值：若成功执行，返回 0；失败返回-1。

另：stat 函数与 lstat 函数功能类似，当被获取文件不是链接文件时，二者没有区别，如果是链接文件，则 stat 函数具有穿透力，即直接取到链接文件指向的被链接文件的信息；而 lstat 函数不具有穿透力，只获取链接文件本身的信息。

3.4.4 readlink

readlink 函数通过链接的路径将链接的内容复制到缓冲区中，具体定义如下：

```
int readlink(  
    const char *path,  
    struct stat *buf  
);
```

参数介绍：

path: 链接的路径。

buf: 缓冲区指针。

bufsiz: 缓冲区大小限制。

返回值：当该函数调用成功时，该函数返回放置在 buf 中的字节的数量；若调用失败，则返回-1。

3.4.5 symlink

symlink 函数在指定路径新建一个目标链接文件，并将上面缓冲区的内容复制到其中，具体定义如下：

```
int symlink(  
    const char *path1,  
    const char *path2  
);
```

参数介绍：

path1: 缓冲区指针。

path2: 目标链接文件路径。

3.4.6 lutimes

`lutimes` 函数将新建立的链接文件的时间信息设置为旧文件的时间信息，具体定义如下：

```
int lutimes(  
    const char *filename,  
    const struct timeval tv[2]  
);
```

参数介绍：

filename: 链接文件的路径。

tv: 保存了访问时间和修改时间。**tv[0]**存的访问时间，**tv[1]**存的修改时间。

另：同上面的 `stat` 和 `lstat` 的区别，`lutimes` 相比 `utimes` 对于链接文件没有穿透能力。

3.4.7 open

`open` 函数以只读的方式打开源文件，具体定义如下：

```
int open(  
    const char *pathname,  
    int oflag[,mode_t mode]  
);
```

参数介绍：

pathname: 该文件的路径名。

oflag: 该参数可取的值有很多，在本实验中只用到了 `O_RDONLY`（打开并只读）。

返回值：正确打开，返回文件的描述符；否则返回-1。

3.4.8 creat

`creat` 在指定路径建立目标文件。

```
int creat(  
    const char *pathname,  
    mode_t mode  
);
```

参数介绍：

pathname: 目标文件路径。

mode: 在创建文件时指定文件权限，在本实验中，保证跟源文件权限相同。

返回值：正确创建，返回文件的描述符；否则返回-1。

3.4.9 read & write

read 和 write 函数从文件读/写入文件，具体定义如下：

```
ssize_t read(  
    int fd,  
    void *buf,  
    size_t nbytes  
);  
  
ssize_t write(  
    int fd,  
    const void *buf,  
    size_t nbytes  
);
```

参数介绍：

fd: 指定的文件指针。

buf: 所读的数据保存在的缓冲区地址。

nbyte: 请求读的字节数。

3.4.10 close

close 函数为关闭文件，具体定义如下：

```
int close(  
    int fd  
);
```

参数介绍：

fd: 文件的描述符。

返回值：成功返回 0；失败返回-1。

3.4.11 mkdir

mkdir 在新路径名处建立与旧路径访问权限相同的文件夹，具体定义如下：

```
int mkdir(  
    const char *pathname,  
    mode_t mode
```



```
);
```

参数介绍：

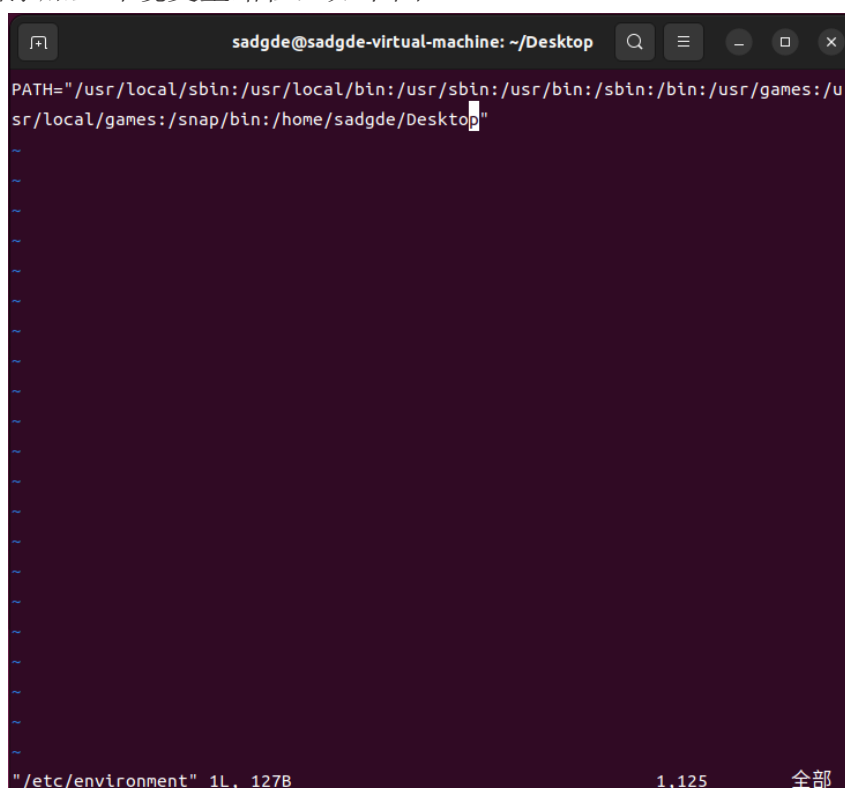
pathname: 文件夹路径

mode: 文件夹访问权限。

返回值：如果调用成功则返回 **0**；失败则返回 **-1**。

3.5 命令设置

完成代码编写后，目前只能通过 `./mycopy` 进行运行，为了添加 `mycopy` 命令，我们需要对环境变量进行添加。因为我将编译好的可执行文件放在了桌面，故将桌面添加至环境变量路径，如下图：



A terminal window titled 'sadgde@sadgde-virtual-machine: ~/Desktop'. The terminal shows the current PATH environment variable: `PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/sadgde/Desktop"`. The cursor is at the end of the line. The terminal has a dark purple background with light blue text. At the bottom, it shows `"/etc/environment" 1L, 127B` and `1,125` with a '全部' (All) button.

四、实验结果及分析

4.1 代码分解

此处仅展示实现主要功能的代码，完整代码详见 `mycopy.cpp` 文件。

4.1.1 check_path_valid 函数

该函数判断目标文件/文件夹路径是否包含在源文件/文件夹路径中，若是，返回 **1**；否则返回 **0**。

```
1. bool check_path_valid(char *sourceFile, char *destinationFile){
2.     char fullSrcPath[FILE_LEN] = {0};
3.     char fullDestPath[FILE_LEN] = {0};
4.     // 得到完整文件路径
5.     realpath(sourceFile, fullSrcPath);
6.     realpath(destinationFile, fullDestPath);
7.     printf("SourceFullPath: %s\n", fullSrcPath);
8.     printf("DestinationFullPath: %s\n", fullDestPath);
9.
10.    bool flag = true;
11.    for(int i = 0; i < FILE_LEN && fullSrcPath[i]; i++){
12.        if(fullSrcPath[i] != fullDestPath[i]){
13.            flag = false;
14.            break;
15.        }
16.    }
17.    return flag;
18.}
```

4.1.2 mycopy_main 函数

该函数用来判断当前路径下的文件类型（链接、文件夹、文件），并调用对应的复制函数。

```
1. void mycopy_main(char *source_path, char *dest_path){
2.     struct stat tmp_stat;
3.     lstat(source_path, &tmp_stat);
4.
5.     if(S_ISLNK(tmp_stat.st_mode)){ // 如果是链接
6.         copy_link(source_path, dest_path, &tmp_stat);
7.     }
8.     else if(S_ISDIR(tmp_stat.st_mode)){ // 如果是文件夹
9.         copy_dir(source_path, dest_path, &tmp_stat);
10.    }
11.    else{ // 如果是文件
12.        copy_file(source_path, dest_path, &tmp_stat);
13.    }
14.}
```

4.1.3 StoreTime 函数

该函数将文件中的访问时间、修改时间存到 `timeval` 结构体中。

```
1. void StoreTime(timeval *time_store, struct stat stat_store){
2.     time_store[0].tv_sec = stat_store.st_atime; // 存访问时间
```

```
3. time_store[1].tv_sec = stat_store.st_mtime; // 存修改时间
4. time_store[0].tv_usec = 0;
5. time_store[1].tv_usec = 0;
6. }
```

4.1.4 copy_link 函数

该函数完成了软链接的复制。

```
1. void copy_link(char *sourceRoot, char *destRoot, struct stat *t
   mp_stat){
2.   char linkBuff[Buf_Size] = {0};
3.   readlink(sourceRoot, linkBuff, sizeof(linkBuff)); // 将源链接信
   息存入缓冲区
4.   symlink(linkBuff, destRoot); // 将缓冲区信息写入目标链接
5.   timeval timeTmp[2];
6.   StoreTime(timeTmp, *tmp_stat); // 存原链接中的时间戳写入目标
   链接
7.   int succ = lutimes(destRoot, timeTmp);
8.   if(succ == -1) {
9.     perror("Failed to change the time of the link!\n");
10.  }
11.  else {
12.    printf("Successfully copy the link: %s\n", sourceRoot);
13.  }
14. }
```

4.1.5 copy_dir 函数

该函数完成了文件夹的复制。

```
1. void copy_dir(char *sourceRoot, char *destRoot, struct stat *tm
   p_stat){
2.   mkdir(destRoot, tmp_stat -> st_mode); // 在目标路径新建一个权限
   相同的文件夹
3.   DIR *pSearchDir = opendir(sourceRoot);
4.   struct dirent *pSingleDir;
5.   while((pSingleDir = readdir(pSearchDir)) != NULL){ // 通过
   readdir 函数检索每一项目录信息
6.     char dirName[FILE_LEN] = {0};
7.     strcpy(dirName, pSingleDir -> d_name);
8.     // 一个目录中有两个子目录，分别指向自己和它的父级，对于这两个我们不
   用复制
9.     if(!strcmp(dirName, ".") || !strcmp(dirName, "..")) continue;
10.  }
```

```
11. char sourceTmp[FILE_LEN] = {0};
12. char destTmp[FILE_LEN] = {0};
13. // 得到下一级的路径, 并递归完成复制
14. sprintf(sourceTmp, "%s/%s", sourceRoot, dirName);
15. sprintf(destTmp, "%s/%s", destRoot, dirName);
16.
17. mycopy_main(sourceTmp, destTmp);
18. }
19.
20. struct utimbuf timeDir;
21. timeDir.actime = tmp_stat -> st_atime;
22. timeDir.modtime = tmp_stat -> st_mtime;
23. utime(destRoot, &timeDir); // 更改文件时间戳
24. printf("Successfully copy the dir: %s\n", sourceRoot);
25. }
```

4.1.6 copy_file 函数

该函数完成了文件的复制。

```
1. void copy_file(char *sourceRoot, char *destRoot, struct stat *tmp_stat){
2.     int srcFd = open(sourceRoot, O_RDONLY); // 只读打开
3.     int destFd = creat(destRoot, tmp_stat -> st_mode); // 在目标路径新建一个权限相同的文件
4.
5.     int bufferSize = 0;
6.     char text_buffer[Buf_Size];
7.     // 循环写入缓冲区, 并写给目标路径的文件
8.     while((bufferSize = read(srcFd, text_buffer, Buf_Size)) > 0){
9.         write(destFd, text_buffer, bufferSize);
10.    }
11.
12.    struct utimbuf timeTmp;
13.    timeTmp.actime = tmp_stat -> st_atime;
14.    timeTmp.modtime = tmp_stat -> st_mtime;
15.    utime(destRoot, &timeTmp); // 更改文件的时间戳
16.    // 关闭文件
17.    close(srcFd);
18.    close(destFd);
19. }
```

4.2 运行结果截图

复制文件夹：

```
sadgde@sadgde-virtual-machine:~/Desktop$ mycopy source dest
Initializing...
Source File exist.
SourceFullPath:           /home/sadgde/Desktop/source
DestinationFullPath:       /home/sadgde/Desktop/dest
Destination fath verification complete.
Successfully copy the file: source/testdir/source code/code.zip
Successfully copy the dir:  source/testdir/source code
Successfully copy the dir:  source/testdir/documents
Successfully copy the dir:  source/testdir/Slide
Successfully copy the file: source/testdir/x.pptx
Successfully copy the file: source/testdir/1.pdf
Successfully copy the dir:  source/testdir/Paper
Successfully copy the file: source/testdir/doc.zip
Successfully copy the dir:  source/testdir/Project
Successfully copy the file: source/testdir/2.docx
Successfully copy the dir:  source/testdir
Successfully copy the file: source/testfile
Successfully copy the link:  source/testlinkfile
Successfully copy the link:  source/testlinkdir
Successfully copy the dir:  source
Copy Finished in 0.00598 seconds.
```

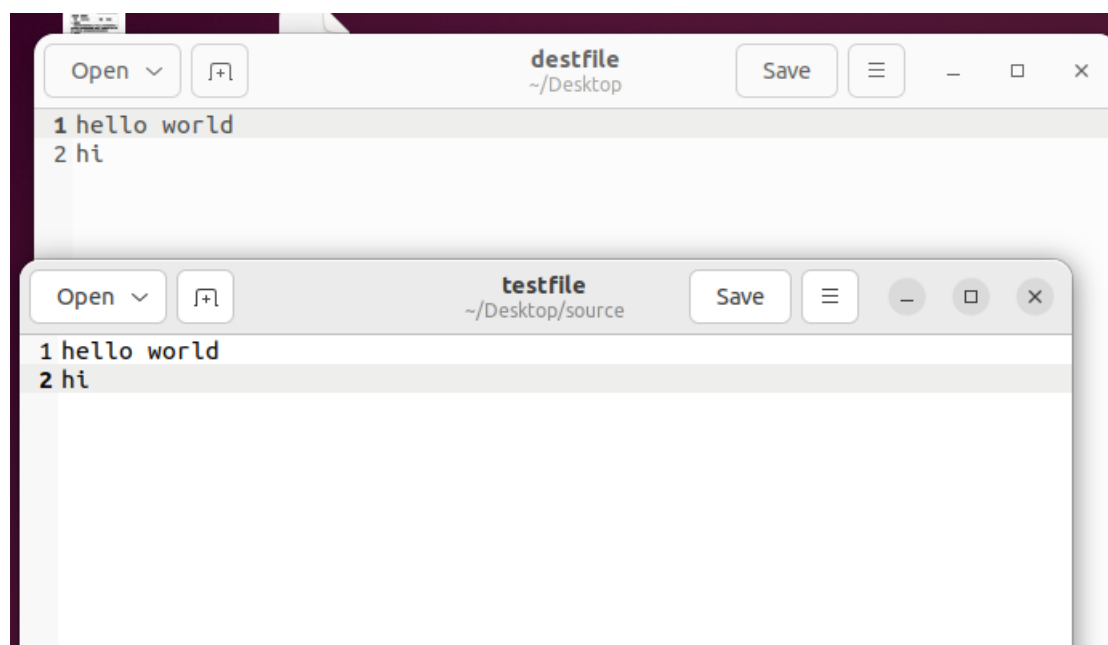
复制文件：

```
sadgde@sadgde-virtual-machine:~/Desktop$ mycopy source/testfile destfile
Initializing...
Source File exist.
SourceFullPath:           /home/sadgde/Desktop/source/testfile
DestinationFullPath:       /home/sadgde/Desktop/destfile
Destination fath verification complete.
Successfully copy the file: source/testfile
Copy Finished in 0.00011 seconds.
```

检查复制的内容对比：

```
sadgde@sadgde-virtual-machine:~/Desktop$ ls -l
total 892
drwxrwxr-x 3 sadgde sadgde 4096 11月 27 16:23 dest
-rw-rw-r-- 1 sadgde sadgde 15 11月 27 16:23 destfile
-rwxrwxr-x 1 sadgde sadgde 21312 11月 27 23:03 mycopy
-rw----- 1 sadgde sadgde 5203 11月 27 23:03 mycopy.cpp
-rw----- 1 sadgde sadgde 3713 11月 27 18:00 mycopy_wang.cpp
-rw----- 1 sadgde sadgde 863915 12月 25 2019 report.pdf
drwxrwxr-x 3 sadgde sadgde 4096 11月 27 16:23 source
sadgde@sadgde-virtual-machine:~/Desktop$ ls -l source
total 8
drwxrwxr-x 7 sadgde sadgde 4096 11月 27 16:38 testdir
-rw-rw-r-- 1 sadgde sadgde 15 11月 27 16:23 testfile
lrwxrwxrwx 1 sadgde sadgde 7 11月 27 16:06 testlinkdir -> testdir
lrwxrwxrwx 1 sadgde sadgde 8 11月 27 16:20 testlinkfile -> testfile
sadgde@sadgde-virtual-machine:~/Desktop$ ls -l dest
total 8
drwxrwxr-x 7 sadgde sadgde 4096 11月 27 16:38 testdir
-rw-rw-r-- 1 sadgde sadgde 15 11月 27 16:23 testfile
lrwxrwxrwx 1 sadgde sadgde 7 11月 27 16:06 testlinkdir -> testdir
lrwxrwxrwx 1 sadgde sadgde 8 11月 27 16:20 testlinkfile -> testfile
```

文件内容对比：



可以看到，不管是文件还是文件夹还是软链接，都完成了一模一样的复制，内容、时间戳均成功完成了复制。

五、实验收获与体会

通过这次实验，结合理论课的知识 and 网上关于 **linux** 系统下的有关文件属性信息获取、修改的函数和结构体信息，完成了对于软链接、文件、文件夹的复制的实践。学习的过程比较漫长痛苦，但是能够成功进行复制之后还是很开心的。这次实验过程比较顺利，可能因为没有涉及到一些比较恶心人的类型转换的问题，所以没有遇到什么问题。

附录：程序清单及说明

1120200822-郑子帆-07112002-实验 4 文件复制.pdf: 实验报告

mycopy.cpp 文件: **linux** 系统下文件复制实现代码

实验运行录屏.mp4: 运行结果的录屏文件