操作系统课程设计实验报告

实验名称：复制文件

姓名/学号：曾煜瑾/1120172765

一、 实验目的

1. 了解在 Windows 中，文件系统如何保存在磁盘、光盘等存储介质上的信息。并通过文件系统提供的各种 API，对文件进行同步和异步读写，深入理解 Windows 文件系统的功能和作用，掌握同步 I/O 和异步 I/O 的特点

2. 熟悉 Linux 文件系统提供的有关文件操作的系统调用。文件系统是使用计算机信息系统的重要接口。通过使用文件系统的系统调用命令操作文件，以达到对文件系统实现功能的理解和掌握

二、 实验内容

完成一个目录复制命令 mycp，包括目录下的文件和子目录，运行结果如下：

```
beta@bugs.com [~/]# ls –la sem
total 56
drwxr-xr-x  3 beta beta 4096 Dec 19 02:53 ./
drwxr-xr-x  8 beta beta 4096 Nov 27 08:49 ../
-rw-r--r--  1 beta beta  128 Nov 27 09:31 Makefile
-rwxr-xr-x  1 beta beta 5705 Nov 27 08:50 consumer*
-rw-r--r--  1 beta beta  349 Nov 27 09:30 consumer.c
drwxr-xr-x  2 beta beta 4096 Dec 19 02:53 subdir/
beta@bugs.com [~/]# mycp sem target
beta@bugs.com [~/]# ls –la target
total 56
drwxr-xr-x  3 beta beta 4096 Dec 19 02:53 ./
drwxr-xr-x  8 beta beta 4096 Nov 27 08:49 ../
-rw-r--r--  1 beta beta  128 Nov 27 09:31 Makefile
-rwxr-xr-x  1 beta beta 5705 Nov 27 08:50 consumer*
-rw-r--r--  1 beta beta  349 Nov 27 09:30 consumer.c
drwxr-xr-x  2 beta beta 4096 Dec 19 02:53 subdir/
```

说明：

Linux：creat, read, write 等系统调用，要求支持软链接

Windows：CreateFile(), ReadFile(), WriteFile(), CloseHandle()等函数

特别注意复制后，不仅权限一致，而且时间属性也一致。

三、实验环境

	名称	版本
Windows 操作系统	Windows 10	企业版
Windows IDE	VSCode	1.40.1.0
Linux 操作系统	Ubuntu 16.04	内核 4.13.0
Linux IDE	gcc	5.4.0

四、程序设计与实现

设计思路：

无论是 Windows 系统还是 Linux 系统，都是先判断输入参数的合理性，如果不合理给出相应提示，如果合理，则先建立一个文件夹。之后遍历源目录的每一项，如果是子目录，则建立一个同名目录，递归进入该目录直到找到单个文件；如果是单个文件，则调用读文件函数和写文件函数，将源文件内容写入目标文件。另外，对创建的每个目录和文件，都更新其权限和时间同复制源文件一致。

Windows 版本：

（1）API 介绍：

1. FindFirstFile(): 根据文件名查找文件

   FindNextFile(): 判断当前目录下是否有下一个目录或文件

   头文件：Windows.h

   函数声明：

```
HANDLE FindFirstFileA(
    LPCSTR     lpFileName,
    LPWIN32_FIND_DATAA     lpFindFileData
);
BOOL FindNextFileA(
    HANDLE    hFindFile,
    LPWIN32_FIND_DATAA    lpFindFileData
);
```

说明：

lpFileName: 目录或路径以及文件名，文件名可以包含通配符

lpFindFileData: 指向 WIN32_FIND_DATA 结构的指针，该结构接收相关找到的文件或目录的信息

返回值：

• 若函数成功，则返回值用于 FindNextFile 和 FindClose 使用的句柄，而 lpFindFileData 包含找到文件或目录的信息

• 若函数失败，则返回值为 INVALID_HANDLE_VALUE

2. CreateFile(): 创建或打开一个文件，返回可访问的句柄

头文件：Windows.h

函数声明为：

```
HANDLE CreateFile(
    LPCTSTR lpFileName, //普通文件名或者设备文件名
    DWORD dwDesiredAccess, //访问模式（写/读）
    DWORD dwShareMode, //共享模式
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, //指向安全属性的指针
    DWORD dwCreationDisposition, //如何创建
    DWORD dwFlagsAndAttributes, //文件属性
    HANDLE hTemplateFile //用于复制文件句柄
);
```

返回值：

• 若函数成功，则返回指定文件的打开句柄

• 若函数失败，则返回 INVALID_HANDLE_VALUE

3. ReadFile(): 从文件指针指向的位置开始将数据读出到一个文件中

头文件：Windows.h

函数声明为：

```
BOOL ReadFile(
    HANDLE hFile, //文件的句柄
    LPVOID lpBuffer, //用于保存读入数据的一个缓冲区
    DWORD nNumberOfBytesToRead, //要读入的字节数
    LPDWORD lpNumberOfBytesRead, //指向实际读取字节数的指针
```

LPOVERLAPPED lpOverlapped

//定义了一次异步读取操作。否则，应将这个参数设为 NULL

);

返回值：

- 若函数成功，则返回非零（TRUE）
- 若函数失败，则返回零（FALSE）

4. WriteFile(): 将数据写入一个文件中

头文件：Windows.h

函数声明为：

BOOL WriteFile(

HANDLE hFile, //文件句柄

LPCVOID lpBuffer, //数据缓存区指针

DWORD nNumberOfBytesToWrite, //你要写的字节数

LPDWORD lpNumberOfBytesWritten, //用于保存实际写入字节数的存储区域的指针

LPOVERLAPPED lpOverlapped//OVERLAPPED 结构体指针

);

返回值：

- 若函数成功，则返回非零（TRUE）
- 若函数失败，则返回零（FALSE）

5. SetFileTime(): 修改文件的时间属性

函数声明为：

BOOL SetFileTime(

HANDLE hFile, //要修改文件的句柄

const FILETIME *lpCreationTime, //文件创建时间

const FILETIME *lpLastAccessTime, //文件最后访问时间

const FILETIME *lpLastWriteTime //文件最后写入时间

);

返回值：

- 若函数成功，则返回非零（TRUE）
- 若函数失败，则返回零（FALSE）

6. SetFileAttributes(): 修改文件的权限属性

函数声明为：

BOOL SetFileAttributes(

LPCTSTR lpFileName, //文件名

DWORD dwFileAttributes //权限值

);

返回值：

- 若函数成功，则返回非零（TRUE）
- 若函数失败，则返回零（FALSE）

（2）代码分析：

1. 判断输入参数是否正确

三重检验，否则给出相应提示：

1.1. 输入参数必须为 3 个，如：mycp sourceDir destinationDir

```
if (argc != 3)
{
    printf("Please enter valid parameters !\n");
    printf("For example: mycp sourceDir destinationDir\n");
}
```

1.2. 源文件夹必须存在

```
if (FindFirstFile(argv[1], &lpfindfiledata) == INVALID_HANDLE_VALUE)
{
    printf("Can not find the source directory\n");
}
```

1.3. 目标文件夹必须不存在

```
if (FindFirstFile(argv[2], &lpfindfiledata) != INVALID_HANDLE_VALUE)
{
    printf("The target directory is already exists");
}
```

2. 判断是否是文件夹

```
HANDLE hfind = FindFirstFile(source, &lpfindfiledata);//数据缓冲区
while (FindNextFile(hfind, &lpfindfiledata) != 0)//循环查找FindFirstFile()函数搜索后的下一个文件
{
    //查找下一个文件成功
    if ((lpfindfiledata.dwFileAttributes) == 16)//判断是否是目录(若为目录FILE_ATTRIBUTE_DIRECTORY是16)
    {
```

3. 如果是单个文件

```
void copyFile(const char *sourceFile, const char *destinationFile){
else//无目录
{                                                   pfilefinddata);
    memset(source, '0', sizeof(source));
    lstrcpy(source, fsource);                       ,
    lstrcat(source, "\\");                          LL);
    lstrcat(source, lpfindfiledata.cFileName);      File,
    lstrcat(target, lpfindfiledata.cFileName);      ,
    CopyFile(source, target);//直接调用文件复制函数  NULL);
    lstrcpy(source, fsource);                       filefinddata.nFileSizeHigh;
    lstrcat(source, "\\");
    lstrcpy(target, ftarget);
    lstrcat(target, "\\");                          L);
}                                                   d, NULL);
```

4. 复制单个文件

```
void copyFile(const char *sourceFile, const char *destinationFile){
    WIN32_FIND_DATA lpfilefinddata;
    HANDLE hfirstfile = FindFirstFile(sourceFile, &lpfilefinddata);
    HANDLE hsourcefile = CreateFile(sourceFile,
        GENERIC_READ | GENERIC_WRITE,FILE_SHARE_READ,
        NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    HANDLE hdestinationfile = CreateFile(destinationFile,
        GENERIC_READ | GENERIC_WRITE,FILE_SHARE_READ,
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    LONG filesize = lpfilefinddata.nFileSizeLow - lpfilefinddata.nFileSizeHigh;
    DWORD word;
    int *buff = new int[filesize];
    ReadFile(hsourcefile, buff, filesize, &word, NULL);
    WriteFile(hdestinationfile, buff, filesize, &word, NULL);
```

5. 修改单个文件的权限属性

```
DWORD dwAttrs = GetFileAttributes(fsource);
SetFileAttributes(ftarget, dwAttrs);
```

6. 修改单个文件的时间属性

```
SetFileTime(htarget,//指定文件的句柄
    &lpfindfiledata.ftCreationTime,//文件创建时间
    &lpfindfiledata.ftLastAccessTime,//文件最后访问时间
    &lpfindfiledata.ftLastWriteTime);//文件最后写入时间
```

7. 修改新建文件夹的时间属性

```
CreateDirectory(target,NULL);//为目标文件创建目录
mycp(source, target);//进入子目录复制

HANDLE hDir = CreateFile(target,
    GENERIC_READ|GENERIC_WRITE,
    FILE_SHARE_READ,
    NULL,
    OPEN_EXISTING,
    FILE_FLAG_BACKUP_SEMANTICS,
    NULL);

SetFileTime(hDir,
    &lpfindfiledata.ftCreationTime,
    &lpfindfiledata.ftLastAccessTime,
    &lpfindfiledata.ftLastWriteTime);

CloseHandle(hDir);
```

（3）运行效果

1. 准备工作

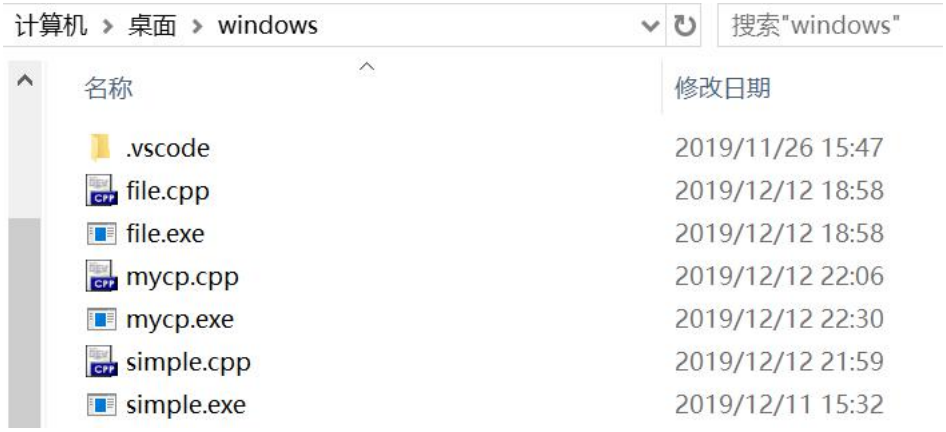
1.1. 待复制文件夹 zzz

说明：zzz 文件夹中的所有文件均为存档文件属性

1.2. 使用命令行为 testA.docx 文件设置只读属性

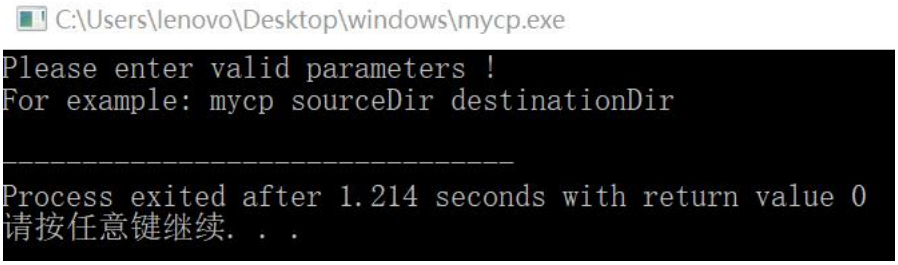
```
C:\Users\lenovo\Desktop\zzz>attrib testA.docx
A                   C:\Users\lenovo\Desktop\zzz\testA.docx

C:\Users\lenovo\Desktop\zzz>attrib +r testA.docx

C:\Users\lenovo\Desktop\zzz>attrib testA.docx
A    R              C:\Users\lenovo\Desktop\zzz\testA.docx
```

1.3. 未复制 zzz 前的目标文件夹 windows



2. 测试输入合理性

2.1. 编译源代码



2.2. 输入参数数量错误

```
C:\Users\lenovo\Desktop\windows>mycp 2
Please enter valid parameters !
For example: mycp sourceDir destinationDir
```

2.3. 原文件夹不存在

```
C:\Users\lenovo\Desktop\windows>mycp 2 copyright
Can not find the source directory
```

2.4. 目标文件夹已存在（.vscode 已经存在）

```
C:\Users\lenovo\Desktop\windows>mycp ../zzz .vscode
The target directory is already exists
```

3. 输入正确格式，完成复制

```
C:\Users\lenovo\Desktop\windows>mycp ../zzz copyright
Create new directory copyright
Copy is done !
```

3.1. 验证时间属性

可以看到所有文件夹和单个文件的时间属性都和原文件夹一致，而不是当前时间。

3.2. 验证权限属性

```
C:\Users\lenovo\Desktop\windows>attrib copyright/testA.docx
A    R              C:\Users\lenovo\Desktop\windows\copyright\testA.docx
```

验证方法：查看特殊文件 testA.docx 的权限

可以看到复制后的 testA.docx 和原文件的权限一致，都含有可读权限。

Linux 版本：

（1）数据结构介绍

1. DIR 结构体

是一个内部结构体，用于保存当前被读取的目录相关信息

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;
```

2. stat 结构体

描述一个文件系统中的文件属性的结构

头文件：sys/stat.h

定义如下：

```
struct stat{
    mode_t st_mode; //文件对应的模式，包含权限信息
    ino_t st_ino; //inode 节点号
    dev_t st_dev; //设备号
    dev_t st_rdev; //特殊设备号
    nlink_t st_nlink; //文件连接数
    uid_t st_uid; //文件所有者
    git_t st_gid; //文件所有者对应的组
    off_t st_size; //文件对应的字节数
    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; //文件状态改变的时间
    blksize_t st_blksize; //文件内容对应的块大小
    blkcnt_t st_blocks; //文件对应的块数量
};
```

3. utimbuf 结构体

表示文件属性

头文件：sys.utime.h

定义如下：

```
struct utimbuf{
   time_t actime; //文件的访问时间
   time_t modtime; //文件的修改时间
}
```

4. dirent 结构体

存储读取目录的内容

定义如下：

```
struct dirent{
   long d_ino; //索引节点号
   off_t d_off; //在目录文件中的偏移
   unsigned short d_reclen; //文件名长度
   unsigned char d_type; //文件类型
   char d_name[NAME_MAX+1]; //文件名
};
```

（2）API 介绍

1. opendir(): 打开一个目录

头文件：sys/stat.h, dirent.h

函数声明：DIR* opendir(const char* path);

说明：

path: 目录的路径

返回值：一个 DIR 结构体指针，打开失败时返回空指针

2. mkdir(): 创建一个目录

头文件：sys/stat.h, dirent.h

函数声明：int mkdir(const char *path, mode_t mode);

说明：

path: 要创建的目录的路径

mode：目录权限

返回值：0 表示返回成功，-1 表示错误

3. readdir(): 读取目录内容

头文件：dirent.h

函数声明：struct dirent* readdir(DIR *dir_handle);

说明：

dir_handle: 一个 DIR 结构体指针

返回值：一个 dirent 结构体指针

4. utime(): 修改文件的时间属性

头文件：utime.h

函数声明：int utime(const char *pathname, const struct utimbuf *times);

说明：

pathname: 要修改的文件的路径

times: 一个 utimbuf 结构体，存储文件的时间属性

返回值：成功返回 0，失败返回-1

5. lstat(): 获取文件信息

头文件：sys/stat.h

函数声明：int lstat(const char *path, struct stat *buf);

说明：

path: 文件路径名

buf: 指向 stat 结构体的指针

返回值：执行成功返回 0，执行失败返回-1

6. open(): 打开一个文件

头文件：sys/type.h, sys/stat.h, fcntl.h

函数声明：int fd=open(const char *pathname, int flag);

说明：

pathname: 要打开文件的路径名

flag: 指定文件打开方式

返回值：返回访问句柄，类型为一个整数，若打开失败则返回-1

7. creat(): 创建一个文件

头文件：fcntl.h

函数声明：int creat(const char *pathname,mode_t mode);

说明：

pathname: 要创建的文件的路径

mode: 要创建文件的权限属性

返回值：返回访问句柄，类型为一个整数，若创建失败则返回-1

8. close(): 关闭一个文件

头文件：unistd.h

函数声明：int close(int fd);

说明：

fd: 打开文件时返回的文件访问句柄

返回值：成功关闭返回 0，关闭关闭返回-1

9. write(): 向已经打开的文件中写入数据

头文件：unisd.h

函数声明：ssize_t write(int fd,const void *buf,size_t nbyte);

说明：

fd: 打开文件时返回的文件访问句柄

buf: 缓冲区数据块指针

nbyte: 数据的大小

该函数把 buf 所指的内存写入 nbyte 个字节到 fd 所指的文件中，文件读写位置也会随之移动

返回值：成功写入则返回写入的字节数，即 nbyte，发生错误返回-1

10. read(): 从已经打开的文件读取内容

头文件：unistd.h

函数声明：ssize_t read(int fd,void *buf,size_t nbyte);

说明：

fd: 打开文件时返回的文件访问句柄

buf: 缓冲区数据块指针

nbyte: 文件读取的大小

该函数会把 fd 所指向的文件读出 nbyte 个字节填入 buf 所指的数据块中

返回值：若读取成功，则返回读出的字节数，可以和 nbyte 比较，若少了可能读到了文件末尾，若读取失败则返回-1

（3）代码分析

1. 判断输入参数是否正确

三重检验，输入不合格则给出相应提示。此部分与 Windows 版本的设计完全一样，不多做解释，直接贴出源码:

```
int main(int argc, char *argv[])
{
    struct stat statbuf;
    struct utimbuf timeby;
    DIR * dir;
    if (argc != 3)
    {
        printf("Please enter valid parameters !\n");
        printf("For example: mycp sourceDir destinationDir\n");
    }
    else
    {
        if ((dir = opendir(argv[1])) == NULL)
        {
            printf("Can not find the source directory\n");
        }
        else
        {
            if ((dir = opendir(argv[2])) != NULL)
            {
                printf("The target directory is already exists");
            }
            else if ((dir = opendir(argv[2])) == NULL)
            {
                printf("Create new directory %s\n",argv[2]);
                mkdir(argv[2], statbuf.st_mode);//创建目录
```

2. 判断是否是文件夹

```
while ((entry = readdir(dir)) != NULL)//读目录
{
    if (entry->d_type == 4)//是文件夹,递归复制
    {
        if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0)
        {
```

3. 如果是单个文件

```
else//无目录
{
    strcat(source, "/");
    strcat(source, entry->d_name);
    strcat(target, "/");
    strcat(target, entry->d_name);
    CopyFile(source, target);
    strcpy(source, fsource);
    strcpy(target, ftarget);
}
```

4. 复制单个文件

```
fdr = creat(ftarget, statbuf.st_mode);//创建新文件,返回文件描述符
while ((wordbit = read(fd, BUFFER, buf_size)) > 0)//读取源文件字节数>0
{
    write(fdr, BUFFER, wordbit);//写入目标文件
}
```

5. 如何修改单个文件的权限属性

不同于 Windows，需要用 SetFileAttribute()函数设置，Linux 在创建新文件的时候利用第二个参数 mode 设置文件的权限，即：

```
stat(fsource, &statbuf);
fdr = creat(ftarget, statbuf.st_mode);//创建新文件,返回文件描述符
```

说明：第一行代码使得结构体 stat statbuf 储存了源文件 fsource 的文件属性，而 st_mode 是权限属性。

6. 修改单个文件的时间属性

```
timeby.actime = statbuf.st_atime;//修改时间属性
timeby.modtime = statbuf.st_mtime;
utime(ftarget, &timeby);
```

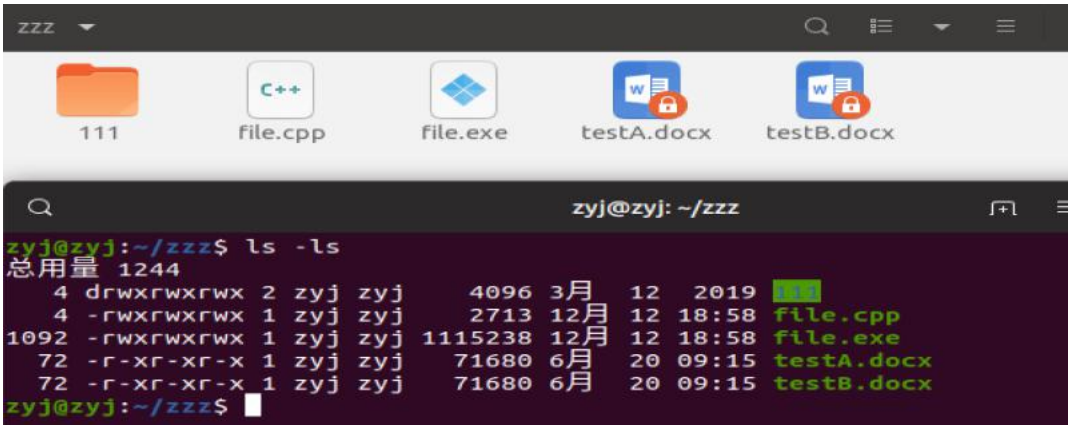
7. 修改新建文件夹的权限属性和时间属性

```
mkdir(target, statbuf.st_mode);//创建目标目录
mycp(source, target);
timeby.actime = statbuf.st_atime;
timeby.modtime = statbuf.st_mtime;//修改文件存取和修改时间
utime(target, &timeby);
```

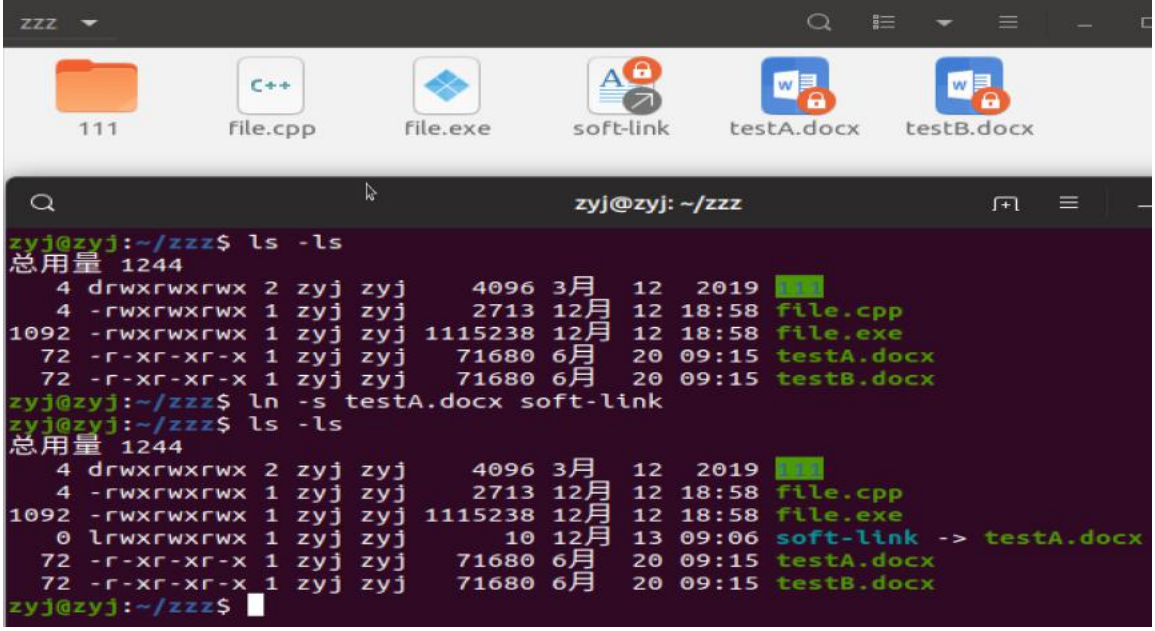
（4）运行效果

1. 准备工作

1.1. 待复制文件夹 zzz

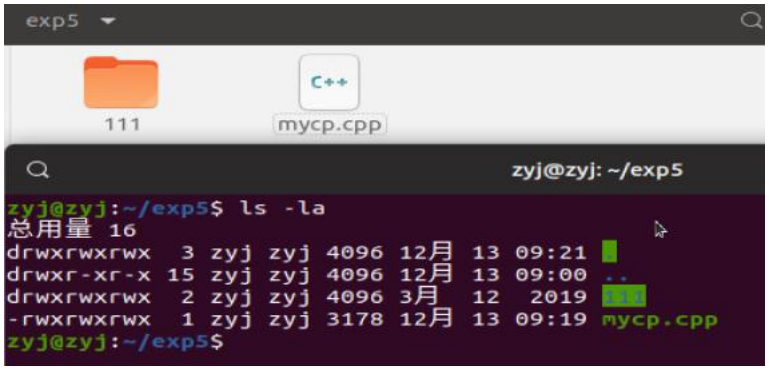


1.2. 使用命令行为文件 testA.docx 增加软链接 soft-link



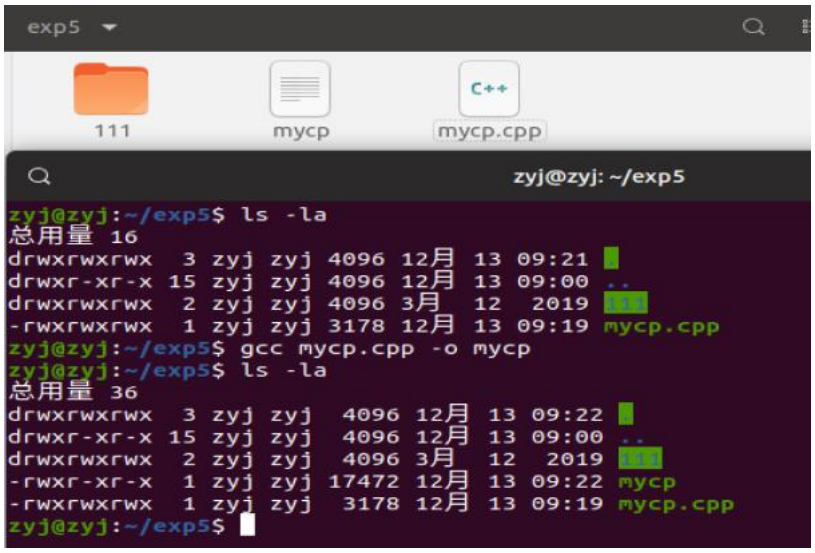
可以看到软链接 soft-link 添加成功。

1.3. 未复制 zzz 前的目标文件夹 exp5



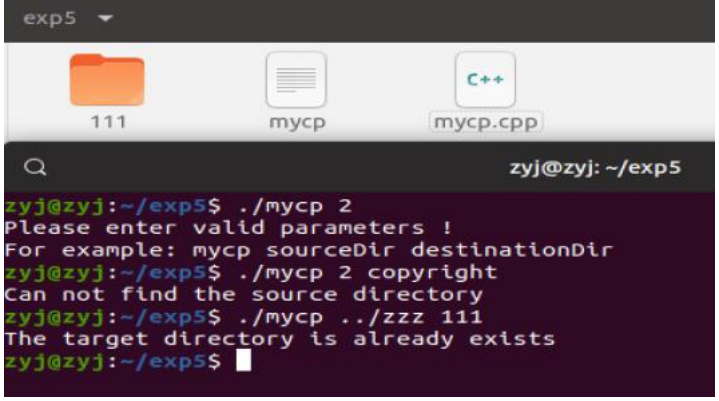
2. 测试输入合理性

2.1 编译源代码

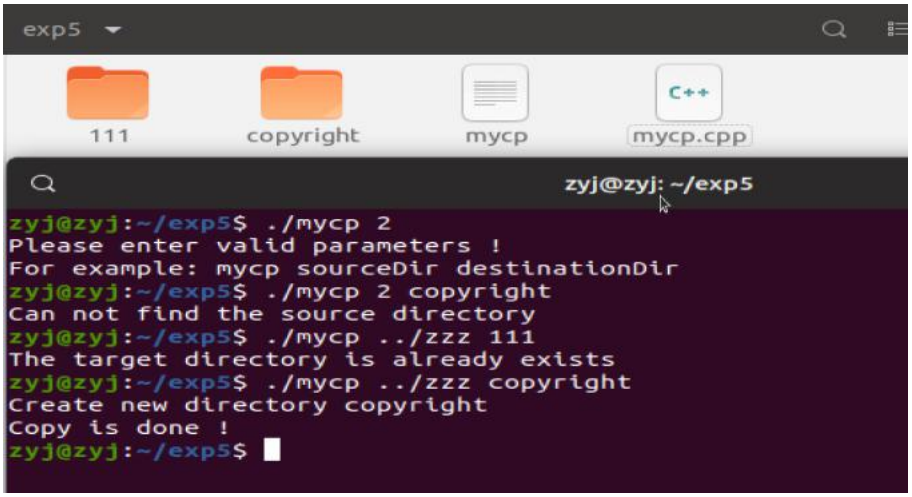


2.2. 测试

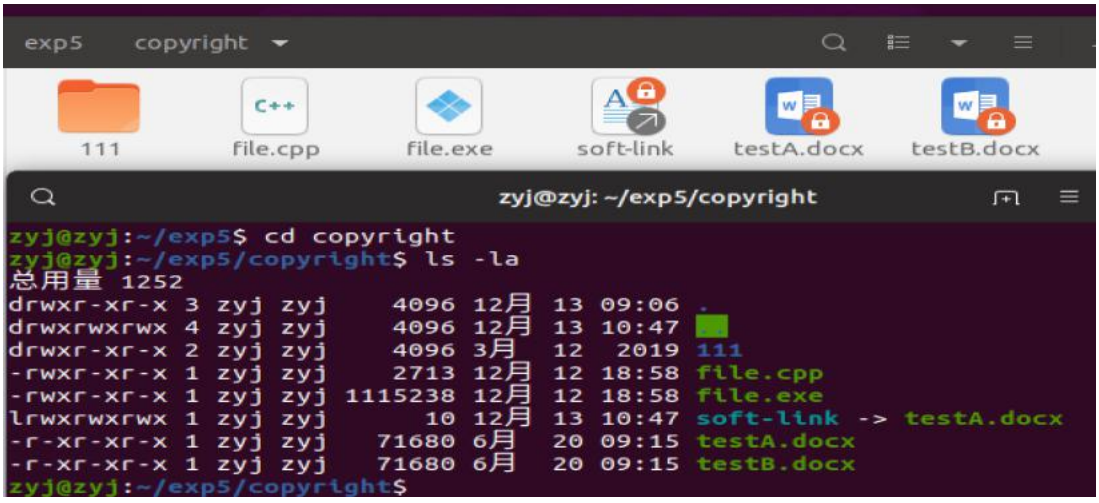
Linux 版本的测试与 Windows 版本完全一致，直接给出测试用例



3. 输入正确格式，完成复制



3.1 验证软链接、权限属性、时间属性



• 可以看到目标文件夹 copyright 中含有软链接，说明 mycp 支持软链接复制

• 可以看到除软链接外的所有文件的权限属性和时间属性均和原文件一致（软链接相当于快捷方式，不同于硬链接，个人感觉时间属性应是当前时间）

五、实验收获与体会

这个实验总算做完了，花了接近两整天时间，和之前几个实验耗时差不多，走了很多坑，不过还好实现了题目要求的所有功能，包括目标文件夹中所有子文

件夹和单个文件的权限属性、时间属性与原文件中的对应文件一致，另外 Linux 还有支持软链接。下面分几点说一下收获与体会。

（1）比较 Windows 版本与 Linux 版本

实验原理完全一样，都是通过递归 mycp()函数到单个文件然后调用 copyFile()函数完成单个文件复制，单个文件复制完成后修改权限和时间，而文件夹的权限和时间是在实现所有单个文件复制的递归最后部分完成修改。

函数接口差别很大，Windows 通过 CreateFile(), ReadFile(), WriteFIle()实现创建文件、读文件、写文件，而 Linux 是通过 creat(), read(), write()实现这三个功能，Windows 这些函数的参数很多，Linux 则简单多了。

再来简单比较权限属性和时间属性如何修改，默认情况下权限是基本权限，时间是当前时间，而这并不符合实验要求，所以需要修改。Windows 是通过调用 SetFileAttributes(), SetFileTime()函数实现修改，并且 SetFileTime()一定要在所有内容复制完成后再被调用以实现修改时间属性。而 Linux 权限属性是创建文件和文件夹的时候通过给定参数实现，不需要单独调用别的函数，时间属性是通过完成所有内容复制后调用 utime()函数实现。

（2）走过的一个大坑

这个坑花了太多太多时间调试——Windows 版本文件夹时间属性的修改

单个文件修改时间属性很简单，但是文件夹的修改却大费周折。首先是一开始不知道调用哪个函数，我以为会有个类似于 SetFileTIme()的 SetDirTime()函数，但是实际上还是通过 SetFileTime()函数实现。其次如何实现呢，SetFileTime()的第一个参数是句柄，而创建单个文件的 CreateFile()返回值也是句柄，所以该返回值可以作为 SetFileTime()的第一个参数指定单个文件，但是创建文件夹的 CreateDirectory()返回值不是句柄，所以我一直在找资料该如何获取句柄。最后我看到了一篇文章了解到竟然是调用 CreateFile()函数获取句柄，因为 CreateFile()函数不单是创建一个文件，还可以打开一个已有文件。就这样，终于可以修改文件夹的时间属性了。

另外，修改代码以后，文件夹的时间属性还是当前时间，这就奇怪了。对于这个问题我是通过打 log 输出修改前和修改后的三种时间（创建时间、最后访问时间、最后写入时间），发现修改后的创建时间和修改前的创建时间是一致的，但是另外两个时间不一样，而 Windows 窗口中显示的是修改时间，即最后写入时间。我明白了，原来是修改代码的位置不对，应该是放在递归结束后。因为如

果在递归之前调用，那么之后创建单个文件的时候会修改最后写入时间，自然 Windows 窗口看到的时间属性是刚才的时间。

然而更换了代码位置之后，发现显示的时间不是当前时间，但是和原文件夹的时间不一致，这也很烦人。对于这个问题我是通过打 log 输出当前文件的路径，发现原来是路径错误，把子目录的路径作为参数，自然父目录的时间是它最后访问的子目录的时间，于是还是要调换代码位置。再一次尝试之后，终于没有问题了，太开心了。

这部分代码如下：

```
if ((strcmp(lpfindfiledata.cFileName, ".") != 0) && (strcmp(lpfindfiledata.cFileName, "..") != 0))
{
    memset(source, '0', sizeof(source));
    lstrcpy(source, fsource);
    lstrcat(source, "\\");
    lstrcat(source, lpfindfiledata.cFileName);//追加文件
    lstrcat(target, lpfindfiledata.cFileName);

    CreateDirectory(target,NULL);//为目标文件创建目录
    mycp(source, target);//进入子目录复制

    HANDLE hDir = CreateFile(target, GENERIC_READ|GENERIC_WRITE,
        FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_FLAG_BACKUP_SEMANTICS, NULL);

    SetFileTime(hDir,
        &lpfindfiledata.ftCreationTime,
        &lpfindfiledata.ftLastAccessTime,
        &lpfindfiledata.ftLastWriteTime);

    CloseHandle(hDir);
    lstrcpy(source, fsource);
    lstrcat(source, "\\");
    lstrcpy(target, ftarget);
    lstrcat(target, "\\");
}
```

（3）**参考资料**

1. 文件复制（野狼）：

   https://blog.csdn.net/yugemengjing/article/details/72792626

2. 文件复制（PatrickYangyz）：

   https://blog.csdn.net/PatrickYangyz/article/details/89074038

3. linux ln 用法（软链接和硬链接）：

   https://blog.csdn.net/mengzuchao/article/details/80426316

4. SetFileAttributes 和 GetFileAttributes：

   https://www.cnblogs.com/kex1n/archive/2011/08/25/2153542.html

5. 获取并修改文件夹时间（创建时间、修改时间、访问时间）：

   https://www.jianshu.com/p/9416d6d40fdc

6. CMD 中使用 attrib 命令设置文件只读、隐藏属性详解：

   https://www.jb51.net/article/53104.htm

（4）路还很长，脚踏实地，加油！

——2019 年 12 月 13 日

（5）后续完善

上个星期去验收，发现一定要改软链接的时间，于是当天加了修改软链接时间的代码，因为这个星期以来比较忙，于是实验报告拖到今天平安夜来改了。

代码只需多加几行即可，增添代码为：

```
if (S_ISLNK(statebuf1.st_mode))//是软链接
{
    int rslt = readlink(fsource, buf, 511);
    buf[rslt] = '\0';
    symlink(buf, ftarget);
    struct timeval times[2];
    times[0].tv_sec = statebuf1.st_atime;
    times[0].tv_usec = 0;
    times[1].tv_sec = statebuf1.st_mtime;
    times[1].tv_usec = 0;
    lutimes(ftarget, times);
    return;
}
```

另外还需要增加一个头文件#include <sys/time.h>。

接下来进行测试，省略操作步骤，直接给出复制结果，如下所示：

```
zyj@zyj:~/exp5$ ./mycp zzz 123
Create new directory 123
Copy is done !
zyj@zyj:~/exp5$ ls -la zzz
总用量 1252
drwxrwxrwx 3 zyj zyj    4096 12月 18 23:51 .
drwxrwxrwx 4 zyj zyj    4096 12月 24 23:03 ..
drwxrwxrwx 2 zyj zyj    4096 12月 18 23:01 111
-rwxrwxrwx 1 zyj zyj    2713 12月 12 18:58 file.cpp
-rwxrwxrwx 1 zyj zyj 1115238 12月 12 18:58 file.exe
lrwxrwxrwx 1 zyj zyj       8 12月 18 23:01 soft-link -> file.cpp
lrwxrwxrwx 1 zyj zyj      27 12月 18 23:51 soft-link-2 -> /home/zyj/exp5/zzz/file.exe
-r-xr-xr-x 1 zyj zyj   71680 6月   20  2019 testA.docx
-r-xr-xr-x 1 zyj zyj   71680 6月   20  2019 testB.docx
zyj@zyj:~/exp5$ ls -la 123
总用量 1252
drwxr-xr-x 3 zyj zyj    4096 12月 18 23:51 .
drwxrwxrwx 4 zyj zyj    4096 12月 24 23:03 ..
drwxr-xr-x 2 zyj zyj    4096 12月 18 23:01 111
-rwxr-xr-x 1 zyj zyj    2713 12月 12 18:58 file.cpp
-rwxr-xr-x 1 zyj zyj 1115238 12月 12 18:58 file.exe
lrwxrwxrwx 1 zyj zyj       8 12月 18 23:01 soft-link -> file.cpp
lrwxrwxrwx 1 zyj zyj      27 12月 18 23:51 soft-link-2 -> /home/zyj/exp5/zzz/file.exe
-r-xr-xr-x 1 zyj zyj   71680 6月   20  2019 testA.docx
-r-xr-xr-x 1 zyj zyj   71680 6月   20  2019 testB.docx
zyj@zyj:~/exp5$
```

可以看出两个文件夹的软连接时间一致，修改成功。

平安夜快乐！

——2019 年 12 月 24 日