北京理工大学

操作系统课程设计实验报告

实验五 复制文件

指导教师：陆慧梅老师

班　　级：07111507

学　　号：1120151880

姓　　名：廖汉龙

邮　　箱：liamliaohl@gmail.com

2018 年 4 月 23 日

实验报告链接：https://github.com/HanlongLiao/Course/tree/master/OS

目录

一、实验目的

1. 熟悉文件操作的相关系统函数。熟悉 Linux 和 Windows 两种环境中文件操作的相关函数，掌握创建文件、复制文件、读写文件的方法。理解各个函数的作用，理解函数参数。

2. 理解文件系统的概念。通过对文件的复制操作，理解 Linux 和 Windows 两种环境下文件系统的作用。

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，运行结果如代码 2-1：

```
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/
```

代码 2-1

并且要求：
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、实验环境及配置方法

1.Windows 环境：

操作系统	Windows 家庭中文版
运行内存	8G
GPU	Core i5 4200

编辑器	Visual studio Code
编译环境	GCC 4.9.2

表 3-1

2. Linux 环境：

操作系统	**Ubuntu 16.04 LTS**
虚拟机	Vmware WorkStation
编辑器	Visual studio Code
编译环境	GGC 5.2.0

表 3-2

四、调用 API 说明

4.1 windows 系统 API 介绍

- CreateDirectory() 创建目录

```
BOOL WINAPI CreateDirectory(
    LPCTSTR                lpPathName,              //是新创建目录的路径名
    LPSECURITY_ATTRIBUTES lpSecurityAttributes    //是为目录设置的安全属性结构，若该参数为 NULL，
                                                  //则设置默认为默认的安全属性
);
```

代码 4-1-1

成功调用时返回 True，否则返回 false，且通过函数 GetLastError()可以获得错误码。

- RemoveDirecotory() 删除目录

```
BOOL WINAPI RemoveDirectory(
    LPCTSTR lpPathName        //为要删除的目录名
);
```

代码 4-1-2

成功时返回 **true**，否则返回 **false**。当进程没有删除权限或者目录中没有文件或其他子目录时，删除失败。

- GetCurrentDirectory() 得到当前目录

```
DWORD WINAPI GetCurrentDirectory(
DWORD  nBufferLength,            //为缓冲区的最大字符数
LPTSTR lpBuffer                  //是指向存放当前目录的缓冲区的指针
);
```

代码 4-1-3

函数调用成功时，用进程的当前目录填充有参数 lpBuffer 指向的缓冲区，并返回复制到缓冲区中的字符数（不包括结尾的 0）。使用函数 GetLastError() 函数来获得错误信息。为了确保函数调用成功，通常使用如下的形式：

```
TCHAR szCurDir[MAX_PATH];
DWORD dwCurDir = GetCurrentDirectory(sizeof(dzCurDir)/sizeof(TCHAR), szCurDir);
```

代码 4-1-4

其中，MAX_PATH 定义在文件 WINDEF 中，大小为 260。如果没有定义，可以自行第一该常量。

- SetCurrentDiretory() 改变当前目录

```
BOOL WINAPI SetCurrentDirectory(
    LPCTSTR lpPathName          //是指存放当前目录字符串的缓冲区的指针
);
```

代码 4-1-5

返回非 0 表示成功，0 表示失败，且通过函数 GetLastError()可以获得错误码

- FindFirstFile() 查找指定文件路径的文件

```
HANDLE WINAPI FindFirstFile(
    LPCTSTR          lpFileName,        //指向一个以 0 结尾的字符串的文件名，文件名可以包含通配符(* 和 ?)
    LPWIN32_FIND_DATA lpFindFileData     //是一个 Win32_FIND_DATA 结构的地址
);
```

代码 4-1-6

如果调用成功，返回值为非 0，否则返回值为 0，且可通过 GetLastError()函数来获得错误信息。

其中 Win32_FIND_DATA 结构的定义如下：

```
typedef struct _WIN32_FIND_DATA {
  DWORD    dwFileAttributes;      //文件属性
  FILETIME ftCreationTime;        //文件创建时间
  FILETIME ftLastAccessTime;      //文件最后一次访问时间
  FILETIME ftLastWriteTime;       //文件最后一次修改时间
  DWORD    nFileSizeHigh;         //文件长度高 32 位
  DWORD    nFileSizeLow;          //文件长度低 32 位
  DWORD    dwReserved0;           //系统保留
  DWORD    dwReserved1;           //系统保留
  TCHAR    cFileName[MAX_PATH];   //长文件名
  TCHAR    cAlternateFileName[14];//8.3 格式文件名
} WIN32_FIND_DATA, *PWIN32_FIND_DATA, *LPWIN32_FIND_DATA;
```

代码 4-1-7

- FindNextFile() 查找 FindFirstFile()函数搜索之后的下一个文件

```
BOOL WINAPI FindNextFile(
    HANDLE            hFindFile,       //是调用 FindFirstFile 时，返回的文件句柄
    LPWIN32_FIND_DATA lpFindFileData   //是一个 Win32_Find_Data 结构的地址
);
```

代码 4-1-8

如果调用函数成功，返回值为非 0，否则返回值为 0，且可以通过 GetLastError()函数来获取错误信息。

- FindClose() 关闭由 FindFirstFile()函数搜索到的文件句柄

```
BOOL WINAPI FindClose(
    HANDLE hFindFile           //为要关闭的由 FindFirstFile()返回的文件句柄
);
```

代码 4-1-9

如果调用成功，返回值为非 0，否则返回值为 0，且可通过 GetLastError()来获得错误信息

- GetFileTime 检索文件或目录，上次访问和上次修改的日期和时间

```
BOOL WINAPI GetFileTime(
    HANDLE     hFile,            //要检索日期和时间的文件或目录的句柄
    LPFILETIME lpCreationTime,   //指向 FILETIME 结构的指针，用于接收创建文件或目录的日期和时间。
                                 //如果应用程序不需要此信息，则此参数可以为 NULL
    LPFILETIME lpLastAccessTime, //指向 FILETIME 结构的指针，用于接收上次访问文件或目录的日期和时间
    LPFILETIME lpLastWriteTime   //指向 FILETIME 结构的指针，用于接收上次写入文件或目录的日期和时间，
                                 //截断或覆盖（例如，使用 WriteFile 或 SetEndOfFile）
);
```

代码 4-1-10

如果函数成功，返回值为非零。

如果函数失败，返回值为零。要获得扩展的错误信息，可调用 GetLastError。

- SetFileTime() 设置创建，上次访问或上次修改指定文件或目录的日期和时间

```
BOOL WINAPI SetFileTime(
    HANDLE   hFile,
    const FILETIME *lpCreationTime,      //文件或目录的句柄。该句柄必须使用具有 FILE_WRITE_ATTRIBUTES
                                         //访问权限的 CreateFile 函数 创建
    const FILETIME *lpLastAccessTime,    //指向包含文件或目录的新创建日期和时间的 FILETIME 结构的指针
```

```
    const FILETIME *lpLastWriteTime      //指向 FILETIME 结构的指针，该结构包含文件或目录的新的上次修改日期和时间
);
```

代码 4-1-11

如果函数成功，返回值为非零。

如果函数失败，返回值为零。要获得扩展的错误信息，可调用 GetLastError。

- CreateFile() 创建获得打开文件

```
HANDLE WINAPI CreateFile(
    LPCTSTR               lpFileName,            //为指向一个以 NULL 结束的字符串的指针，该字符串是打开或者创建
                                                   //文件等对象的名字
    DWORD                 dwDesiredAccess,       //为指定对象的访问类型，可能的访问类型为读，写，读写访问或者
                                                   //查询设备等类型
    DWORD                 dwShareMode,           //为规定与其他进程共享文件的方式，0 为不允许共享。FILE_SHARE_READ
                                                   //或 FILE_SHARE_WRITE 表示运行对文件进程共享访问
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,   // 为指向一个 SECURITY_ARRTIBUTES 结构的指针，该结构指定一个安全对象，
                                                   //在创建文件时使用。如果为 NULL,表示使用默认安全对象
    DWORD                 dwCreationDisposition,  //为指定如何创建文件
    DWORD                 dwFlagsAndAttributes,   //为指定文件的属性和标志
    HANDLE                hTemplateFile          //如果不为 0，则表示一个文件句柄。新文件将从这个文件中赋值扩展属性
);
```

代码 4-1-12

如果函数调用成功，返回值为指向指定文件的打开句柄；如果函数调用失败，返回值为 INVALD_HENDLE_VALUE，且会设置 GetLastError.

- ReadFile() 从文件中读取数据

```
BOOL WINAPI ReadFile(
    HANDLE       hFile,                    //设备的句柄
    LPVOID       lpBuffer,                 //指向接收从文件或设备读取的数据的缓冲区的指针。
                                           //这个缓冲区在读操作期间必须保持有效。在读取操作完成之前，调用方
                                           //不得使用此缓冲区。
    DWORD        nNumberOfBytesToRead,     //指向接收使用同步 hFile 参数时读取的字节数的变量的指针
    LPDWORD      lpNumberOfBytesRead,      //如果使用 FILE_FLAG_OVERLAPPED 打开 hFile 参数， 则需要指向 OVERLAPPED
                                           //结构的指针，否则它可以为 NULL
```

```
    LPOVERLAPPED lpOverlapped                    //如果使用 FILE_FLAG_OVERLAPPED 打开
hFile 参数， 则需要指向 OVERLAPPED 结构的指针，
                                                 //否则它可以为 NULL。
                                                 //如果使用 FILE_FLAG_OVERLAPPED 打
开 hFile，则 lpOverlapped 参数必须指向有效且唯一
                                                 //的 OVERLAPPED 结构，
                                                 //否则该函数可能会错误地报告读取操作
已完成。
);
```

代码 4-1-13

如果函数成功，则返回值为非零（TRUE），如果函数失败或异步完成，则返回值为零（FALSE）。要获得扩展的错误信息，可调用 GetLastError 函数。

- WriteFile() 向文件中写数据

```
BOOL WINAPI WriteFile(
    HANDLE       hFile,                  //文件或 I / O 设备的句柄
    LPCVOID      lpBuffer,               //指向包含要写入文件或设备的数
据的缓冲区的指针。
                                         //这个缓冲区在写操作期间必须
保持有效，写操作完成之前，调用方不得使用此缓冲区
    DWORD        nNumberOfBytesToWrite,  //要写入文件或设备的字节数
    LPDWORD      lpNumberOfBytesWritten, //指向接收使用同步 hFile 参数
时写入的字节数的变量的指针
    LPOVERLAPPED lpOverlapped            //如果使用
FILE_FLAG_OVERLAPPED 打开 hFile 参数， 则需要指向 OVERLAPPED 结构的指针，
                                         //否则此参数可以为 NULL
);
```

代码 4-1-14

如果函数成功，则返回值为非零（TRUE），如果函数失败或异步完成，则返回值为零（FALSE）。要获得扩展的错误信息，可调用 GetLastError 函数。

4.2 Linux 系统的 API 简单介绍

- stat()

```
int stat(const char *file_name, //所要打开的路径名
struct stat *buf)//要存入文件信息的结构体
```

代码 4-2-1

通过文件名 filename 获取文件信息，并保存在 buf 所指的结构体 stat 中。 函数原型如下：
其中 stat 的内容如下：

```
struct stat {
   dev_t st_dev; //文件的设备编号
   ino_t st_ino; //节点
   mode_t st_mode; //文件的类型和存取的权限
```

```
    nlink_t st_nlink; //连到该文件的硬连接数目，刚建立的文件值为 1
    uid_t st_uid; //用户 ID
    gid_t st_gid; //组 ID
    dev_t st_rdev; ///(设备类型)若此文件为设备文件，则为其设备编号
    off_t st_size; //文件字节数(文件大小)
    unsigned long st_blksize; //块大小(文件系统的 I/O 缓冲区大小)
    unsigned long st_blocks; //块数
    time_t st_atime; //最后一次访问时间
    time_t st_mtime; //最后一次修改时间
    time_t st_ctime; //最后一次改变时间(指属性)
};
```

代码 4-2-2

- lstat()

```
int lstat(const char * file_name, struct stat * buf);
```

代码 4-2-3

lstat()与 stat()作用完全相同，都是取得参数 file_name 所指的文件状态，其差别在于，当文件为符号连接时，lstat()会返回该 link 本身的状态。 执行成功则返回 0，失败返回-1，错误代码存于 error。

- creat()

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

代码 4-2-4

使用 creat 函数创建一个新文件，如果原来该文件存在，会将这个文件的长度截短为 0。若成功， 则会返回为只写打开的文件描述符， 出错则返回-1

- write()

```
ssize_t write(int filedes, void* buff, size_t nbytes);
```

代码 4-2-5

调用 write 函数向打开的文件写入数据，成功则返回实际写入的 byte 数，出错则返回-1

- open()

```
int open(const char *pathname, int flag);
```

代码 4-2-6

调用 open 函数打开或者创建一个文件。使用 open 返回的文件描述符作为参数传递给 write 或 read。

- opendir() 打开指定目录

```
extern DIR* opendir(const char * name);
```

代码 4-2-7

接口用来打开指定的的目录，并返回 DIR*形态的目录流。当 opendir()打开成功时，会返回一个 DIR*型的目录流，若打开失败，则返回 NULL,可用于检测输入路径是否存在。

- readdir() 读取目录

```
struct dirent* readdir(DIR* dir);
```

代码 4-2-8

readdir 函数返回一个指向 dirent 结构体的指针，该结构体代表了由 dir 指向的目录流中的下一个目录项；如果读到 end-of-file 或者出现错误，则返回 NULL。
在 Linux 中 dirent 结构的定义如下：

```
struct dirent {
    ino_t d_ino;        /* inode number */
    off_t d_off;        /* offset to the next dirent */
    unsigned short d_reclen; /* length of this record */
    unsigned char d_type;                //文件的类型
    char d_name[256];                    //文件名称
};
```

代码 4-2-9

- mkdir()

```
int mkdir( const char *path, //要创建的目录名
mode_t mode); //目录权限
```

代码 4-2-10

创建所输入的新的目录， 返回 0 表示成功， 返回 -1 表示错误，并且会设置 error 值。 函数原型如下:

- readlink()

```
int readlink(const char * path, char * buf, size_t bufsiz);
```

代码 4-2-11

readlink()会将参数 path 的符号连接内容存到参数 buf 所指的内存空间，返回的内容不是以 NULL 作字符串结尾，但会将字符串的字符数返回。 若参数 bufsiz 小于符号连接的内容长度，过长的内容会被截断。执行成功则传符号连接所指的文件路径字符串，失败则返回-1,错误代码存于 error。

五、实验原理与步骤

5.1 实验原理

现代操作系统向用户提供了强大的文件管理系统，灵活的目录结构——层次式的树形目录结构。它有一个主目录（又称为根目录），在根目录下有许多用户目录和普通文件目录项，每个用户目录下依次有许多目录或普通文件作为其目录项。文件夹是树的非叶节点，文件则是叶子节点。同时，文件夹或者目录本身也是一个文件，他的主要信息是包含的子文件索引，这样一来，所有对文件的操作实质本质上都变成了对“目录树”的操作。我们要访问某一个文件就是访问这棵树的叶子节点，“目录树”这种数据结构拥有很多成型的算法，对数的遍历也拥有也是非常常见的方法，文件树示意图如图 5-1-1：

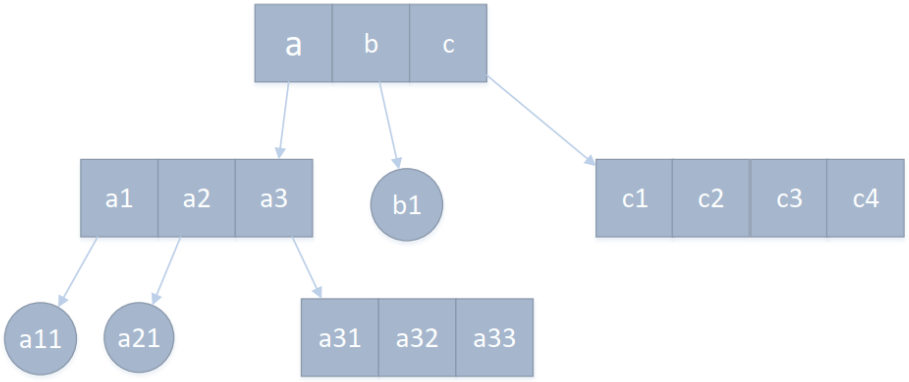

图 5-1-1

实现文件夹拷贝实际上是遍历访问整颗“目录树”的过程，在遍历访问的同时完成复制。对数的遍历一般采用 DFS 搜索较为方便，因为 DFS 可以使用递归算法实现。所以本实验的核心函数就是设计一个递归函数——DirWalk()，其流程如图 5-1-2 所示：

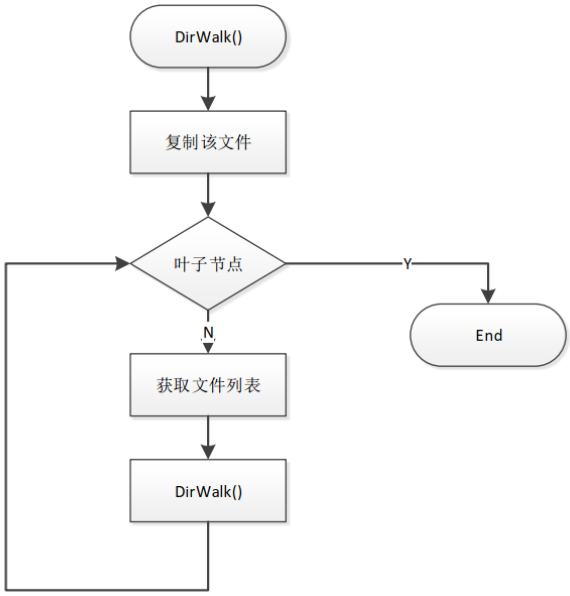

图 5-1-2

DirWalk( )函数接受 3 个参数，源文件路径，目的文件路径和当前搜索的深度。在函数开始执行后， DirWalk()首先根据源路径获取当前获取子文件的列表，对每一子文件判断其类型，如果还是一个目录文件，则继续调用 DirWalk( )对子文件进行处理，如果是一个普通文件，则调用其他函数对文件进行直接复制。每次调用 DirWalk( )函数，深度都是上一层的深度 +1 ，深度参数用于打印目录树。

在 Linux 中和 Windows 中存在差别的是对于链接文件的拷贝，下面是关于链接文件的理论内容。

Linux 链接分两种，一种被称为硬链接（Hard Link），另一种被称为符号链接（Symbolic Link）。默认情况下，ln 命令产生硬链接。

在 Liunx 中，文件都有文件名与数据，这被分成两个部分：用户数据 (user data) 与元数据 (metadata)。用户数据，即文件数据块 (data block)，数据块是记录文件真实内容的地方；而元数据则是文件的附加属性，如文件大小、创建时间、所有者等信息。在 Linux 中，元数据中的 inode 号（inode 是文件元数据的一部分但其并不包含文件名，

inode 号即索引节点号）才是文件的唯一标识而非文件名。文件名仅是为了方便人们的记忆和使用，系统或程序通过 inode 号寻找正确的文件数据块。图 5-1-3 展示了程序通过文件名获取文件内容的过程。

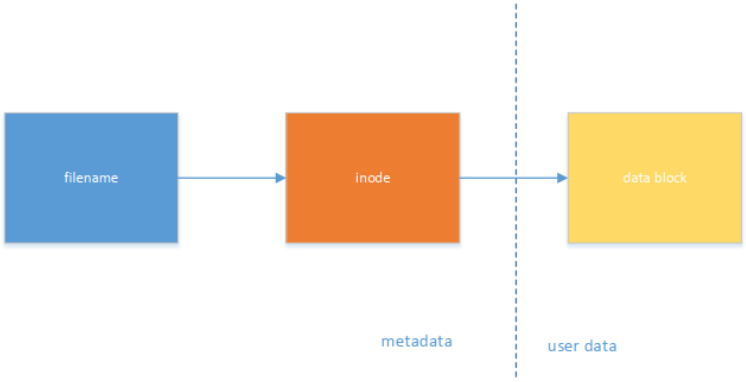

图 5-1-3

在 Linux 系统中查看 inode 号可使用命令 stat 或 ls -i。

为解决文件的共享使用，Linux 系统引入了两种链接：硬链接（hard link）与软链接（又称符号链接，即 soft link 或 symbolic link）。链接为 Linux 系统解决了文件的共享使用，还带来了隐藏文件路径、增加权限安全及节省存储等好处。若一个 inode 号对应多个文件名，则称这些文件为硬链接。换言之，硬链接就是同一个文件使用了多个别名

由于硬链接是有着相同 inode 号仅文件名不同的文件，因此硬链接存在以下几点特性：

- 文件有相同的 inode 及 data block；
- 只能对已存在的文件进行创建；
- 不能交叉文件系统进行硬链接的创建；
- 不能对目录进行创建，只可对文件创建；
- 删除一个硬链接文件并不影响其他有相同 inode 号的文件。

inode 号仅在各文件系统下是唯一的，当 Linux 挂载多个文件系统后将出现 inode 号重复的现象，因此硬链接创建时不可跨文件系统。

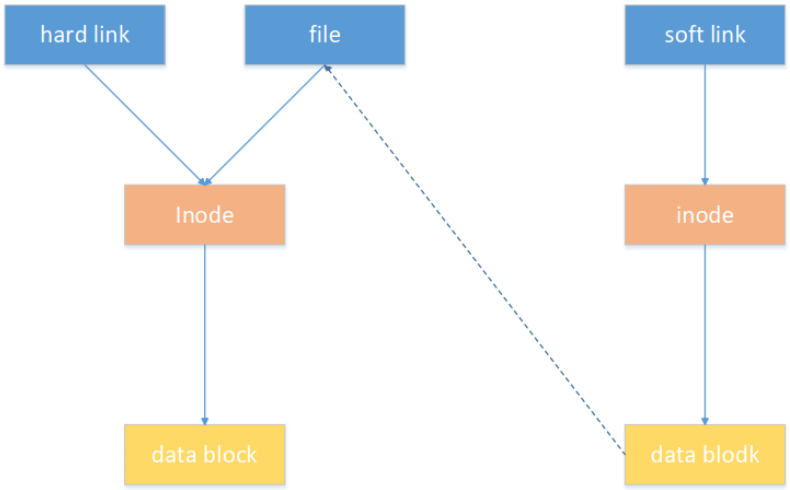

图 5-1-4

软链接与硬链接不同，若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接。软链接就是一个普通文件，只是数据块内容有点特殊。软链接有着自己的 inode 号以及用户数据块（见图 5-1-4）。因此软链接的创建与使用没有类似硬链接的诸多限制：

- 软链接有自己的文件属性及权限等；
- 可对不存在的文件或目录创建软链接；
- 软链接可交叉文件系统；
- 软链接可对文件或目录创建；
- 创建软链接时，链接计数 i_nlink 不会增加；
- 删除软链接并不影响被指向的文件，但若被指向的原文件被删除，则相关软连接被称为死链接（即 dangling link，若被指向路径文件被重新创建，死链接可恢复为正常的软链接）。

当然软链接的用户数据也可以是另一个软链接的路径，其解析过程是递归的。但需注意：软链接创建时原文件的路径指向使用绝对路径较好。使用相对路径创建的软链接被移动后该软链接文件将成为一个死链接

5.2 Windows 系统下的实现步骤

实现文件夹复制的基本思想是文件树的遍历，如果把目录文件也看成文件的话，那么这棵树的叶子节点为普通文件，叶子节点为目录文件。根据文件树这样的特点，那么递归函数就可以这样设计，对于当前打开的目录，首先用 FindFirstFile() API 打开一个目录内容的句柄，再用 FindNextFile() API 依次访问下一个文件，其中，访问的位置不需要程序员自己维护。用一个循环访问目录中的所有文件，如果是目录文件，就先完成当前子文件的复制， 再调用 DirWalk( )函数对子文件进行处理， 如果是一个普通文件， 那么直接调用 CopyFile()API 进行复制即可。

为了在遍历过程中，方便存放目标路径以及当前拷贝的深度，方便打印，参考了教材上相应的事先方法，在 Windows 系统下，我的方法是通过维护一个结构体实现的。结构体的定义如下：

```
typedef struct _DIRWALK
{
    int depth;      //深度
    BOOL FindNext;  //查找成功
    TCHAR DesPath[MAX_PATH];        //目标路径名
    TCHAR DestFullName[MAX_PATH];   //目标文件名
    WIN32_FIND_DATA FindData;       //存放查找到的文件信息
}DIRWALK, *pDIRWALK;
```

代码 5-2-1

这个结构体可以记录程序运行当前时刻的所有信息：复制的深度，是否查找成功，复制到的目标路径名称，以及当前查找到的文件信息，文件信息使用 WIN32_FIND_DATA 结构体存储，在前面 API 介绍中以及有说明。

整个查找文件的过程中，最核心的部分是 DirWalk()程序的实现：

```
static void DirWalk(pDIRWALK DW)
```

```
{
    DW->depth++;
    //par1: 表示查找的文件的类型
    //par2: 查找到的文件的属性保存位置，包括文件名，文件的创建时间等等
    HANDLE hFind = FindFirstFile(_TEXT("*.*"), &DW->FindData);
    while (DW->FindNext)
    {
        //是父路径或当前路径  取下一个
        if ((lstrcmp(DW->FindData.cFileName, _TEXT(".")) == 0) ||
            (lstrcmp(DW->FindData.cFileName, _TEXT("..")) == 0));
        //判断当前是否是文件夹
        else if (DW->FindData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        {
            _stprintf_s(DW->DesPath,
_TEXT("%s\\%s"),DW->DesPath,  DW->FindData.cFileName);//目标文件名
            printf("%*sCreateDirectory %s\n", 4 * DW->depth, "", DW->DesPath);
            CreateDirectory(DW->DesPath, NULL);          //创建文件夹
            //切换当前的进程的当前工作目录
            SetCurrentDirectory(DW->FindData.cFileName);
            DirWalk(DW);                                 //递归遍历复制
            SetCurrentDirectory(_TEXT(".."));            //返回上一层文件夹
            _tcscpy(DW->FindData.cFileName, _tcsrchr(DW->DesPath, _TEXT('\\')) +
1);
            CopyTime(DW->FindData.cFileName, DW->DesPath);//复制文件夹时间
            _tcsrchr(DW->DesPath, _TEXT('\\'))[0] = 0;
        }
        else
        {
            _stprintf(DW->DestFullName, _TEXT("%s\\%s"), DW->DesPath,
DW->FindData.cFileName);
            printf("%*sCopyFile %s\n", 4 * DW->depth, "",DW->FindData.cFileName);
            myCopyFile(DW->FindData.cFileName, DW->DestFullName);       //复制文件
函数
        }
        DW->FindNext = FindNextFile(hFind, &DW->FindData);//查找下一个文件
    }
    if (hFind != INVALID_HANDLE_VALUE)
        FindClose(hFind);
    DW->depth--;
}
```

代码 5-2-2

这是整个代码最为核心的代码，我下面对其进行说明：

首先是进入到这个函数，说明当前已经递归到目录的下一层，所以，结构体的递归层数需要加 1：DW->depth++;

然后查找到的当前第一个文件，并且将文件信息保存在 DW->FindData 中。

如果当前这个文件查找是成功的，进入循环，首先判断这个文件是否是当前路径或者是父路径，如果是，这说明这次查找已经完成，应该遍历上一层的下一个文件，判断的语句如下：

```
if ((lstrcmp(DW->FindData.cFileName, _TEXT(".")) == 0) ||
            (lstrcmp(DW->FindData.cFileName, _TEXT("..")) == 0));
```

代码 5-2-3

如果不是的话，说明可以进行当前文件的复制，但存在两种情况，当前的文件有可能是文件夹或者文件，如果是文件夹，那么需要递归，进入到这个文件夹的下一层，执行相同的操作；如果是文件，那么调用 mycopyfile()函数，直接复制文件。

如果判断当前是文件夹，首先改变目标地址的地址，增加一层，还需要在当前路径下创建一个文件夹，切换当前的进程到源地址的下一层，再进行递归，直到以当前文件夹为根节点的所有文件拷贝结束，再返回到上一层，并且设置文件的时间。

对于是文件夹的处理语句如下：

```
_stprintf_s(DW->DesPath, _TEXT("%s\\%s"),
DW->DesPath,   DW->FindData.cFileName);//目标文件名
            printf("%*sCreateDirectory %s\n", 4 * DW->depth, "", DW->DesPath);
            CreateDirectory(DW->DesPath, NULL);          //创建文件夹
            //切换当前的进程的当前工作目录
            SetCurrentDirectory(DW->FindData.cFileName);
            DirWalk(DW);                                 //递归遍历复制
            SetCurrentDirectory(_TEXT(".."));            //返回上一层文件夹
            _tcscpy(DW->FindData.cFileName, _tcsrchr(DW->DesPath, _TEXT('\\')) +
1);
            CopyTime(DW->FindData.cFileName, DW->DesPath);//复制文件夹时间
            _tcsrchr(DW->DesPath, _TEXT('\\'))[0] = 0;
```

代码 5-2-4

如果当前判断是文件，直接在当前路径下拷贝文件，调用 mycopyfile()函数。

最后是深度的自减，当前层的所有的文件已经拷贝完毕，则返回上一层，深度自减。

下面是拷贝文件的函数的简单介绍：

这部分实现通过

下面简单说明拷贝时间的函数 CopyTime()。在 windows 系统下较为容易实现，但是在 linux 下似乎有很多人出现了问题，在后面的 linux 的部分将做一些说明。在 windows 系统下，只需要创建指向文件的句柄，然后调用系统 API，GetFileTime()和 SetFileTime()可以实现文件时间的拷贝，使用方法如下：

```
GetFileTime(hSrc, &CreationTime, &LastAccessTime, &LastWriteTime);//获取文件时间
    SetFileTime(hDes, &CreationTime, &LastAccessTime, &LastWriteTime);//设置文件时
间
```

5.3 Linux 系统下的实现

Linux 下的实现方法与 Windows 下没有差别，但是由于在 Linux 下有链接文件，并且对于文件的判断的方式等都有更好的方式，所以，在 Linux 下并没有像 Windows 下定义了一个结构体。最核心的部分是 CopyAllFile()函数

```
void CopyAllFile(char* source, char* target, int deepth)
{
    char cSource_temp[MAX_PATH];
    char cTarget_temp[MAX_PATH];
    char cSource_temp_all[MAX_PATH];

    DIR *dp;

    struct dirent *entry;
    struct stat statbuf;
    if ((dp = opendir(source)) == NULL)
    {
        printf("opendir Error.\n");
    }
    while ((entry = readdir(dp)) != NULL)
    {
        lstat(entry->d_name, &statbuf); //文件名
        if ((strcmp(".", entry->d_name) != 0) && (strcmp("..", entry->d_name) !=
0))  //文件夹
        {
            if (entry->d_type == 4) //目录类型
            {
                strcpy(cSource_temp, source);
                strcat(cSource_temp, "/");
                strcat(cSource_temp, entry->d_name);
                strcpy(cTarget_temp, target);
                strcat(cTarget_temp, "/");
                strcat(cTarget_temp, entry->d_name);
                lstat(cSource_temp, &statbuf);
                mkdir(cTarget_temp, statbuf.st_mode);
                printf("%*s Copyfolder: %s\n", 4 * deepth, "", cSource_temp);
                int a = deepth;
                a ++;
                CopyAllFile(cSource_temp, cTarget_temp, a);
                SetTime(cSource_temp, cTarget_temp);
            }
            if (entry->d_type == 10)//链接文件
            {
                char buf[1024];
                memset(buf,0,1024);
                ssize_t len = 0;

                strcpy(cSource_temp, source);
                strcat(cSource_temp, "/");
```

```
                strcat(cSource_temp, entry->d_name);
                strcpy(cTarget_temp, target);
                strcat(cTarget_temp, "/");
                strcat(cTarget_temp, entry->d_name);

                len = readlink(cSource_temp, buf, 1024 - 1);
                buf[len] = '\0';
                symlink(buf, cTarget_temp);
                SetTime(cSource_temp, cTarget_temp);
            }
            else   //当前为文件
            {
                strcpy(cSource_temp, source);
                strcat(cSource_temp, "/");
                strcat(cSource_temp, entry->d_name);
                strcpy(cTarget_temp, target);
                strcat(cTarget_temp, "/");
                strcat(cTarget_temp, entry->d_name);
                int a = deepth;
                copyFile(cSource_temp, cTarget_temp, a);
                SetTime(cSource_temp, cTarget_temp);
            }
        }
    }
}
```

代码 5-3-1

下列是以上程序的结构图：

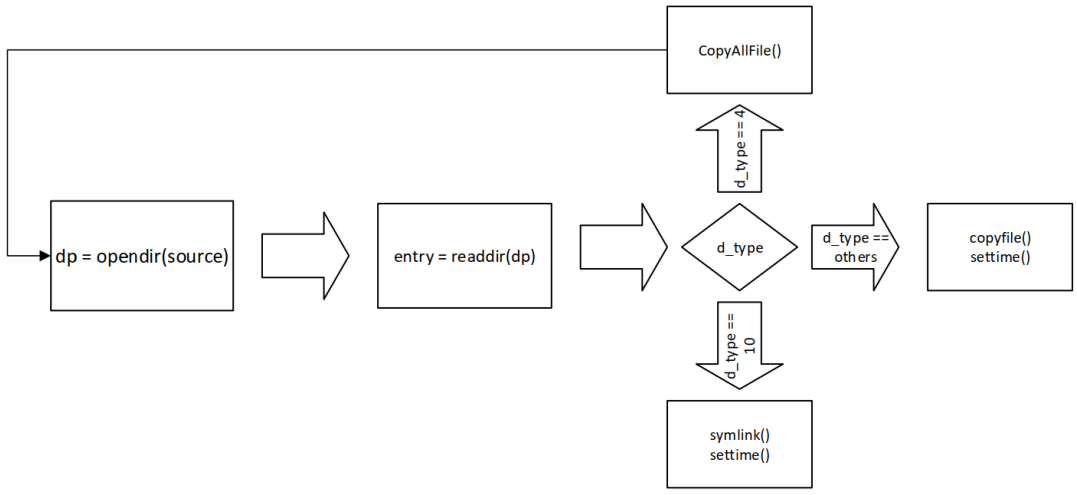

图 5-3-1

函数中先定义了字符串，用于临时存储源路径名和目标路径名；

定义了 DIR *dp 的指针，用于指向 opendir(source)返回的指针，opendir()API 在前边的简单的接口介绍中已经有说明，得到一个指向文件的指针，定义了 dirent 的指针

entry,存储 readdir(dp)的返回值，dirent 结构体代表了由 dir 指向的目录流中的下一个目录项；如果读到 end-of-file 或者出现错误，则返回 NULL。

接下来是循环读取文件过程。

entry 指向了当前文件的信息：

```
struct dirent {
ino_t d_ino;        /* inode number */
off_t d_off;        /* offset to the next dirent */
unsigned short d_reclen; /* length of this record */
unsigned char d_type;                //文件的类型
char d_name[256];                    //文件名称
};
```

代码 5-3-2

其中，结构体中的 d_type 定义了文件的类型，这和 windows 系统中的调用存在差别，这里可以直接根据 d_type 判断文件的类型：

```
enum
{
    DT_UNKNOWN = 0,
 # define DT_UNKNOWN
DT_UNKNOWN
     DT_FIFO = 1,
 # define DT_FIFO DT_FIFO
     DT_CHR = 2,
 # define DT_CHR DT_CHR
     DT_DIR = 4,
 # define DT_DIR DT_DIR
     DT_BLK = 6,
 # define DT_BLK DT_BLK
     DT_REG = 8,
 # define DT_REG DT_REG
     DT_LNK = 10,
 # define DT_LNK DT_LNK
     DT_SOCK = 12,
 # define DT_SOCK DT_SOCK
     DT_WHT = 14
 # define DT_WHT DT_WHT
};
```

代码 5-3-3

根据上述的类型，我们需要使用的特殊的文件类型有两种，d_type == 4 为目录类型，d_type == 10 为链接文件，其他所有的文件都是可以直接拷贝的。

如果是文件夹，即判断当前的 d_type == 4，则还存在递归，和 Windows 下的方式相同，首先将路径转换，并且在目标路径下建立文件夹，在 Linux 下建立文件夹的方式是调用 mkdir()的 API，在递归退栈后，设置时间。具体的代码可以阅读代码 5-3-1 中的 d_type == 4 的部分。

在判断当前不是文件夹和链接文件时，则直接调用 mycopy()函数，拷贝文件。

拷贝链接文件和 windows 下有较大区别。

在处理链接文件时（在本实验中是指软链接文件），根据在 5.1 中提到的内容，不能和其他的文件一样调用拷贝函数。首先使用 readlink()将源链接文件的指向的文件内容拷贝到缓冲区，使用 symlink()将目标链接文件指向缓冲区。

d_type == others，表示此时是一般文件，调用 copyFile()函数，拷贝文件，实现方式与 windows 系统下类似。

Linux 系统下设置相同的时间是需要说明的。

这一块有很多同学都遇到错误，时间总是会有偏差，甚至我看到有一段代码每次总是会相差一个固定的时间，所以他们的处理办法在拷贝了时间之后，加上一个固定的时间差，这样回复源文件的时间。我仔细查看了代码之后，发现了最大的差别在于使用的设置时间的函数的不同，对此，我给出了我设置时间的函数，如下代码片段所示：

```
void SetTime(char * source, char* target)
{
    struct stat statbuf;
    lstat(source, &statbuf);
    struct timespec filetime[2] = {statbuf.st_atim,statbuf.st_mtim};     //得到源文件的相关时间
    //纳秒级数量级改变时间戳
    //通过两个时间来改变时间戳，最后访问时间和最后改动时间
    utimensat(AT_FDCWD,target, filetime,AT_SYMLINK_NOFOLLOW);            //通过filetime[2]改变目标文件时间
}
```

代码 5-3-4

我的思路是首先将文件信息存在一个缓冲区中，然后取出源文件的时间信息，存在 filetime 中。调用接口 utimensat()来设置目标文件的时间，这里与大家是有差别的，我通过实现发现，使用 utimensat（）拷贝时间时，没有差别。

六、实验结果与感想

6.1 Windows 系统下的实验结果截图

我拷贝了 F 盘下的一个一个文件夹

拷贝的结果如下，可以看到拷贝的文件的信息：

```
F:\CODE\os\cmake-build-debug>os_5.exe "F:\pdf&Res" F:\test
CopyFile 553280 计算机网络（第7版）-谢希仁.pdf
CreateDirectory F:\test\books
    CopyFile [量化] 宽客人生：华尔街的数量金融大师.pdf
    CopyFile 北鸢.mobi
    CopyFile 逻辑学导论(第十一版).pdf
CopyFile C Primer Plus _pdf.pdf
CopyFile C++Primer.pdf
CopyFile lab1.ASM
CopyFile Python 基础编程 第二版.pdf
CopyFile Python 核心编程 第三版.pdf
CopyFile 大数据时代.mobi
CopyFile 大话数据结构.epub
CopyFile 深入理解计算机网络-王达.mobi
CopyFile 深入理解计算机网络-王达.pdf
CopyFile 马克思1844年经济学哲学手稿.pdf
CopyFile 鼠疫-阿尔贝·加缪.mobi
```

图 6-1-1

拷贝完之后，对比源文件与目标文件的信息：

```
PS F:\> ls '.\pdf&Res\'


    目录: F:\pdf&Res


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2018/1/12     23:44                books
-a----        2018/4/22     17:13       70809807 553280 计算机网络（第7版）-谢希仁.pdf
-a----        2017/7/11      8:48       54963687 C Primer Plus _pdf.pdf
-a----        2017/7/11      8:53      154458151 C++Primer.pdf
-a----        2018/2/28     20:48           2308 lab1.ASM
-a----        2017/7/18     10:15       57902896 Python 基础编程 第二版.pdf
-a----        2017/7/17     14:17       23442258 Python 核心编程 第三版.pdf
-a----         2018/2/6      6:32        1061398 大数据时代.mobi
-a----        2018/4/22     18:03       12085241 大话数据结构.epub
-a----         2018/3/7     15:32       13454746 深入理解计算机网络-王达.mobi
-a----        2018/2/26     18:30       22203943 深入理解计算机网络-王达.pdf
-a----         2018/3/4      9:24        4638342 马克思1844年经济学哲学手稿.pdf
-a----        2018/2/22      7:37         510642 鼠疫-阿尔贝·加缪.mobi
```

图 6-1-2

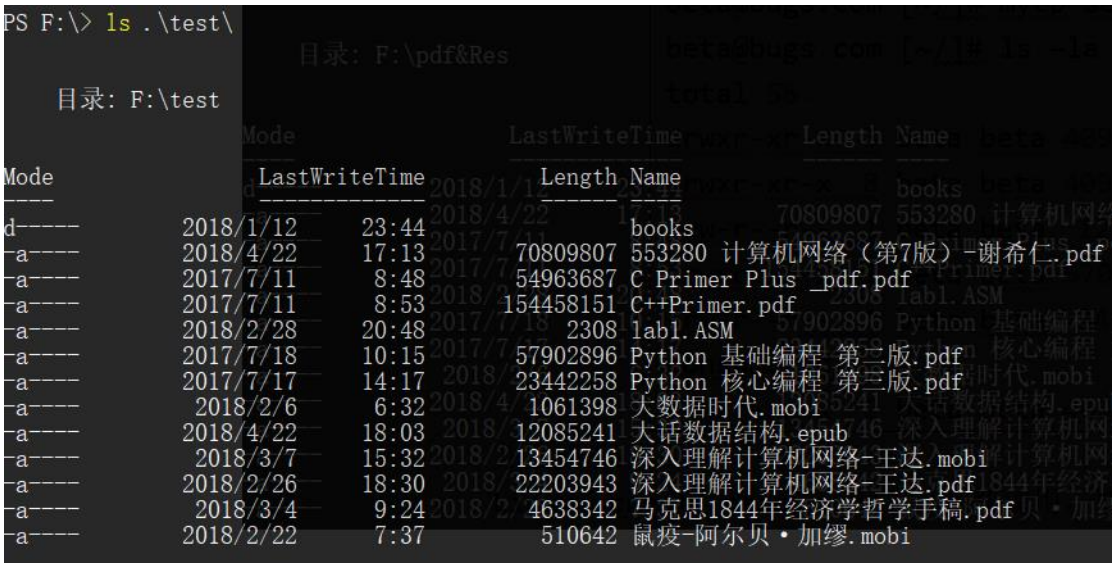

图 6-1-3

通过对比，发现源文件与目标文件的信息，包括修改时间与读取权限等完全一致。

6.2 在 Liunx 下的测试结果

首先在需要复制的文件夹中创建硬链接和软连接文件

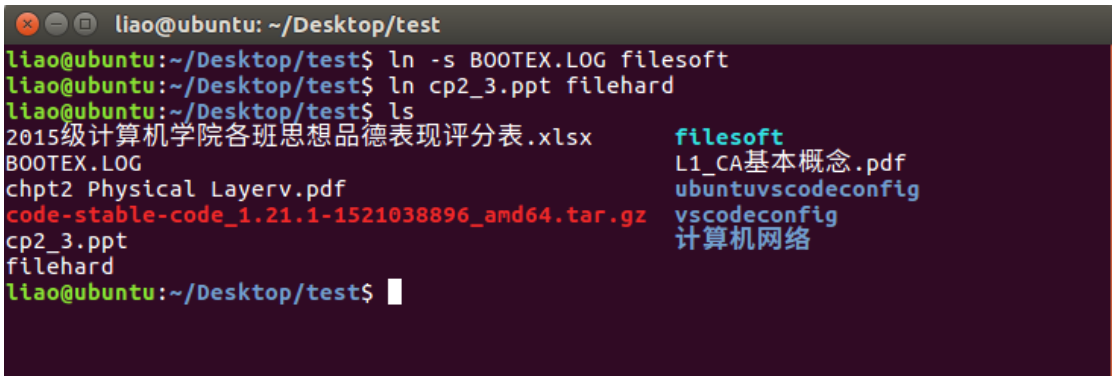

图 6-2-1

运行程序，拷贝在桌面下的一个文件，拷贝过程中的具体信息如下：

```
liao@ubuntu:~/vsc/OS/exp5$ ./mycopy /home/liao/Desktop/test /home/liao/Desktop/test1
Copyfile: /home/liao/Desktop/test/code-stable-code_1.21.1-1521038896_amd64.tar.gz
Copyfile: /home/liao/Desktop/test/cp2_3.ppt
Copyfile: /home/liao/Desktop/test/BOOTEX.LOG
Copyfile: /home/liao/Desktop/test/filehard
Copyfile: /home/liao/Desktop/test/2015级计算机学院各班思想品德表现评分表.xlsx
Copyfile: /home/liao/Desktop/test/chpt2 Physical Layerv.pdf
Copyfolder: /home/liao/Desktop/test/计算机网络
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH2-1.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/course info.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH7.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH1-1.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网作业.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH2-2.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH5-1.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH5-2.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH4.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH1-2.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH6.ppt
    Copyfile: /home/liao/Desktop/test/计算机网络/计网CH3.ppt
Copyfolder: /home/liao/Desktop/test/vscodeconfig
    Copyfile: /home/liao/Desktop/test/vscodeconfig/tasks.json
    Copyfile: /home/liao/Desktop/test/vscodeconfig/settings.json
    Copyfile: /home/liao/Desktop/test/vscodeconfig/c_cpp_properties.json
    Copyfile: /home/liao/Desktop/test/vscodeconfig/launch.json
    Copyfolder: /home/liao/Desktop/test/vscodeconfig/hah
        Copyfolder: /home/liao/Desktop/test/vscodeconfig/hah/liao
            Copyfile: /home/liao/Desktop/test/vscodeconfig/hah/liao/c_cpp_properties.json
        Copyfile: /home/liao/Desktop/test/vscodeconfig/hah/c_cpp_properties.json
Copyfolder: /home/liao/Desktop/test/ubuntuvscodeconfig
    Copyfile: /home/liao/Desktop/test/ubuntuvscodeconfig/tasks.json
    Copyfile: /home/liao/Desktop/test/ubuntuvscodeconfig/launch.json
Copyfile: /home/liao/Desktop/test/L1_CA基本概念.pdf
```

图 6-2-2

拷贝之后，查看源文件与目标文件的具体信息：

```
liao@ubuntu:~/Desktop$ ls -la test
总用量 99792
drwxrwxr-x 5 liao liao     4096 4月  30 19:26 .
drwxr-xr-x 5 liao liao     4096 4月  30 19:39 ..
-rw-r--r-- 1 liao liao    47144 3月  16 02:27 2015级计算机学院各班思想品德表现评分表.xlsx
-rw-r--r-- 2 liao liao     4508 3月  19 15:15 BOOTEX.LOG
-rw-r--r-- 1 liao liao  9852890 3月  13 11:17 chpt2 Physical Layerv.pdf
-rw-r--r-- 1 liao liao 67065892 3月  23 11:50 code-stable-code_1.21.1-1521038896_amd64.tar.gz
-rw-r--r-- 2 liao liao  9824256 3月  26 14:56 cp2_3.ppt
-rw-r--r-- 2 liao liao  9824256 3月  26 14:56 filehard
lrwxrwxrwx 1 liao liao       10 4月  30 19:25 filesoft -> BOOTEX.LOG
-rw-r--r-- 1 liao liao  5532400 3月  13 11:17 L1_CA基本概念.pdf
drwxr-xr-x 2 liao liao     4096 3月  23 13:12 ubuntuvscodeconfig
drwxr-xr-x 3 liao liao     4096 3月  27 08:02 vscodeconfig
drwxr-xr-x 2 liao liao     4096 3月  19 13:34 计算机网络
liao@ubuntu:~/Desktop$ ls -la test1
总用量 99792
drwxrwxr-x 5 liao liao     4096 4月  30 19:26 .
drwxr-xr-x 5 liao liao     4096 4月  30 19:39 ..
-rw-r--r-- 1 liao liao    47144 3月  16 02:27 2015级计算机学院各班思想品德表现评分表.xlsx
-rw-r--r-- 1 liao liao     4508 3月  19 15:15 BOOTEX.LOG
-rw-r--r-- 1 liao liao  9852890 3月  13 11:17 chpt2 Physical Layerv.pdf
-rw-r--r-- 1 liao liao 67065892 3月  23 11:50 code-stable-code_1.21.1-1521038896_amd64.tar.gz
-rw-r--r-- 1 liao liao  9824256 3月  26 14:56 cp2_3.ppt
-rw-r--r-- 1 liao liao  9824256 3月  26 14:56 filehard
lrwxrwxrwx 1 liao liao       10 4月  30 19:25 filesoft -> BOOTEX.LOG
-rw-r--r-- 1 liao liao  5532400 3月  13 11:17 L1_CA基本概念.pdf
drwxr-xr-x 2 liao liao     4096 3月  23 13:12 ubuntuvscodeconfig
drwxr-xr-x 3 liao liao     4096 3月  27 08:02 vscodeconfig
drwxr-xr-x 2 liao liao     4096 3月  19 13:34 计算机网络
liao@ubuntu:~/Desktop$
```

图 6-2-3

通过对比，返现源文件与目标文件的信息，包括修改时间与读取权限完全一致。

6.3 实验总结与感想

本次实验，自己编写了 windows 系统下和 Linux 系统下的拷贝函数，实现了两个系统下的文件拷贝的基本功能，并且通过了实验的要求，拷贝前后的文件属性——文件的访问权限，文件的修改时间等都没有发生改变。

实现过程中遇到了一些困难，比如如何递归实现对于所有文件的遍历，如何保证拷贝后的文件的时间属性与源文件保持一致等，还有对于 Linux 系统下的链接文件的概念理解与拷贝的实现等，一开始都出现了问题，后来通过参照课本的讲解以及参照网上的一些博客等，实现了相应的功能。通过这个实验，我基本掌握了如何在 windows 系统中和 Linux 中如何编写拷贝文件的方法以及拷贝文件的原理，为以后进一步学习相应的知识打下了基础。