**实验五 文件复制**

班级：07121502 学号：1320151098 姓名：赵璐

# 一、实验目的

## 熟悉 Windows 和 Linux 环境下的文件操作相关系统 API 的功能及作用，并尝试完成简单的文件操作，对文件目录进行操作。

**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](mailto:beta@bugs.com) [~/]# ls –l 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](mailto:beta@bugs.com) [~/]# mycp sem target [beta@bugs.com](mailto:beta@bugs.com) [~/]# ls –l 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；

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

**三、程序设计及实现**

**3.1实验基本思路**

使用文件（夹）复制函数在给定的文件或文件夹目录中进行遍历：

1. 如果遍历到的是单一文件，则直接将源文件复制到目标目录下。
2. 如果遍历到的是文件夹，那么就在目标目录中创建一个和源文件夹相同名称的文件夹。并且再以遍历到的文件夹路径作为源目录参数，将新建的文件夹目录作为目标路径递归调用文件（夹）复制函数。
3. 当源文件（夹）被遍历结束后，程序完成全部文件的复制， 结束运行。

**3.2 Windows 调用系统API**

a)SetFileTime()

功能：设置文件的创建、访问及上次修改时间。

原型：

BOOL

WINAPI

SetFileTime(

\_In\_ HANDLE hFile,//系统文件句柄

\_In\_opt\_ CONST FILETIME \* lpCreationTime,//文件创建时间

\_In\_opt\_ CONST FILETIME \* lpLastAccessTime,//文件上一次访问时间

\_In\_opt\_ CONST FILETIME \* lpLastWriteTime//文件最近一次修改时间

);

b)GetFileTime()

功能：取得指定文件的时间信息

原型：

BOOL WINAPI

GetFileTime(

\_In\_ HANDLE hFile,//系统文件句柄

\_Out\_opt\_ LPFILETIME lpCreationTime,//用于装载文件的创建时间

\_Out\_opt\_ LPFILETIME lpLastAccessTime,//用于装载文件的上一次访问时

\_Out\_opt\_ LPFILETIME lpLastWriteTime//用于装载文件上一次修改时间

);

c)FindFirstFile()

功能：根据文件名查找文件。

原型：

HANDLE WINAPI

FindFirstFileA(

\_In\_ LPCSTR lpFileName, //将搜索的文件名

\_Out\_ LPWIN32\_FIND\_DATAA lpFindFileData//用于装载与找到的文件有关的信息的结构体

);

其中，LPWIN32\_FIND\_DATAA 结构体的定义如下

typedef struct \_WIN32\_FIND\_DATAA {

DWORD dwFileAttributes;//文件属性

FILETIME ftCreationTime;//文件的创建日期

FILETIME ftLastAccessTime;//文件的最后存取日期

FILETIME ftLastWriteTime;//文件最后的修改日期

DWORD nFileSizeHigh;//文件长度的高32位

DWORD nFileSizeLow;//文件长度的低32位

DWORD dwReserved0;//内部使用

DWORD dwReserved1;//内部使用

\_Field\_z\_ CHAR cFileName[ MAX\_PATH ];//本次找到的文件名

\_Field\_z\_ CHAR cAlternateFileName[ 14 ];//文件的短文件名

#ifdef \_MAC

DWORD dwFileType;

DWORD dwCreatorType;

WORD wFinderFlags;

#endif

} WIN32\_FIND\_DATAA, \*PWIN32\_FIND\_DATAA, \*LPWIN32\_FIND\_DATAA;

d)FindNextFile()

功能：根据调用 FindFirstFile 函数时指定的一个文件名查找下一个文件

原型：

BOOL WINAPI

FindNextFileA(

\_In\_ HANDLE hFindFile,//上一次查找返回的句柄

\_Out\_ LPWIN32\_FIND\_DATAA lpFindFileData//用于装载与找到的文件有关的信息的结构体

);

3.3 Linux调用系统API

a)opendir

功能：用来打开参数 name 指定的目录, 并返回 DIR\*形态的目录流。

原型：DIR \* opendir(const char \* name);//要打开的文件路径

b)readdir

功能：eaddir 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 或者出现了错误，那么返回NULL。

原型：struct dirent \*readdir(DIR \*dir);

其中，dirent结构体定义如下：

struct dirent{

ino\_t d\_ino;

off\_t d\_off;

unsigned short d\_reclen;

undigned char d\_type;

char d\_name[256];

}

c)mkdir()

功能：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 errno 值。

原型：

int mkdir( const char \*path, //要创建的目录名

mode\_t mode); //目录权限

d)stat()

功能：通过文件名 filename 获取文件信息，并保存在 buf 所指的结构体中

原型：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

其中，stat的结构体内容为：

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)

};

e)utime()

功能：用来修改参数 filename 文件所属的 inode 存取时间。

原型：int utime(const char \* filename, struct utimbuf \* buf);

其中结构体 utimbuf 的定义如下：

struct utimbuf

{

time\_t actime;/\*access time\*/

time\_t modtime; /\* modify time\*/

};

f)rendlink()

功能：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。

原型：int readlink(const char \* path, char \* buf, size\_t bufsiz);

g)creat()

功能：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

原型：int creat(const char \*pathname, mode\_t mode);

h)write()

功能：调用 write 函数向打开的文件写入数据

原型：ssize\_t write(int filedes, void\* buff, size\_t nbytes);

1. open()

功能：调用 open 函数打开或者创建一个文件

原型：int open(const char \*pathname, int flag);

**四、实验结果及分析**

**4.1 Windows**

1.在文件夹lab5中，将lab1复制一份为lab2，起初文件夹中不存在lab2。

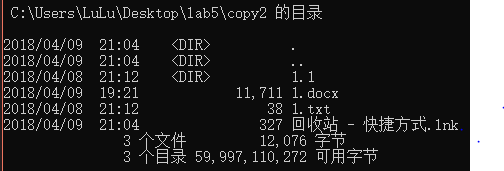
使用命令 #文件复制 ./copy1 ./copy2



2.copy1的目录



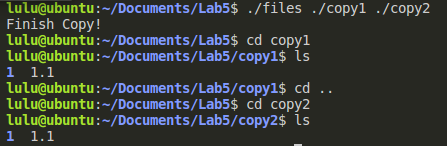
3.copy2的目录



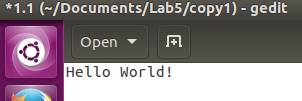
**4.2 Linux**

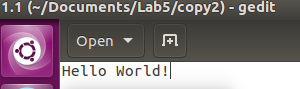
1.在文件夹lab5中，将lab1复制一份为lab2，起初文件夹中不存在lab2。

使用命令 #./files ./copy1 ./copy2



2.文档中的内容也被成功复制





**五、实验心得**

进行了本实验后我进一步了解了 Windows 和 Linux 中系统调用函数的应用，接触到了很多的系统对文件的操作。并且在实验过程中，通过调试各个 API，逐渐了解到系统预定义的很多结构体以及函数调用是真的非常方便。