一、实验目的

1.熟悉 Windows 和 Linux 中的文件操作

(1).相关系统调用

(2). 对于文件夹、软链接等不同类型文件的处理

2. 对文件属性信息的设置

二、实验内容

在 Windows 和 Linux 中，设计一个文件复制命令 mycp,能够复制所指定文件夹下的所有内容到用户指定的新文件夹，包括文件夹下的子文件夹以及链接文件。

三、实验环境及配置方法

Windows:

OS： Windows10

IDE:Visual Studio 2019

Linux:

Os:Ubuntu 20.10 groovy gorilla

Visual studio code and gcc

四、实验方法和实验步骤（程序设计与实现）

Windows 复制命令 mycp

实现逻辑：用户从命令行中给出源文件夹以及目标文件夹路径，接收到路径后程序创建相应的目标文件夹，之后开始递归复制过程，使用 FindFirstFile 以及 FindNextFile 系统调用检索当前文件夹下的文件，如果是文件则将其复制到指定文件夹，如果是文件夹则调用自身，并传入调用前的路径来维护相对路径信息从而在正确的位置创建文件或文件夹。

相关系统调用及数据结构

CreateDirectory

```
BOOL CreateDirectoryA(
  LPCSTR                lpPathName,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

使用该调用，需要传入文件夹路径以及安全参数，本实验中将该调用封装如下：

```
bool DirectoryCreate(char dir[])
{
    if (CreateDirectory(dir, NULL))
    {
        printf("%s has been created.\n", dir);
        return true;
    }
    printf("failed to create directory %s\n", dir);
    return false;
}
```

FindFirstFile

```
HANDLE FindFirstFileA(
  LPCSTR             lpFileName,
  LPWIN32_FIND_DATAA lpFindFileData
);
```

FindNextFile

```
BOOL FindNextFileA(
  HANDLE             hFindFile,
  LPWIN32_FIND_DATAA lpFindFileData
);
```

WIN32*FIND*DATA

以上两个系统调用和一个数据结构用来获取文件信息。

WIN32*FIND*DATA 结构定义如下：

```
typedef struct _WIN32_FIND_DATA {
DWORD dwFileAttributes;
FILETIME ftCreationTime;
FILETIME ftLastAccessTime;
FILETIME ftLastWriteTime;
DWORD nFileSizeHigh;
DWORD nFileSizeLow;
DWORD dwOID;
TCHAR cFileName[MAX_PATH];
} WIN32_FIND_DATA;
```

本次实验中主要用到 dwFileAttributes 成员来判断该文件是否是一个文件夹以及 cFileName 来获取文件名。

FindFirstFile 与 FindNextFile 的用法与实验四中所用到的遍历进程的

Process32First、Process32Next 类似。

FindFirstFile 将给出的路径的文件信息写入 WIN32FINDDATA 结构中，之后返回一个文件句柄，FindNextFile 将传入的句柄所对应的文件写入 WIN32DATAFIND 结构，从而实现了文件夹的遍历。

实验中递归调用代码如下：

```
if ((FileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 && strcmp
(FileData.cFileName, ".") != 0 && strcmp(FileData.cFileName, "..") != 0)
	{
		//对于目录项且不为.\..

		strcpy(writeDirectoryName, currRdir);		//创建目录
		strcat(writeDirectoryName, "/");
		strcat(writeDirectoryName, FileData.cFileName);
		DirectoryCreate(writeDirectoryName);

		strcpy(fileSearch, dir);	//编辑进入下一层文件夹的格式
		strcat(fileSearch, "/");
		strcat(fileSearch, FileData.cFileName);

		strcpy(beforeEntry, currRdir);		//维护相对目录信息
		strcat(currRdir, FileData.cFileName);
		strcat(currRdir, "/");


		RecursionClone(fileSearch,beforeEntry);		//递归进入子文件夹
```

CopyFile

```
BOOL CopyFile(
  LPCTSTR lpExistingFileName,
  LPCTSTR lpNewFileName,
  BOOL    bFailIfExists
);
```

win32 Api 中的复制文件函数对于本次实验可谓是提供了不少帮助，省去了不必要的打开原文件，读取并将其写入新文件，之后再修改新创建的文件的属性信息的步骤。将原文件路径以及新文件路径传入，并可以做到完美的复制，复制产生的文件具有与原文件相同的属性信息。

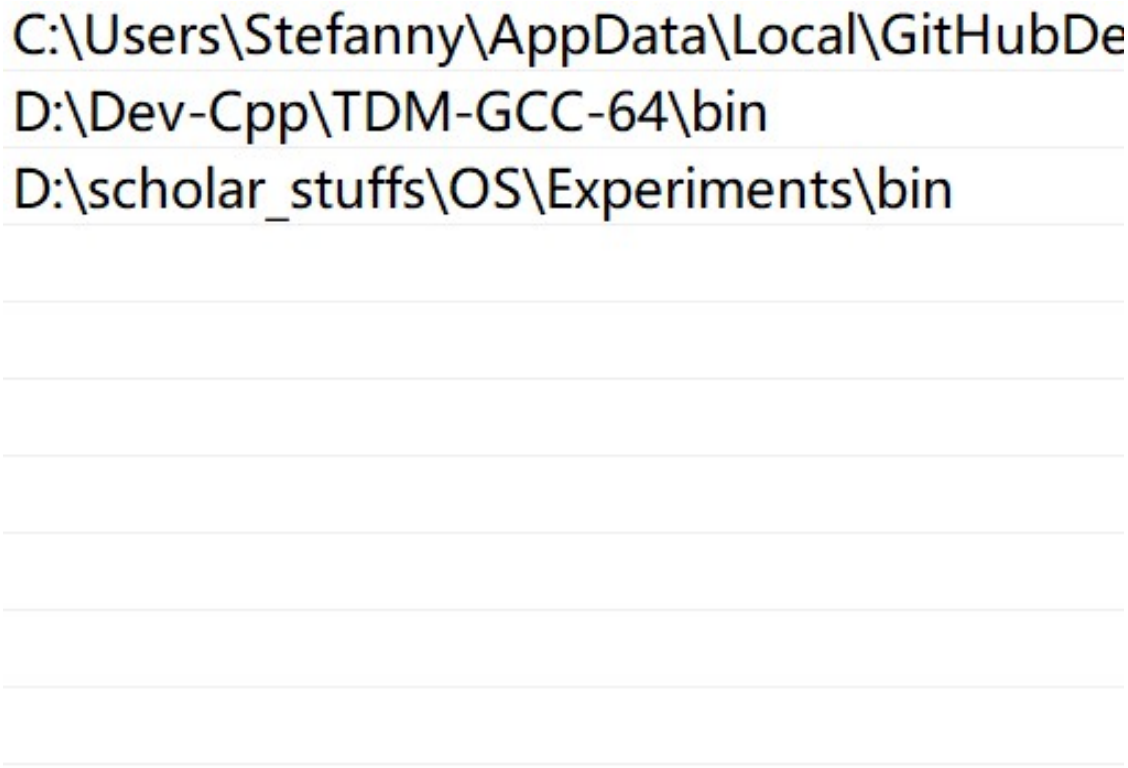
实验中复制文件代码如下：

```
if (strcmp(FileData.cFileName, ".") == 0 || strcmp(FileData.cFileName, ".."
") == 0)
		{
```

```
            continue;
        }
        //对非目录文件
        strcpy(writeFileName, currRdir);
        strcat(writeFileName, "");
        strcat(writeFileName, FileData.cFileName);

        strcpy(readFileName, dir);
        strcat(readFileName, "/");
        strcat(readFileName, FileData.cFileName);

        printf("original file: %s\n", readFileName);
        //复制文件，最后的参数设置为false 表示允许覆盖已有文件
        if (!CopyFile(readFileName, writeFileName, false))
        {
            printf("Could not copy file.%d\n", GetLastError());

        }
        else
        {
            printf("Copy completed.\n");
        }
```

将可执行文件添加到环境变量

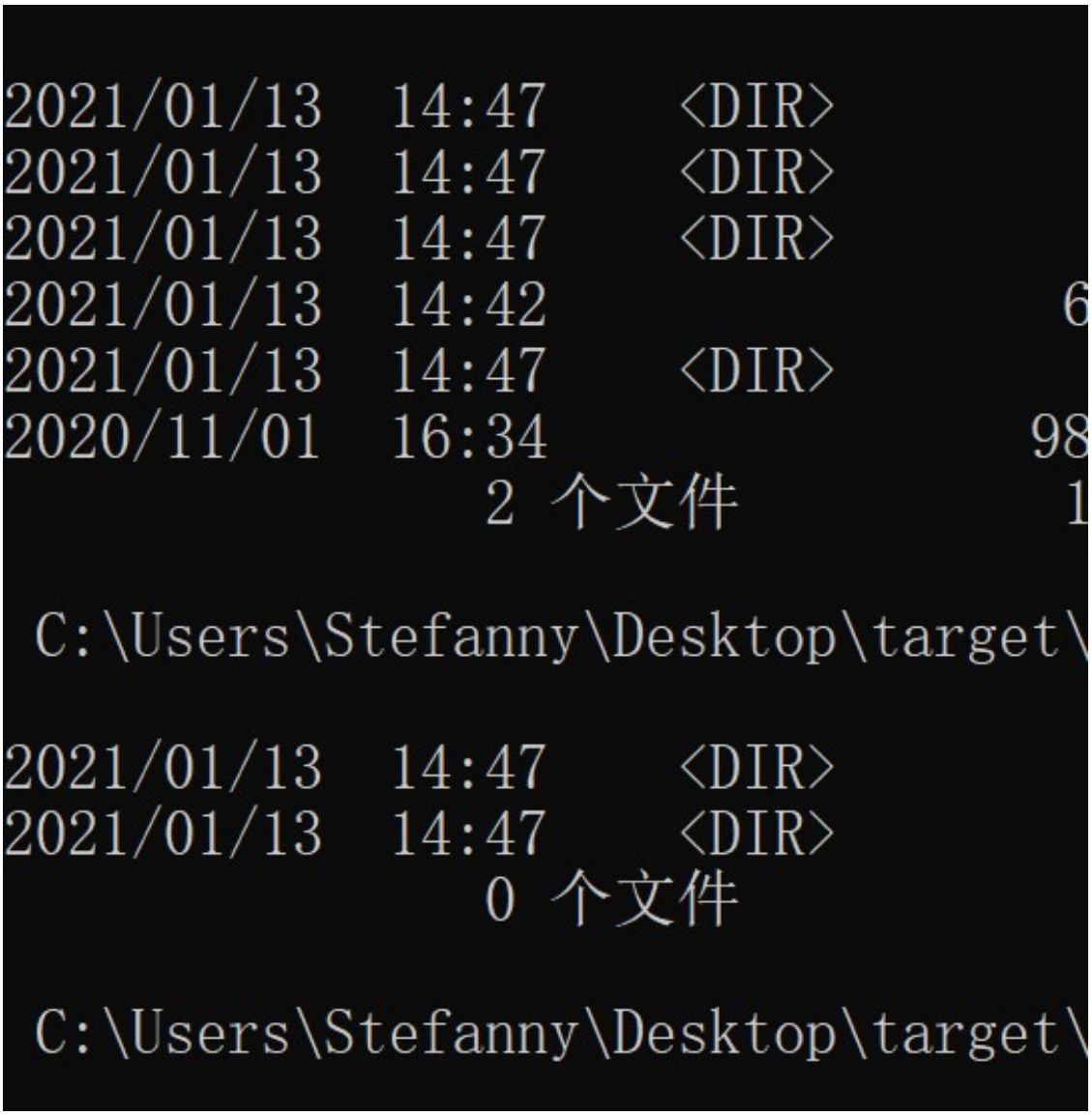
将编译好的程序 mycp.exe 放到一个我自己创建的 bin 文件夹中，并将其加入环境变量



之后即可在 cmd 中使用 mycp 命令来拷贝文件夹。

复制效果


2021/01/13 14:44 <DIR> .
2021/01/13 14:42 <DIR> a
2021/01/13 14:42 65 a
2021/01/13 14:44 <DIR> D
2020/11/01 16:34 988 N
2 个文件 1,05
D:\sem\anotherDirectory 的目录
2021/01/13 14:42 <DIR> .
2021/01/13 14:42 <DIR> .
0 个文件
D:\sem\Directory 的目录
2021/01/13 14:44 <DIR> .
2021/01/13 14:44 <DIR> .
2021/01/13 14:44 <DIR> o
2021/01/13 14:43 17 q
1 个文件 1
D:\sem\Directory\ooo 的目录



Linux 复制命令 mycp

实现逻辑：与 Windows 中的实现方式答题相同，不同在于此次复制过程采用了打开原文件与写入文件，读取原文件并写入新文件，于是便有了修改新文件信息以保持一致的过程以及对于符号链接的单独处理。

相关系统调用及数据结构

lstat

```
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
```

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

通过文件名查到文件的状态信息并将其存入 stat 结构中。

stat 结构定义如下：

```
struct stat {
               dev_t     st_dev;         /* ID of device containing file */
               ino_t     st_ino;         /* Inode number */
               mode_t    st_mode;        /* File type and mode */
               nlink_t   st_nlink;       /* Number of hard links */
               uid_t     st_uid;         /* User ID of owner */
               gid_t     st_gid;         /* Group ID of owner */
               dev_t     st_rdev;        /* Device ID (if special file) */
               off_t     st_size;        /* Total size, in bytes */
               blksize_t st_blksize;     /* Block size for filesystem I/O */

               blkcnt_t  st_blocks;      /* Number of 512B blocks allocated
*/

               /* Since Linux 2.6, the kernel supports nanosecond
                  precision for the following timestamp fields.
                  For the details before Linux 2.6, see NOTES. */

               struct timespec st_atim;  /* Time of last access */
               struct timespec st_mtim;  /* Time of last modification */
               struct timespec st_ctim;  /* Time of last status change */

           #define st_atime st_atim.tv_sec      /* Backward compatibility *
/
           #define st_mtime st_mtim.tv_sec
           #define st_ctime st_ctim.tv_sec
           };
```

mkdir

```
#include <sys/stat.h>

int mkdir(const char *path, mode_t mode);
```

函数将创建一个名字一个新的目录路径。新目录的文件许可位应从 mode 初始化 。模式自变量的这些文件许可位应由进程的文件创建掩码来修改。

opendir

```
#include<sys/types.h>
#include<dirent.h>
```

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

用于打开一个目录并建立一个目录流，如果成功，返回一个指向 DIR 结构的指针，该指针用于读取目录数据项

readdir

```
#include<sys/types.h>
#include<dirent.h>

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

返回一个 dirent 结构的指向目录流 dirp 中下一个目录项的有关资料，如果发生错误或者达到目录尾，readdir 将返回 NULL，

closedir

```
#include<sys/types.h>
#include<dirent.h>

int closedir(DIR *dirp);
```

关闭一个目录流并释放与之关联的资源。

symlink

```
#include<unistd.h>

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

创建一个 path2 目录项指向 path1 目录项的符号链接

readlink

```
#include <unistd.h>

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

readlink()将符号链接路径的内容放入大小为 bufsiz 的缓冲区 buf 中。readlink（）不会将空字节附加到 buf。万一缓冲区太小而无法容纳所有内容，它将截断内容（长度为 bufsiz 字符）。

实验中对软链接的处理代码如下：

```
if(S_ISLNK(statbuf.st_mode))        //单独处理符号链接
{
    char readfileName[MAX_DIR_LENGTH];  //当前的读取名
    char writelinkedfile[MAX_DIR_LENGTH];   //被链接的文件
```

```
    strcpy(readfileName,readdirName);     //构造写入文件时的路径
    strcat(readfileName,"/");
    strcat(readfileName,entry->d_name);

    //printf("link:%s\n",readfileName);
    char buf[MAX_DIR_LENGTH];       //存储链接指向的文件信息
    if(readlink(readfileName,buf,sizeof(buf))<0)
    {
    perror("failed to read symbol link.\n");
    exit(EXIT_FAILURE);
    }
    //printf("%s\n",buf);
    strcpy(writelinkedfile,writedirName);
    strcat(writelinkedfile,"/");
    strcat(writelinkedfile,buf);

    //读取到链接指向文件之后，先创建对应文件再创建链接
    in=open(buf,O_RDONLY);  //打开被指向的文件
    out=open(writelinkedfile,0775);
    while((nread=read(in,block,sizeof(block)))>0)
    {
    write(out,block,nread);
    }

    //printf("%s\n%s\n",writelinkedfile,writefileName);

    //创建完被链接文件之后创建链接
    if(symlink(buf,writefileName)<0)
    {
    //perror("failed to create symbol link.\n");
    //exit(EXIT_FAILURE);
    continue;
    }
    //修改文件权限及属性信息
    timeNmode(statbuf);

}
```

read

write

open

以上三个调用为打开文件，写入以及读取功能。

实验中的复制过程如下：

```
in=open(entry->d_name,O_RDONLY);    //打开文件
out=open(writefileName,0775);
while((nread=read(in,block,sizeof(block)))>0)
{
write(out,block,nread);
}

//修改文件权限及属性信息
timeNmode(statbuf);
```

utime

chmod

以上两个系统调用为解决文件属性问题，utime 可以设置文件的上次访问以及修改时间，而 chmod 更改文件的访问权限，实验中将两个调用封装在 timeNmode 函数中以一次性更改文件的权限以及时间信息。

```
void timeNmode(struct stat statbuf)          //修改权限以及时间信息
{
    struct utimbuf timebuf;      //存储时间信息
    mode_t new_mode;             //存储权限信息

    timebuf.actime=statbuf.st_atime;
    timebuf.modtime=statbuf.st_mtime;
    new_mode=statbuf.st_mode;

    int modtime=utime(replacement,&timebuf);
    if(modtime==-1)
    {
        perror("failed to change time info.\n");
        exit(EXIT_FAILURE);
    }
    int modmode=chmod(replacement,new_mode);
    if(modmode==-1)
    {
        perror("failed to change mode info.\n");
        exit(EXIT_FAILURE);
    }
```

```
}
```

将可执行文件添加到环境变量

修改.bashrc 文件

在最后一行写入

```
export PATH="$PATH:~/me_codez/LinuxFile/bin"s
```

五、实验结果和分析

本次实验结果，基本符合预期，在 Windows 中可以完美完成文件夹的递归复制以及对文件信息等属性的保留。但是在 Linux 中的复制命令在处理软链接时会遇到复制过后文件属性信息无法保留的情况，该问题暂时还未得到解决，有待日后完善。

对于有多种功能需求的小型应用，应该从基本框架开始，先实现最基本的功能，再逐步添加对于特殊情况下使用的功能。对于本次实验来说，想要复制一个文件夹及其子文件夹中的所有文件，可以先实现复制文件以及文件夹的功能，再调整完善加入递归复制，对软链接的处理，以及对自己创建的文件的属性信息的修改设置等等。

此外，对于每个功能的检验需要编写良好的测试用例。在对 Windows 复制的功能进行测试时，我创建了一个如下结构的文件夹。

由于文件夹下只有一个子文件夹 DDD，经过测试并没有发现程序在处理相对 目录时的逻辑错误，之后由于巧合我又在 for_test 文件夹下新建了子文件夹 EEE ，发现对 EEE 的创建会发生到 DDD 子文件夹中，之后修改了代码逻辑解决了问题。