实验名称	实验四：复制文件		
学号	1120220576	姓名	宋宇翔

一、实验目的

学习 Linux 与 Windows 关于文件读取的库函数，了解操作系统中文件管理功能的具体实现方法与储存结构。

二、实验内容

编写一个文件复制程序，要求能够复制普通文件、嵌套文件夹(文件夹里有文件夹)、链接文件（Linux）/快捷方式文件（Windows）。

Linux: creat，read，write 等系统调用

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

三、实验环境及配置方法

操作系统分别为 Windows11，Ubuntu22.04。

四、实验方法和实验步骤（程序设计与实现）

1. Linux 系统

对于有嵌套文件夹以及链接文件（Linux）的文件复制，需要一定的递归结构遍历整个文件。幸运的是，对于 Linux 系统，readdir 会自动的遍历给定路径下的所有文件以及文件夹，因此只需对这些不同的文件类型进行判断，然后循环复制即可。具体的实现代码（普通文件）如下。

```
    memset(destination,0,sizeof(destination));
    strcpy(destination, dest);
    strcat(destination, "/");
        struct dirent *entry;
    while ((entry = readdir(dir)) != NULL)//遍历源目录
    {
        else // 普通文件
        {
            //构造路径
                        memset(original_path, 0, sizeof(original_path));
                        strcpy(original_path, source);
                        strcat(original_path, "/");
                        strcat(original_path, entry->d_name);
```

```
                        memset(destination,0,sizeof(destination));
                        strcpy(destination, dest);
                        strcat(destination, "/");
                        strcat(destination, entry->d_name);
                        //复制软链接
                        CopyFile(original_path, destination);
                        //同步信息
                        SyncFile(original_path,destination);
        }
    }
```

CopyFile 的函数具体定义如下，通过建立缓冲区数组，并依次写入将缓冲区数据目标路径即可。

```
void CopyFile(char *source, char *target) // 直接复制
{
    //打开与创建文件
        struct stat statbuf;
    stat(source, &statbuf);
        int _source = open(source, 0); //打开
    int _target = creat(target, statbuf.st_mode); //创建

        //传输文件
        char BUFFER[KB];
        int wordbit;
    while ((wordbit = read(_source, BUFFER, KB)) > 0)
    {
        //写入
        if (write(_target, BUFFER, wordbit) != wordbit)
        {
            printf("error when writing buffer!\n");
            exit(-1);
        }
    }

        //关闭文件
    close(_source);
    close(_target);
}
```

对于链接文件，需要调用 readlink 和 symlink 函数进行处理，具体代码如下。

```
void LinkFileCopy(char *source, char *dest) //复制软链接
{
        //复制软链接
    char buffer[2 * KB];
    char oldpath[KB];
    getcwd(oldpath, sizeof(oldpath));
    strcat(oldpath, "/");
    memset(buffer, 0, sizeof(buffer));
    readlink(source, buffer, 2 * KB);//读取软链接到buffer
    symlink(buffer, dest);//将软链接赋给dest
}
```

2. Windows 系统

对于 Windows 系统而言，操作的逻辑类似，区别在于在该系统中，对文件的操作是通过 HANDLE 数据格式（句柄实现的），对文件复制的具体实现方法如下。

```
void CopyFile(char* source_file, char* dest_file) //复制文件
{
        WIN32_FIND_DATA lpfindfiledata;
        HANDLE hfindfile = FindFirstFile(source_file, &lpfindfiledata);
        HANDLE hsource = CreateFile(source_file, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
        HANDLE hdest_file = CreateFile(dest_file, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
        // 句柄获取

        LONG size = lpfindfiledata.nFileSizeLow - lpfindfiledata.nFileSizeHigh;
        int* buffer = new int[size];
        DWORD temp;
        bool tmp = ReadFile(hsource, buffer, size, &temp, NULL);
        WriteFile(hdest_file, buffer, size, &temp, NULL);
        // 复制文件时间

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

        SetFileAttributes(dest_file, GetFileAttributes(source_file));
        // 设置文件属性
```

```
        CloseHandle(hfindfile);
        CloseHandle(hsource);
        CloseHandle(hdest_file);
        // 关闭句柄
}
```

可见同样需要建立缓冲区数组，储存文件信息；区别在于对文件的操作需要使用句柄，同时需要声明的参数较多，如文件属性等。

对于遍历文件夹中所有文件的功能实现，Windows 也提供类类似 Linux 中的遍历函数 FindNextFile，通过给定当前文件夹地址的句柄以及当前文件的地址即可进行循环遍历，以下是部分代码：

```
        while (FindNextFile(hfindfile, &lpfindfiledata) != 0) //遍历
        {
                if (lpfindfiledata.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) //目录
                {
                        if (strcmp(lpfindfiledata.cFileName, ".") != 0 &&
strcmp(lpfindfiledata.cFileName, "..") != 0)
                        {
                                memset(dir_source, 0, sizeof(dir_source));
                                strcpy_s(dir_source, source_file);
                                strcat_s(dir_source, "\\");
                                strcat_s(dir_source,
lpfindfiledata.cFileName);
                                strcat_s(destination,
lpfindfiledata.cFileName);
                                CreateDirectory(destination, NULL);
                                CopyDir(dir_source, destination);
                                //复制文件时间
                                HANDLE handle_source = CreateFile(dir_source,
GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ, NULL, OPEN_EXISTING,
FILE_FLAG_BACKUP_SEMANTICS, NULL);
                                HANDLE handle_destination =
CreateFile(destination, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_FLAG_BACKUP_SEMANTICS, NULL);
                                FILETIME createtime, accesstime, writetime;
                                GetFileTime(handle_source, &createtime,
&accesstime, &writetime);
                                SetFileTime(handle_destination, &createtime,
&accesstime, &writetime);
```

```
                                SetFileAttributes(destination,
GetFileAttributes(dir_source));
                                strcpy_s(destination, dest_file);
                                strcat_s(destination, "\\");
                            }
                        }

                }
                CloseHandle(hfindfile);
```

五、实验结果和分析

1. Linux 系统

以下是需要复制的文件夹 target 的结构

```
(base) yuxiang@yuxiang-virtual-machine:~/BIT/OS/4$ cd target
(base) yuxiang@yuxiang-virtual-machine:~/BIT/OS/4/target$ ll
总计 16
drwxrwxr-x 3 yuxiang yuxiang 4096 12月  3 14:52 ./
drwxrwxr-x 4 yuxiang yuxiang 4096 12月  3 14:53 ../
-rw-rw-r-- 1 yuxiang yuxiang    7 12月  3 14:48 2.txt
drwxrwxr-x 2 yuxiang yuxiang 4096 12月  3 14:45 sub_target/
lrwxrwxrwx 1 yuxiang yuxiang   11 12月  3 14:52 sub_target_link -> sub_target//
```

在完成编译 mvcp.cpp 之后，输入命令

```
./mvcp target target_copy
```

对 target 文件夹以及其中的子文件夹、链接文件与文本文件进行复制

```
(base) yuxiang@yuxiang-virtual-machine:~/BIT/OS/4$ gcc mvcp.cpp -o mvcp
(base) yuxiang@yuxiang-virtual-machine:~/BIT/OS/4$ ./mvcp target target_copied
创建target_copied目录
Direction copying completed
```

然后在该路径下查看复制的文件夹 target_copied 的结构

```
(base) yuxiang@yuxiang-virtual-machine:~/BIT/OS/4$ cd target_copied
(base) yuxiang@yuxiang-virtual-machine:~/BIT/OS/4/target_copied$ ll
总计 16
drwxrwxr-x 3 yuxiang yuxiang 4096 12月  3 14:52 ./
drwxrwxr-x 4 yuxiang yuxiang 4096 12月  3 14:53 ../
-rw-rw-r-- 1 yuxiang yuxiang    7 12月  3 14:48 2.txt
drwxrwxr-x 2 yuxiang yuxiang 4096 12月  3 14:45 sub_target/
lrwxrwxrwx 1 yuxiang yuxiang   11 12月  3 14:52 sub_target_link -> sub_target//
```

与原文件夹结构一致。

2. Windows 系统

在 Windows 系统中，执行步骤与 Linux 系统类似，需要复制的文件夹结构如下：

```
    目录: C:\Users\syuxi\Desktop\OS\4\test


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         2024/12/3     15:49                sub_test
-a----         2024/12/3     15:45          95378 mvcp_windows.exe
```

在完成对 mvcp_windows.cpp 编译后，输入命令

```
mvcp_windows test test_copied
```

运行后查看 target_copied 文件夹结构如下：

```
    目录: C:\Users\syuxi\Desktop\OS\4\test_copy


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         2024/12/3     15:49                sub_test
-a----         2024/12/3     15:45          95378 mvcp_windows.exe
```

可见原文件夹中的应用程序文件以及子文件夹均复制成功

六、讨论、心得

此次实验让我学到了许多关于 Linux 与 Windows 系统关于文件访问与读取的 API，特别是对 Windows 系统句柄的调用。

不仅如此，我对操作系统中文件管理的底层原理和实际实现有了更深刻的理解。Linux 系统强调底层控制，API 简单而灵活，如`readlink`和`symlink`对软链接的处理让我深入认识了文件类型的差异；而 Windows 则通过句柄抽象和属性同步提供了更高层次的封装，虽然参数复杂但更适合快速开发。实验中，我解决了路径拼接、缓冲区管理、句柄资源释放等问题，锻炼了调试能力和代码优化技巧。同时，通过与同学讨论，我体会到团队协作的价值，并在编程风格上更加注重可读性和模块化设计。此次实验不仅让我掌握了跨平台文件操作的核心技术，还增强了我分析问题和解决问题的能力，为未来的学习和开发积累了宝贵经验。