操作系统课程设计实验报告

实验名称： 复制文件

姓名/学号： 宋尚儒/1120180717

一、 实验目的

设计并实现 windows 和 linux 下目录复制命令。学习使用 linux 和 windows 下有关目录读写和文件读写的 API 函数，对文件系统有初步的理解

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，在 linux 下运行结果如下：

```
~
❯ ./gitclones/osexp/mycp ~/test/ ~/sametest

~
❯ ls -Rla test/ sametest/
sametest/:
总用量 16
drwxr-xr-x  3 unv unv 4096 12月 16 09:01 ./
drwx------ 24 unv unv 4096 12月 18 16:02 ../
-rw-r--r--  1 unv unv   20 12月 16 09:00 example
drwxr-xr-x  2 unv unv 4096 12月 16 09:02 exp_folder/
lrwxrwxrwx  1 unv unv   35 12月 16 09:01 proxychains.conf -> /home/unv/dotfiles/proxychains.conf

sametest/exp_folder:
总用量 16
drwxr-xr-x 2 unv unv 4096 12月 16 09:02 ./
drwxr-xr-x 3 unv unv 4096 12月 16 09:01 ../
-rw-r--r-- 1 unv unv   35 12月 16 09:02 exp2
-rw-r--r-- 1 unv unv    9 12月 16 09:02 exp3

test/:
总用量 16
drwxr-xr-x  3 unv unv 4096 12月 16 09:01 ./
drwx------ 24 unv unv 4096 12月 18 16:02 ../
-rw-r--r--  1 unv unv   20 12月 16 09:00 example
drwxr-xr-x  2 unv unv 4096 12月 16 09:02 exp_folder/
lrwxrwxrwx  1 unv unv   35 12月 16 09:01 proxychains.conf -> /home/unv/dotfiles/proxychains.conf

test/exp_folder:
总用量 16
drwxr-xr-x 2 unv unv 4096 12月 16 09:02 ./
drwxr-xr-x 3 unv unv 4096 12月 16 09:01 ../
-rw-r--r-- 1 unv unv   35 12月 16 09:02 exp2
-rw-r--r-- 1 unv unv    9 12月 16 09:02 exp3
```

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

复制后，不仅读写权限一致，而且时间属性也一致。

三、 实验环境

Windows10

VMWare15.5

Ubuntu-20.04

四、 程序设计与实现

首先我们从示例中可以看到，这一命令除自身外还需包含两个参数，第一个是原目录，第二个是目标目录，应将源目录中所有文件复制到目标目录中，并修改文件各属性与源目录中文件一致。

值得注意的是，在复制目录的文件时，可能还需要复制子目录以及子目录中的文件，对子目录的复制操作和父目录完全可以使用相同的执行方法，且复制的目录结构呈树状，其中普通文件为子节点，目录文件为非子节点，因此目录复制这一操作很适合以 dfs 的形式进行递归操作。

通用的简要设计思路结构以伪代码形式表现如下，核心即为这三个函数：

```
void Copy_File(源文件,目标文件)
{
    打开源文件，读源文件
    创建目标文件，写目标文件
}
void Copy_Dir(源目录,目标目录)
{
    while(找到源目录下的子文件)
    {
        if(子文件是目录文件)
        {
            创建目标子目录文件
            Copy_Dir(源子目录,目标子目录);
        }
        else
            Copy_File(源子文件,目标子文件);
    }
}
```

```
int main(int argc,char *argv[])
{
    if(参数无误)
    {
        Copy_Dir(argv[1],argv[2]);
    }
    return 0;
}
```

这里没有说明在哪里修改文件属性，这是因为在本次实验的设计中不同系统下有不同的修改形式，但递归复制文件的操作思路大致相同，故修改文件属性的内容会在之后具体阐述。并且值得注意的是，linux 系统复制文件除了判断目录文件和普通文件外，还需要特判链接文件，并做特殊处理，这也会在之后 linux 具体设计中有详细说明。

1. 在 Windows 下实现：

(1) main 函数的结构相对简单，仅需要对参数做判断处理后调用 Copy_Dir 即可。在判断参数时，除了首先判断参数个数外，还需要判断源文件是否存在，目标目录是否存在，若参数数量错误或源文件目录不存在则结束程序，若目标目录不存在则创建目标目录并进行下一步操作，即调用函数 Copy_Dir 复制源目录文件到目标目录。

具体来说，判断源文件是否存在需要使用结构 WIN32_FIND_DATA 和函数 FindFirstFile，WIN32_FIND_DATA 结构保存了文件属性信息，具体使用参考如下

```
WIN32_FIND_DATA lffd;
if(FindFirstFile(argv[1],&lffd)==INVALID_HANDLE_VALUE) {}
```

目标文件目录判断方法同上，若需要创建目标文件夹，需要 CreateDirectory 函数，具体使用方法如下

```
CreateDirectory(argv[2],NULL);
```

然后即可调用自定义的 Copy_Dir 函数进行目录复制

(2) 函数 Copy_Dir 作为一个递归调用的函数是整个程序的核心，其基本架构已在之前说明，接下来具体说明一下各部分实现方法。

首先需要找到源目录下的子文件，这里首先需要使用函数 FindFirstFile 找到源目录下的第一个子文件，需要注意的是，这里函数的文件字符串参数必须在源目录名后加上”*.*”.然后使用循环调用函数 FindNextFile，不断找到源目录下的子文件，直到该函数返回值为 NULL，说明没有子文件了，具体实现参考如下

```
strcpy(ts,dir_sor);
strcat(ts,"\\*.*");
WIN32_FIND_DATA lffd;
HANDLE hfile=FindFirstFile(ts,&lffd);
while(FindNextFile(hfile,&lffd)){}
```

然后查看循环内部，此时如上代码所示结构体 lffd 即为当前指向的子文件的属性结构体，lffd.cFileName 即为子文件的名称，然后即可构造子文件的相对路径名，这几步操作较为简单就不做具体展示。

根据子文件属性判断下一步操作，这需要对 lffd.dwFileAttributes 做出判断，若是目录文件，则需要创建新目录，并递归调用函数 Copy_Dir，若为普通文件，则调用函数 Copy_File，结构参考如下

```
构造源子文件路径 ts
构造目标子文件路径 tt
if(lffd.dwFileAttributes==FILE_ATTRIBUTE_DIRECTORY)
{
    CreateDirectory(tt,NULL);
    Copy_Dir(ts,tt);
}
else
{
    Copy_File(ts,tt);
}
```

循环结束后，还需要修改目标目录的所有属性，在此修改属性的原因是因为这是一个递归调用的函数，若在别处修改属性可能因为对目录

内子文件的操作对目录本身时间等属性产生影响，因此需要放在最后便不需要更多复杂的操作，在本实验的设计中使用了自定义函数 Change_Attr 对目录和普通文件属性进行统一修改，在调用前需打开两个文件并获得句柄，其在函数 Copy_Dir 中调用方式具体如下

```
HANDLE hsor=CreateFile(dir_sor,GENERIC_READ|GENERIC_WRITE,FILE_SHARE_READ,
NULL,OPEN_ALWAYS,FILE_ATTRIBUTE_NORMAL|FILE_FLAG_BACKUP_SEMANTICS,NULL);
HANDLE htar=CreateFile(dir_tar,GENERIC_READ|GENERIC_WRITE,FILE_SHARE_READ,
NULL,OPEN_ALWAYS,FILE_ATTRIBUTE_NORMAL|FILE_FLAG_BACKUP_SEMANTICS,NULL);
Change_Attr(dir_sor,dir_tar,hsor,htar);
```

完成目录复制操作，在最后注意关闭各个句柄

(3) 函数 Copy_File 为具体的文件复制操作，输入参数为源文件和目标文件的路径，首先需要调用函数 CreateFile 打开源文件，创建目标文件，注意源文件必须以只读的方式打开，否则无法正确打开只读文件，参考使用如下，已标明关键参数

```
HANDLE hsor=CreateFile(file_sor,          //源文件路径
                       GENERIC_READ,      //只读访问
                       FILE_SHARE_READ,   //共享读
                       NULL,              //默认安全
                       OPEN_EXISTING,     //打开已存在的文件
                       FILE_ATTRIBUTE_NORMAL|FILE_FLAG_BACKUP_SEMANTICS,
                       NULL);

HANDLE htar=CreateFile(file_tar,          //源文件路径
                       GENERIC_READ|GENERIC_WRITE,  //读写访问
                       FILE_SHARE_READ,   //共享读
                       NULL,              //默认安全
                       CREATE_ALWAYS,     //创建新文件
                       FILE_ATTRIBUTE_NORMAL|FILE_FLAG_BACKUP_SEMANTICS,
                       NULL);
```

然后需要进行文件读写操作，具体来说首先需要一个缓冲区 buff，循环调用 ReadFile 函数将源文件的内容读取到 buff 中，并调用 WriteFile 将 buff 中的内容写入目标文件，具体实现可参考如下，需要注意每次读取和写入的数据长度

```
char buff[buff_size];
DWORD tp=0;  //记录每次读取的数据长度
while(ReadFile(hsor,buff,buff_size,&tp,NULL))
{
    WriteFile(htar,buff,tp,&tp,NULL);
    if(tp<buff_size)
        break;
}
```

如此即可完成文件的读写操作，然后还需要调用自定义函数 Change_Attr 修改目标文件的各项属性，由于之前已打开文件获得句柄，所以不需要再次打开，调用方式类似与之前，就不再次说明。

注意最后需要关闭两个文件的句柄。

(4) 因为目录和普通文件修改属性操作类似，所以自定义函数 Change_Attr 进行统一处理，该函数共有 4 个输入参数，为源文件和目标文件的路径和句柄。首先使用函数 GetFileAttributes 获取源文件属性，再通过函数 SetFileAttributes 赋值给目标文件，还需要修改文件的时间信息，这里需要用到函数 GetFileTime 和 SetFiletime，具体使用参考如下

```
void Change_Attr(char *sor,char *tar,HANDLE hsor,HANDLE htar)
{
    DWORD attr=GetFileAttributes(sor);
    SetFileAttributes(tar,attr);

    FILETIME t_cre,t_acc,t_wri;
    GetFileTime(hsor,&t_cre,&t_acc,&t_wri);
    SetFileTime(htar,&t_cre,&t_acc,&t_wri);
}
```

编译生成 mycp.exe，这里以自定义的源目录 source 为例，该目录中包含字符文件、只读图片文件、非空目录，复制为 target 目录，以下为执行结果展示

```
PS D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows> ./mycp source target
PS D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows> dir -r source


    目录: D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows\source


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2020/12/20     12:19                d1
-ar---         2020/10/1     21:00         137701 i1.jpg
-a----        2020/12/20     12:19             14 t1.txt


    目录: D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows\source\d1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2020/10/1     20:59         282343 i2.jpg


PS D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows> dir -r target


    目录: D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows\target


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2020/12/20     12:19                d1
-ar---         2020/10/1     21:00         137701 i1.jpg
-a----        2020/12/20     12:19             14 t1.txt


    目录: D:\资料\2020年秋\操作系统\实验\实验报告\07111803-1120180717-宋尚儒-实验四\windows\target\d1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2020/10/1     20:59         282343 i2.jpg
```

可以看到，目标目录下各文件复制完成，大小、时间、权限等属性与源目录下文件一致

2. 在 Linux 下实现：

(1) main 函数的结构相对简单，仅需要对参数做判断处理后调用 Copy_Dir 即可。在判断参数时，除了首先判断参数个数外，还需要判断源文件是否存在，目标目录是否存在，若参数数量错误或源文件目录不存在则结束程序，若目标目录不存在则创建目标目录并进行下一步操作，即调用函数 Copy_Dir 复制源目录文件到目标目录。

具体来说，判断源文件是否存在需要使用结构 stat 和函数 lstat 以及函数 opendir，stat 结构保存了文件属性信息，具体使用参考如下

```
struct stat statbuf;
lstat(argv[1],&statbuf);
if(S_ISDIR(statbuf.st_mode)){}
if(opendir(argv[1])==NULL) {}
```

目标文件目录判断方法同上，若需要创建目标文件夹，需要CreateDirectory 函数，具体使用方法如下，其中第二个参数为源目录的 stat 结构体的成员，描述了源目录的类型和权限

```
mkdir(argv[2],statbuf.st_mode);
```

然后即可调用自定义的 Copy_Dir 函数进行目录复制

(2) 函数 Copy_Dir 作为一个递归调用的函数是整个程序的核心，其基本架构已在之前说明，接下来具体说明一下各部分实现方法。

首先需要打开源目录，并循环调用函数 readdir 顺序访问所有目录项，到达目录尾部的时候该函数会返回 NULL，对于每一个目录项，可以使用 dirent 结构的结构体 entry_sor 获取当前目录项的名称，具体实现参考如下

```
DIR *pd_sor=opendir(dir_sor);
struct dirent *entry_sor=NULL;
while((entry_sor=readdir(pd_sor))){}
```

然后查看循环内部，此时可使用 entry_sor->d_name 获取当前目录项的名称，注意跳过当前目录和父目录，然后即可构造子文件的相对路径名，这几步操作较为简单就不做具体展示。

根据子文件属性判断下一步操作，首先需要使用结构体 stat 和函数 lstat 获取文件属性结构，进而获得文件分类，这需要调用 S_ISLNK、S_ISDIR、S_ISREG 三种宏对文件分类进行判断，若是目录文件，则需要创建新目录，并递归调用函数 Copy_Dir，若为普通文件，则调用函数 Copy_File，若是链接文件（这也是与 windows 类似函数中不同之处），则调用函数 Copy_Link，结构参考如下

```
构造源子文件路径 ts
构造目标子文件路径 tt
struct stat statbuf;
lstat(ts,&statbuf);
if(S_ISLNK(statbuf.st_mode))
{
    Copy_Link(ts,tt);
}
else if(S_ISDIR(statbuf.st_mode))
{
    mkdir(tt,statbuf.st_mode);
    Copy_Dir(ts,tt);
}
else if(S_ISREG(statbuf.st_mode))
{
    Copy_File(ts,tt);
}
```

循环结束后，还需要修改目标目录的所有属性，原因与 windows 中基本相同，在本实验的设计中使用了自定义函数 Change_Attr 对目录、普通文件和链接文件属性进行统一修改，只需要源文件和目标文件的路径即可调用，操作简单就不在此说明。

(3) 函数 Copy_File 为具体的文件复制操作，输入参数为源文件和目标文件的路径，首先需要调用函数 open 打开源文件，函数 creat 创建目标文件，参考使用如下

```
//打开源文件
int fd_sor=open(file_sor,O_RDONLY);
//创建目标文件
int fd_tar=creat(file_tar,O_WRONLY);
```

然后需要进行文件读写操作，具体来说首先需要一个缓冲区 buff，循环调用 read 函数将源文件的内容读取到 buff 中，read 函数返回值即为实际读取的数据长度，并调用函数 write 将 buff 中长度为 len 的内容写入目标文件，具体实现可参考如下

```
unsigned char buff[buffer_size];
int len;
while((len=read(fd_sor,buff,buffer_size))>0)
{
    write(fd_tar,buff,len);
}
```

如此即可完成文件的读写操作，然后还需要调用自定义函数 Change_Attr 修改目标文件的各项属性，注意最后需要通过两个文件的标识符关闭两个文件。

(4) 函数 Copy_Link 为软链接文件的复制操作，输入参数为源文件和目标文件的路径，首先需要调用函数 readlink 读取软链接内容，保存到字符串 path 中，再调用函数 symlink，使用 path 的内容和目标文件路径创建目标软链接文件并复制内容，具体使用参考如下

```
unsigned char path[buffer_size];
readlink(lnk_sor,path,buffer_size);
symlink(path,lnk_tar);
```

然后还需要调用自定义函数 Change_Attr 修改目标软链接文件的各项属性

(5) 因为目录、普通文件软链接修改属性操作类似，所以自定义函数 Change_Attr 进行统一处理，该函数共有 2 个输入参数，为源文件和目标文件的路径和句柄。首先使用结构体 stat 和函数 lstat 保存源文件属性信息结构，然后使用该结构作为参数调用函数 chmod 修改目标文件权限，调用函数 chown 修改目标文件所有者，具体使用参考如下

```
struct stat statbuf;
lstat(file_sor,&statbuf);
chmod(file_tar,statbuf.st_mode);
chown(file_tar,statbuf.st_uid,statbuf.st_gid);
```

因为链接文件时间修改方式不同于目录文件和普通文件，所以需要做类型判断，类似的判断已在之前做过说明，这里不再重复。对于软链

接文件，需要结构体 timeval 和函数 lutimes 修改访问时间和修改时间，具体使用参考如下

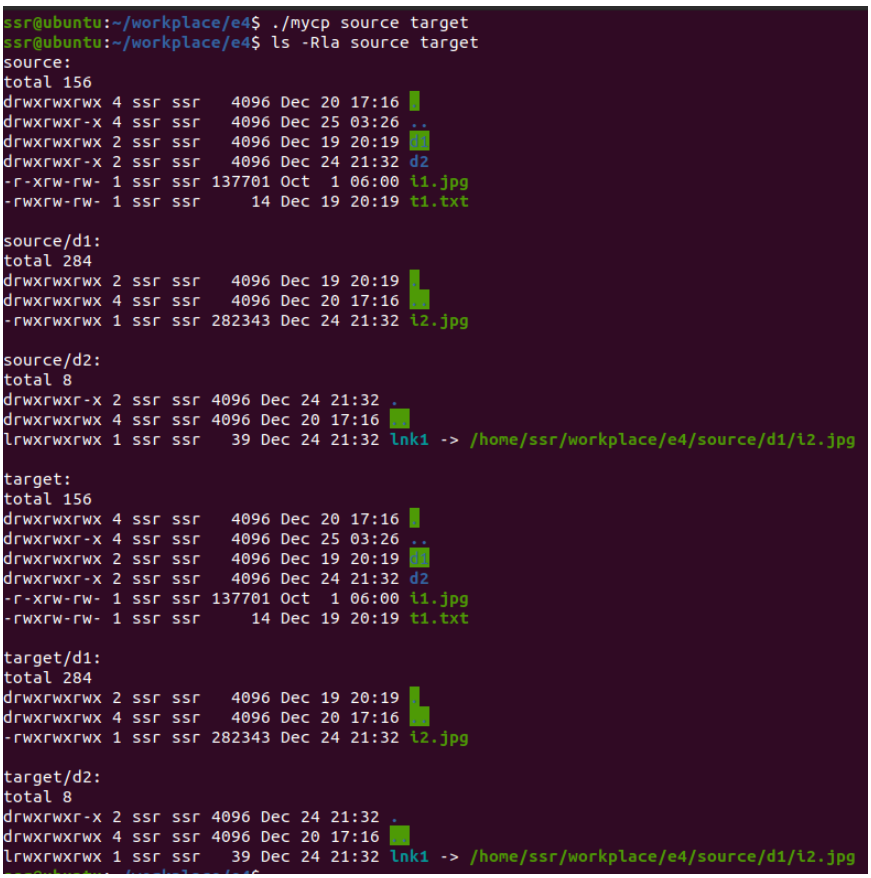
```
struct timeval time_sor[2];
time_sor[0].tv_sec=statbuf.st_mtime;
time_sor[1].tv_sec=statbuf.st_ctime;
lutimes(file_tar,time_sor);
```

对于普通文件和目录文件，需要使用结构体 utimbuf 和函数 utime 修改文件的访问时间和修改时间，具体使用参考如下

```
struct utimbuf utime_sor;
utime_sor.actime=statbuf.st_atime;
utime_sor.modtime=statbuf.st_mtime;
utime(file_tar,&utime_sor);
```

完成属性修改操作

编译生成 mycp.exe，这里以自定义的源目录 source 为例，该目录中包含字符文件、只读图片文件、非空目录以及一个子目录下的软链接文件，复制为 target 目录，以下为执行结果展示



可以看到，目标目录下各文件包括软链接文件复制完成，大小、时间、权限等属性与源目录下文件一致

五、 实验收获与体会

总体来说 api 还是 windows 更复杂一些，但考虑到这次 linux 还需要对软链接做特殊处理，linux 的命令实现可能稍显更麻烦，不得不说 windows 函数的各类参数处理还是不大方便，创建文件对象函数的打开方式和权限等参数给我造成很大困扰，并且由于对文件读写函数不熟悉，一开始使用一次性读取整个文件的形式，在大文件复制上可能产生问题，后来仔细研究了函数调用方式才改成缓冲区读写的方式，不如 linux 直观。Linux 特殊的软链接文件也给我造成一些麻烦，比如修改文件时间函数，一开始是没有特判文件类型直接顺序调用两个函数居然可以正常修改时间，上网查找具体调用方法，但关于 lutimes 函数的使用实在不多，最后简要查看了函数源码，修改成特判文件类型的形式。总体来说对各类文件和目录的 api 使用有了初步了解，可以用在之后的学习中。