实验四 复制文件

班级： 07112004 学号： 1120201883 姓名： 刘博文

一、实验目的

编程实现一个复制命令，以加深对课程中文件系统的认识，掌握linux系统中关于文件操作的部分API

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 [~/]# 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 [~/]# mycp sem target

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：

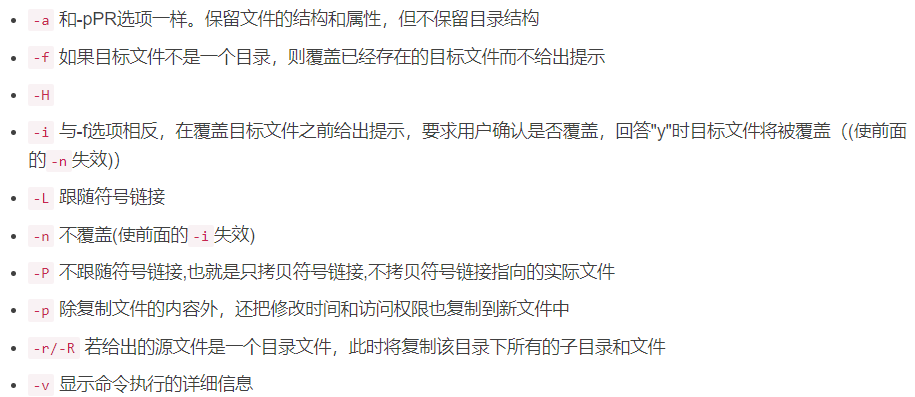
在Linux平台上做。

Linux： mkdir,opendir,readdir,symlink,readlink等系统调用

三、程序设计与实现

这个实验相当于是要求我们自己完成一个cp命令，那么要自己完成一个"mycp"命令，就应该先看看linux自带的cp命令有什么样的功能，cp命令语法：

cp [-options] source\_file dest\_file

cp命令参数：

可以看到cp命令有着非常多的参数，逐个实现有些过于费劲而且其中很多重复工作感觉也是没有必要的，所以我根据我的使用体验也就是我平时使用cp命令附带的最多的参数来调整我自己的mycp命令：默认带有-a, -f这两个个参数，也就是会**支持复制目录**，并在在复制目录时**保留链接、文件属性，默认覆盖重复文件**。

要用到的函数与结构体：

**结构体DIR**

struct \_\_dirstream     
{     
   void \*\_\_fd;      
   char \*\_\_data;      
   int \_\_entry\_data;      
   char \*\_\_ptr;      
   int \_\_entry\_ptr;      
   size\_t \_\_allocation;      
   size\_t \_\_size;      
\_\_libc\_lock\_define (, \_\_lock)      
};     
typedef struct \_\_dirstream DIR;

结构体是目录的抽象结构体定义

**结构体dirent**

struct dirent     
{     
   //索引节点号  
   long d\_ino;  
　　 //在目录文件中的偏移  
   off\_t d\_off;   
   //文件名长  
   unsigned short d\_reclen;  
   //文件类型  
   unsigned char d\_type;  
   //文件名，最长255字符  
   char d\_name [NAME\_MAX+1];   
}

**结构体stat**

struct stat {     
       mode\_t     st\_mode;       //文件访问权限     
       ino\_t      st\_ino;       //索引节点号     
       dev\_t      st\_dev;        //文件使用的设备号     
       dev\_t      st\_rdev;       //设备文件的设备号     
       nlink\_t    st\_nlink;      //文件的硬连接数

//所有者用户识别号  
       uid\_t      st\_uid;             
 //组识别号

gid\_t      st\_gid;             
       off\_t      st\_size;

//最后一次访问该文件的时间  
       time\_t     st\_atime;

//最后一次修改该文件的时间    
       time\_t     st\_mtime;

//最后一次改变该文件状态的时间  
       time\_t     st\_ctime;           
       blksize\_t st\_blksize;  
       blkcnt\_t   st\_blocks;   
};

**函数stat**

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

获取参数path指向文件或目录的信息，并将其保存在参数buf中。成功返回0，失败返回-1

**函数opendir**

DIR \* opendir(const char \* addr);

打开路径为addr的一个文件，成功则返回该文件的DIR，不成功则返回NULL

**函数readdir**

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

返回参数dir目录流的下个目录进入点，若有错误发生或读取到目录文件尾则返回NULL

**函数mkdir**

int mkdir(const char \*pathname, mode\_t mode);

以参数mode方式创建一个以参数pathname命名的目录，mode定义新创建目录的权限

返回0表示成功，返回-1表示错误

**函数open**

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

以参数flag(O\_RDONLY, O\_WRONLY, O\_RDWR)方式打开参数pathname指向的文件，返回文件描述符

**函数readlink**

ssize\_t readlink(const char \*path, char \*buf, size\_t bufsiz);

将参数path的符号连接内容到参数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

**函数symlink**

int symlink(const char \* oldpath, const char \* newpath);

以参数newpath指定的名称来建立一个新的连接(符号连接)到参数oldpath所指定的已存在文件。参数oldpath指定的文件不一定要存在, 如果参数newpath指定的名称为一已存在的文件则不会建立连接

**函数lstat**

int lstat(const char \*pathname, struct stat \*buf);

对于普通文件，函数lstat与函数stat相同；对于链接文件，lstat函数会获取链接文件本身的属性信息而stat函数会获取链接文件指向的文件的属性信息

**函数chmod**

int chmod(const char \*path, mode\_t mode);

以参数mode的权限来更新参数path指定文件的权限

**函数chown**

int chown(const char \*path, uid\_t owner, gid\_t group);

将参数path指定文件的所有用户与组改成参数owner与参数group

**结构体timaval**

struct timeval {  
   //秒  
long tv\_sec;  
   //毫秒  
long tv\_usec;  
};

**函数lutimes**

int lutimes(const char \*filename, const struct timeval tv[2]);

用于修改软连接文件的属性，修改参数filename指定的文件的时间，参数tv[0]是访问时间，tv[1]是修改时间

**结构体utimbuf**

struct utimbuf {  
   //最后访问时间  
time\_t actime;  
   //最后修改时间  
time\_t modtime;  
};

**函数utime**

int utime(const char \*path, const struct utimbuf \*times);

将参数path指定文件的访问和修改时间修改为参数times所指定的内容

之后分析一下应该怎样设计这个程序，在正式开始写程序之前可以大概料想到因为文件系统呈现给用户的是一个树形的结构，那我们要写一个递归的函数来去对目录与文件进行复制。

我们的程序要做的事大概是以下这些：

1. 检测输入的参数个数和是否正确，若正确则调用copy函数开始复制
2. 遍历源目录下所有的目录与文件，并根据种类进行不同的处理：
   * 目录文件：递归调用copy函数复制该目录及该目录下的所有文件
   * 软链接文件：调用copyLink函数复制该链接文件
   * 普通文件：调用copyFile函数复制该文件
3. 在复制时还要保持文件的一些属性不变，比如权限、所属用户与组、最后访问时间与修改时间等

那么也就是我们要实现一下几个函数：



四、实验结果及分析

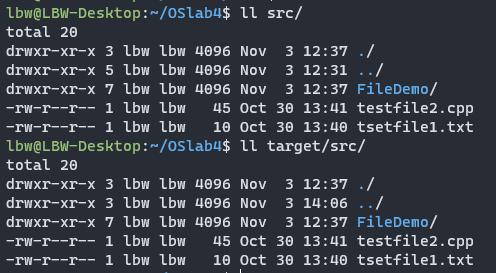
实验环境：WSL Ubuntu20.04LTS

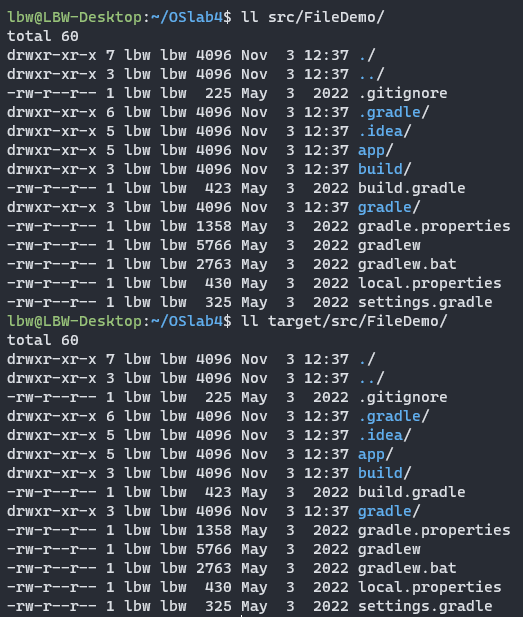
实验源码：lab4.c，代码较长所以不在此贴出

编译源文件：



复制文件：

查看复制后的目录信息：

继续查看目录FileDemo下的信息：

可以看到子目录下的文件与目录的权限、所属用户与组、修改时间等仍然与被复制目录一模一样，继续查看子目录也仍是正确结果，至此可以证明程序的正确性

五、实验收获与体会（黑体四号字）

Linux系统相关API的调用真的比windows上要简单好多，这个实验相比前面的实验没有费太多力气在API的学习上。但是也有一些小坑，比如linux中的./目录和../目录，一开始写的时候没有注意避开这俩，导致最后写出的程序根本没法正确的复制，不过好在这个问题还算比较好解决。