

实验十一

Hollow Man

实验名称:

文件系统编程

实验目的:

1. 学习和掌握使用系统调用操作文件和目录的属性、内容的方法

实验时间

3 学时

预备知识:

1. 系统调用

- 1.1 文件操作

- 1.1.1 open (打开文件)

表头文件 `#include<sys/types.h>`

`#include<sys/stat.h>`

`#include<fcntl.h>`

定义函数 `int open(const char * pathname, int flags);`

`int open(const char * pathname,int flags, mode_t mode);`

函数说明 参数 `pathname` 指向欲打开的文件路径字符串。下列是参数 `flags` 所能使用的标志:

`O_RDONLY` 以只读方式打开文件

`O_WRONLY` 以只写方式打开文件

`O_RDWR` 以可读写方式打开文件。

上述三种旗标是互斥的，也就是不可同时使用，但可与下列的旗标利用 `OR(|)` 运算符组合。

`O_CREAT` 若欲打开的文件不存在则自动建立该文件。

`O_EXCL` 如果 `O_CREAT` 也被设置，此指令会去检查文件是否存在。文件若不存在则建立该文件，否则将导致打开文件错误。此外，若 `O_CREAT` 与 `O_EXCL` 同时设置，并且欲打开的文件为符号连接，则会打开文件失败。

`O_NOCTTY` 如果欲打开的文件为终端机设备时，则不会将该终端机当成进程控制终端

机。

O_TRUNC 若文件存在并且以可写的方式打开时，此旗标会令文件长度清为 0，而原来存于该文件的资料也会消失。

O_APPEND 当读写文件时会从文件尾开始移动，也就是所写入的数据会以附加的方式加入到文件后面。

O_NONBLOCK 以不可阻断的方式打开文件，也就是无论有无数据读取或等待，都会立即返回进程之中。

O_NDELAY 同 O_NONBLOCK。

O_SYNC 以同步的方式打开文件。

O_NOFOLLOW 如果参数 pathname 所指的文件为一符号连接，则会令打开文件失败。

O_DIRECTORY 如果参数 pathname 所指的文件并非为一目录，则会令打开文件失败。

参数 mode 则有下列数种组合，只有在建立新文件时才会生效，此外真正建文件时的权限会受到 umask 值所影响，因此该文件权限应该为 (mode-umaks)。

S_IRWXU00700 权限，代表该文件所有者具有可读、可写及可执行的权限。

S_IRUSR 或 S_IREAD, 00400 权限，代表该文件所有者具有可读取的权限。

S_IWUSR 或 S_IWRITE, 00200 权限，代表该文件所有者具有可写入的权限。

S_IXUSR 或 S_IEXEC, 00100 权限，代表该文件所有者具有可执行的权限。

S_IRWXG 00070 权限，代表该文件用户组具有可读、可写及可执行的权限。

S_IRGRP 00040 权限，代表该文件用户组具有可读的权限。

S_IWGRP 00020 权限，代表该文件用户组具有可写入的权限。

S_IXGRP 00010 权限，代表该文件用户组具有可执行的权限。

S_IRWXO 00007 权限，代表其他用户具有可读、可写及可执行的权限。

S_IROTH 00004 权限，代表其他用户具有可读的权限

S_IWOTH 00002 权限，代表其他用户具有可写入的权限。

S_IXOTH 00001 权限，代表其他用户具有可执行的权限。

返回值 若所有欲核查的权限都通过了检查则返回 0 值，表示成功，只要有一个权限被禁止则返回-1。

错误代码 EEXIST 参数 pathname 所指的文件已存在，却使用了 O_CREAT 和 O_EXCL 旗标。

EACCESS 参数 pathname 所指的文件不符合所要求测试的权限。

EROFS 欲测试写入权限的文件存在于只读文件系统内。

EFAULT 参数 pathname 指针超出可存取内存空间。

EINVAL 参数 mode 不正确。

ENAMETOOLONG 参数 pathname 太长。

ENOTDIR 参数 pathname 不是目录。

ENOMEM 核心内存不足。

ELOOP 参数 pathname 有过多符号连接问题。

EIO I/O 存取错误。

1.1.2 close(见管道通信)

1.1.3 read(见管道通信)

1.1.4 write(见管道通信)

1.1.5 lseek (移动文件的读写位置)

表头文件 `#include<sys/types.h>`

`#include<unistd.h>`

定义函数 `off_t lseek(int fildes, off_t offset ,int whence);`

函数说明 每一个已打开的文件都有一个读写位置，当打开文件时通常其读写位置是指向文件开头，若是以附加的方式打开文件(如 `O_APPEND`)，则读写位置会指向文件尾。当 `read()` 或 `write()` 时，读写位置会随之增加，`lseek()` 便是用来控制该文件的读写位置。参数 `fildes` 为已打开的文件描述词，参数 `offset` 为根据参数 `whence` 来移动读写位置的位移数。

参数 `whence` 为下列其中一种：

`SEEK_SET` 参数 `offset` 即为新的读写位置。

`SEEK_CUR` 以目前的读写位置往后增加 `offset` 个位移量。

`SEEK_END` 将读写位置指向文件尾后再增加 `offset` 个位移量。

当 `whence` 值为 `SEEK_CUR` 或 `SEEK_END` 时，参数 `offset` 允许负值的出现。

下列是较特别的使用方式：

1) 欲将读写位置移到文件开头时：`lseek (int fildes,0,SEEK_SET);`

2) 欲将读写位置移到文件尾时：`lseek (int fildes, 0,SEEK_END);`

3) 想要取得目前文件位置时：`lseek (int fildes, 0,SEEK_CUR);`

返回值 当调用成功时则返回目前的读写位置，也就是距离文件开头多少个字节。若有错误则返回-1，`errno` 会存放错误代码。

附加说明 Linux 系统不允许 `lseek ()` 对 `tty` 装置作用，此项动作会令 `lseek ()` 返回 `ESPIPE`。

1.1.6 fstat (由文件描述词取得文件状态)

表头文件 `#include<sys/stat.h>`

`#include<unistd.h>`

定义函数 `int fstat(int fildes, struct stat *buf);`

函数说明 `fstat()` 用来将参数 `fildes` 所指的文件状态，复制到参数 `buf` 所指的结构中 (`struct stat`)。 `fstat()` 与 `stat()` 作用完全相同，不同处在于传入的参数为已打开的文件描述词。

返回值 执行成功则返回 0，失败返回-1，错误代码存于 `errno`。

`struct stat;`

```
struct stat {
    dev_t      st_dev;      /* device */
    ino_t      st_ino;      /* inode */
    mode_t     st_mode;     /* protection */
    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 type (if inode device) */
    off_t      st_size;     /* total size, in bytes */
    unsigned long st_blksize; /* blocksize for filesystem I/O */
    unsigned long st_blocks; /* number of blocks allocated */
    time_t     st_atime;    /* time of last access */
    time_t     st_mtime;    /* time of last modification */
    time_t     st_ctime;    /* time of last change */
};
```

列数种情况

`S_IFMT 0170000` 文件类型的位掩码

`S_IFSOCK 0140000` socket

`S_IFLNK 0120000` 符号连接

`S_IFREG 0100000` 一般文件

`S_IFBLK 0060000` 区块装置

`S_IFDIR 0040000` 目录

`S_IFCHR 0020000` 字符装置

`S_IFIFO 0010000` 先进先出

`S_ISUID 04000` 文件的 (set user-id on execution) 位

`S_ISGID 02000` 文件的 (set group-id on execution) 位

`S_ISVTX 01000` 文件的 sticky 位

`S_IRUSR (S_IREAD) 00400` 文件所有者具可读取权限

`S_IWUSR (S_IWRITE) 00200` 文件所有者具可写入权限

S_IXUSR (S_IXEXEC) 00100 文件所有者具可执行权限

S_IRGRP 00040 用户组具可读取权限

S_IWGRP 00020 用户组具可写入权限

S_IXGRP 00010 用户组具可执行权限

S_IROTH 00004 其他用户具可读取权限

S_IWOTH 00002 其他用户具可写入权限

S_IXOTH 00001 其他用户具可执行权限

上述的文件类型在 POSIX 中定义了检查这些类型的宏定义

S_ISLNK (st_mode) 判断是否为符号连接

S_ISREG (st_mode) 是否为一般文件

S_ISDIR (st_mode) 是否为目录

S_ISCHR (st_mode) 是否为字符装置文件

S_ISBLK (s3e) 是否为先进先出

S_ISSOCK (st_mode) 是否为 socket

范例

```
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
main()
{
    struct stat buf;
    int fd;
    fd = open ( "/etc/passwd" , O_RDONLY );
    fstat(fd,&buf);
    printf( "/etc/passwd file size +%d\n " ,buf.st_size);
}
```

1.2 文件权限操作

1.2.1 chown (改变文件的所有者)

表头文件 #include<sys/types.h>

#include<unistd.h>

定义函数 int chown(const char * path, uid_t owner, gid_t group);

函数说明 chown()会将参数 path 指定文件的所有者变更为参数 owner 代表的用户，而将

该文件的组变更为参数 group 组。如果参数 owner 或 group 为-1，对应的所有者或组不会有所改变。root 与文件所有者皆可改变文件组，但所有者必须是参数 group 组的成员。当 root 用 chown() 改变文件所有者或组时，该文件若具有 S_ISUID 或 S_ISGID 权限，则会清除此权限位，此外如果具有 S_ISGID 权限但不具 S_IXGRP 位，则该文件会被强制锁定，文件模式会保留。

返回值 成功则返回 0，失败返回-1，错误原因存于 errno。

错误代码 参考 chmod（）。

范例 /* 将/etc/passwd 的所有者和组都设为 root */

```
#include<sys/types.h>
#include<unistd.h>
main()
{
    chown(“/etc/passwd”,0,0);
}
```

1.2.2 chmod（改变文件的权限）

表头文件 #include<sys/types.h>

#include<sys/stat.h>

定义函数 int chmod(const char * path,mode_t mode);

函数说明 chmod()会依参数 mode 权限来更改参数 path 指定文件的权限。

参数 mode 有下列数种组合

S_ISUID 04000 文件的（set user-id on execution）位

S_ISGID 02000 文件的（set group-id on execution）位

S_ISVTX 01000 文件的 sticky 位

S_IRUSR（S_IREAD） 00400 文件所有者具可读取权限

S_IWUSR（S_IWRITE） 00200 文件所有者具可写入权限

S_IXUSR（S_IEXEC） 00100 文件所有者具可执行权限

S_IRGRP 00040 用户组具可读取权限

S_IWGRP 00020 用户组具可写入权限

S_IXGRP 00010 用户组具可执行权限

S_IROTH 00004 其他用户具可读取权限

S_IWOTH 00002 其他用户具可写入权限

S_IXOTH 00001 其他用户具可执行权限

只有该文件的所有者或有效用户识别码为 0，才可以修改该文件权限。基于系统安全，如果欲将数据写入一执行文件，而该执行文件具有 S_ISUID 或 S_ISGID 权限，则这两个位会被清除。如果一目录具有 S_ISUID 位权限，表示在此目录下只有该文件的所有者或 root 可以删除该文件。

返回值 权限改变成功返回 0，失败返回-1，错误原因存于 errno。

错误代码 EPERM 进程的有效用户识别码与欲修改权限的文件拥有者不同，而且也不具 root 权限。

EACCESS 参数 path 所指定的文件无法存取。

EROFS 欲写入权限的文件存在于只读文件系统内。

EFAULT 参数 path 指针超出可存取内存空间。

EINVAL 参数 mode 不正确

ENAMETOOLONG 参数 path 太长

ENOENT 指定的文件不存在

ENOTDIR 参数 path 路径并非一目录

ENOMEM 核心内存不足

ELOOP 参数 path 有过多符号连接问题。

EIO I/O 存取错误

范例 /* 将/etc/passwd 文件权限设成 S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH */

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

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

```
main()
```

```
{
```

```
    chmod(“/etc/passwd”, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
```

```
}
```

1.3 目录操作

1.3.1 getcwd（取得当前的工作目录）

表头文件 #include<unistd.h>

定义函数 char * getcwd(char * buf, size_t size);

函数说明 getcwd() 会将当前的工作目录绝对路径复制到参数 buf 所指的内存空间，参数 size 为 buf 的空间大小。在调用此函数时，buf 所指的内存空间要足够大，若工作目录绝对路径的字符串长度超过参数 size 大小，则返回值 NULL，errno 的值则为 ERANGE。倘若参

数 buf 为 NULL，getcwd() 会依参数 size 的大小自动配置内存(使用 malloc())，如果参数 size 也为 0，则 getcwd() 会依工作目录绝对路径的字符串程度来决定所配置的内存大小，进程可以在使用完此字符串后利用 free() 来释放此空间。

返回值 执行成功则将结果复制到参数 buf 所指的内存空间，或是返回自动配置的字符串指针。失败返回 NULL，错误代码存于 errno。

范例 #include<unistd.h>

```
main()
{
    char buf[80];
    getcwd(buf, sizeof(buf));
    printf("current working directory : %s\n", buf);
}
```

1.3.2 chdir (改变当前的工作目录)

表头文件 #include<unistd.h>

定义函数 int chdir(const char * path);

函数说明 chdir() 用来将当前的工作目录改变成以参数 path 所指的目录。

返回值 执行成功则返回 0，失败返回-1，errno 为错误代码。

范例 #include<unistd.h>

```
main()
{
    chdir("/tmp");
    printf("current working directory: %s\n", getcwd(NULL, NULL));
}
```

1.3.3 opendir (打开目录)

表头文件 #include<sys/types.h>

#include<dirent.h>

定义函数 DIR * opendir(const char * name);

函数说明 opendir() 用来打开参数 name 指定的目录，并返回 DIR*形态的目录流，和 open() 类似，接下来对目录的读取和搜索都要使用此返回值。

返回值 成功则返回 DIR* 形态的目录流，打开失败则返回 NULL。

错误代码 EACCESS 权限不足

EMFILE 已达到进程可同时打开的文件数上限。

ENFILE 已达到系统可同时打开的文件数上限。

ENOTDIR 参数 name 非真正的目录

ENOENT 参数 name 指定的目录不存在，或是参数 name 为一空字符串。

ENOMEM 核心内存不足。

1.3.4 readdir (读取目录)

表头文件 `#include<sys/types.h>`

`#include<dirent.h>`

定义函数 `struct dirent * readdir(DIR * dir);`

函数说明 `readdir()` 返回参数 `dir` 目录流的下个目录进入点。

结构 `dirent` 定义如下

```
struct dirent
{
    ino_t d_ino;                /*此目录进入点的 inode */
    off_t d_off;                /*目录文件开头至此目录进入点的位移
*/
    signed short int d_reclen;    /*文件名长度*/
    char d_name[NAME_MAX+1];     /*文件名*/
};
```

返回值 成功则返回下个目录进入点。有错误发生或读取到目录文件尾则返回 `NULL`。

附加说明 `EBADF` 参数 `dir` 为无效的目录流。

范例 `#include<sys/types.h>`

`#include<dirent.h>`

`#include<unistd.h>`

`main()`

```
{
    DIR * dir;
    struct dirent * ptr;
    int i;
    dir = opendir( "/etc/rc.d" );
    while((ptr = readdir(dir))!=NULL)
    {
        printf( "d_name: %s\n", ptr->d_name);
    }
}
```

```

    }

    closedir(dir);
}

```

1.3.5 closedir (关闭目录)

表头文件 `#include<sys/types.h>`

`#include<dirent.h>`

定义函数 `int closedir(DIR *dir);`

函数说明 `closedir()` 关闭参数 `dir` 所指的目录流。

返回值 关闭成功则返回 0，失败返回-1，错误原因存于 `errno` 中。

错误代码 `EBADF` 参数 `dir` 为无效的目录流

1.3.6 mkdir(创建目录)

表头文件 `#include<fcntl.h>`

`#include<unistd.h>`

`#include<sys/stat.h>`

`#include<sys/types.h>`

定义函数 `int mkdir (const char *pathname, mode_t mode);`

函数说明 以 `mode(mode&~umask)` 为权限建立 `pathname` 指定的目录。

返回值 创建成功则返回 0，失败返回-1，错误原因存于 `errno` 中。

1.3.7 rmdir(删除目录)

表头文件 `#include<fcntl.h>`

`#include<unistd.h>`

`#include<sys/stat.h>`

`#include<sys/types.h>`

定义函数 `int rmdir (const char *pathname);`

函数说明 删除 `pathname` 指定的目录

返回值 成功则返回 0，失败返回-1，错误原因存于 `errno` 中。

2. 标准库函数 (略)

实验要求:

1. 编写一个程序，实现将一个目录的所有内容复制到另一个目录的功能。要求：
源文件（目录）和目标文件（目录）的属主、权限等信息保持一致；
每复制一个文件（目录），在屏幕提示相应信息；

当遇到符号链接文件时，显示该文件为链接文件，不复制。

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
char paths[1000], patht[1000], temp_paths[1000], temp_patht[1000];
void Copy(char *spathname, char *tpathname)
{
    int sfd, tfd;
    struct stat s, t;
    char c;
    sfd = open(spathname, O_RDONLY);
    tfd = open(tpathname, O_RDWR | O_CREAT);
    while (read(sfd, &c, 1) > 0)
        write(tfd, &c, 1);
    fstat(sfd, &s);
    chown(tpathname, s.st_uid, s.st_gid);
    chmod(tpathname, s.st_mode);
    close(sfd);
    close(tfd);
}
void d_copy(char *spathname, char *tpathname)
{
    struct stat s, t, temp_s, temp_t;
    struct dirent *s_p;
    DIR *dirs, *dirt;
    stat(spathname, &s);
    mkdir(tpathname, s.st_mode);
    chown(tpathname, s.st_uid, s.st_gid);
```

```

dirt = opendir(tpathname);
dirs = opendir(spathname);
strcpy(temp_paths, spathname);
strcpy(temp_patht, tpathname);
while ((s_p = readdir(dirs)) != NULL)
{
    if (strcmp(s_p->d_name, ".") != 0 && strcmp(s_p->d_name, "..") != 0)
    {
        strcat(temp_paths, "/");
        strcat(temp_paths, s_p->d_name);
        strcat(temp_patht, "/");
        strcat(temp_patht, s_p->d_name);
        lstat(temp_paths, &s);
        temp_s.st_mode = s.st_mode;
        if (S_ISLNK(temp_s.st_mode))
        {
            printf("%s is a symbol link file\n", temp_paths);
        }
        else if (S_ISREG(temp_s.st_mode))
        {
            printf("Copy file %s ..... \n", temp_paths);
            Copy(temp_paths, temp_patht);
            strcpy(temp_paths, spathname);
            strcpy(temp_patht, tpathname);
        }
        else if (S_ISDIR(temp_s.st_mode))
        {
            printf("Copy directory %s ..... \n", temp_paths);
            d_copy(temp_paths, temp_patht);
            strcpy(temp_paths, spathname);
            strcpy(temp_patht, tpathname);
        }
    }
}

```

```

        }
    }
}

int main()
{
    struct dirent *sp, *tp;
    char spath[1000], tpath[1000], temp_spath[1000], temp_tpath[1000];
    struct stat sbuf, tbuf, temp_sbuf, temp_tbuf;
    char sdirect[1000], tdirect[1000], judge;
    DIR *dir_s, *dir_t;
    printf("Please input the source direct path and name :");
    scanf("%s", sdirect);
    dir_s = opendir(sdirect);
    if (dir_s == NULL)
    {
        printf("This directory don't exist !\n");
        return 0;
    }
    if (stat(sdirect, &sbuf) != 0)
    {
        printf("Get status error !\n");
        return 0;
    }
    printf("Please input the target direct path and name :");
    scanf("%s", tdirect);
    dir_t = opendir(tdirect);
    if (dir_t == NULL)
    {
        mkdir(tdirect, sbuf.st_mode);
        chown(tdirect, sbuf.st_uid, sbuf.st_gid);
        dir_t = opendir(tdirect);
    }
}

```

```

else
{
    chmod(tdirect, sbuf.st_mode);
    chown(tdirect, sbuf.st_uid, sbuf.st_gid);
}
strcpy(spath, sdirect);
strcpy(tpath, tdirect);
strcpy(temp_spath, sdirect);
strcpy(temp_tpath, tdirect);
printf("Begin copy ..... \n");
while ((sp = readdir(dir_s)) != NULL)
{
    if (strcmp(sp->d_name, ".") != 0 && strcmp(sp->d_name, "..") != 0)
    {
        strcat(temp_spath, "/");
        strcat(temp_spath, sp->d_name);
        strcat(temp_tpath, "/");
        strcat(temp_tpath, sp->d_name);
        lstat(temp_spath, &sbuf);
        temp_sbuf.st_mode = sbuf.st_mode;
        if (S_ISLNK(temp_sbuf.st_mode))
        {
            printf("%s is a symbolic link file\n", temp_spath);
        }
        else if ((S_IFMT & temp_sbuf.st_mode) == S_IFREG)
        {
            printf("Copy file %s ..... \n", temp_spath);
            Copy(temp_spath, temp_tpath);
            strcpy(temp_tpath, tpath);
            strcpy(temp_spath, spath);
        }
        else if ((S_IFMT & temp_sbuf.st_mode) == S_IFDIR)

```

```

        {
            printf("Copy directory %s ..... \n", temp_spath);
            d_copy(temp_spath, temp_tpath);
            strcpy(temp_tpath, tpath);
            strcpy(temp_spath, spath);
        }
    }

    printf("Copy end !\n");
    closedir(dir_t);
    closedir(dir_s);
    return 1;
}

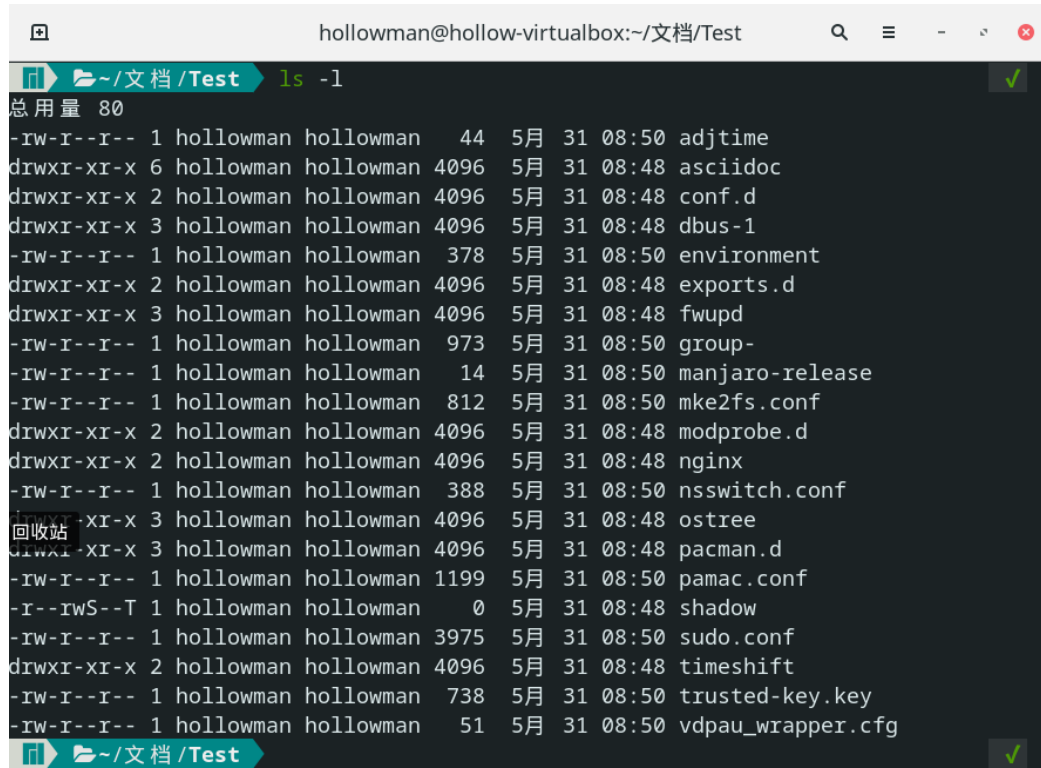
```

```

hollowman@hollow-virtualbox:~/文档
gcc 1.c
./a.out
Please input the source direct path and name :/etc
Please input the target direct path and name :./Test
Begin copy .....
Copy directory /etc/dbus-1 .....
Copy directory /etc/dbus-1/system.d .....
Copy file /etc/dbus-1/system.d/org.freedesktop.NetworkManager.Manjaro.conf .....
Copy file /etc/dbus-1/system.d/org.freedesktop.NetworkManager.Manjaro.conf/org.m
anjaropamac.daemon.conf .....
Copy directory /etc/ostree .....
Copy directory /etc/ostree/remotes.d .....
Copy file /etc/pamac.conf .....
Copy directory /etc/exports.d .....
Copy directory /etc/modprobe.d .....
Copy directory /etc/pacman.d .....
Copy file /etc/pacman.d/mirrorlist .....
Copy directory /etc/pacman.d/gnupg .....
Copy file /etc/pacman.d/gnupg/gpg-v21-migrated .....
Copy file /etc/pacman.d/gnupg/gpg-v21-migrated/gpg-agent.conf .....
Copy file /etc/pacman.d/gnupg/gpg-v21-migrated/gpg-agent.conf/tofu.db .....
Copy file /etc/pacman.d/gnupg/gpg-v21-migrated/gpg-agent.conf/tofu.db/private-k
eys-v1.d .....

```

Test 文件夹中的内容:



The image shows a terminal window titled "hollowman@hollow-virtualbox:~/文档/Test". The terminal displays the output of the command "ls -l". The output lists various files and directories with their permissions, owner, group, size, date, and name. The files include "adjtime", "asciidoc", "conf.d", "dbus-1", "environment", "exports.d", "fwupd", "group-", "manjaro-release", "mke2fs.conf", "modprobe.d", "nginx", "nsswitch.conf", "ostree", "pacman.d", "pamac.conf", "shadow", "sudo.conf", "timeshift", "trusted-key.key", and "vdpau_wrapper.cfg". The terminal also shows a "回收站" (Recycle Bin) icon and a "总用量 80" (Total usage 80) label.

```
hollowman@hollow-virtualbox:~/文档/Test
ls -l
总用量 80
-rw-r--r-- 1 hollowman hollowman 44 5月 31 08:50 adjtime
drwxr-xr-x 6 hollowman hollowman 4096 5月 31 08:48 asciidoc
drwxr-xr-x 2 hollowman hollowman 4096 5月 31 08:48 conf.d
drwxr-xr-x 3 hollowman hollowman 4096 5月 31 08:48 dbus-1
-rw-r--r-- 1 hollowman hollowman 378 5月 31 08:50 environment
drwxr-xr-x 2 hollowman hollowman 4096 5月 31 08:48 exports.d
drwxr-xr-x 3 hollowman hollowman 4096 5月 31 08:48 fwupd
-rw-r--r-- 1 hollowman hollowman 973 5月 31 08:50 group-
-rw-r--r-- 1 hollowman hollowman 14 5月 31 08:50 manjaro-release
-rw-r--r-- 1 hollowman hollowman 812 5月 31 08:50 mke2fs.conf
drwxr-xr-x 2 hollowman hollowman 4096 5月 31 08:48 modprobe.d
drwxr-xr-x 2 hollowman hollowman 4096 5月 31 08:48 nginx
-rw-r--r-- 1 hollowman hollowman 388 5月 31 08:50 nsswitch.conf
drwxr-xr-x 3 hollowman hollowman 4096 5月 31 08:48 ostree
drwxr-xr-x 3 hollowman hollowman 4096 5月 31 08:48 pacman.d
-rw-r--r-- 1 hollowman hollowman 1199 5月 31 08:50 pamac.conf
-r--rws--T 1 hollowman hollowman 0 5月 31 08:48 shadow
-rw-r--r-- 1 hollowman hollowman 3975 5月 31 08:50 sudo.conf
drwxr-xr-x 2 hollowman hollowman 4096 5月 31 08:48 timeshift
-rw-r--r-- 1 hollowman hollowman 738 5月 31 08:50 trusted-key.key
-rw-r--r-- 1 hollowman hollowman 51 5月 31 08:50 vdpau_wrapper.cfg
```