上海工程技术大学

实 验 报 告

实验五　文件管理

1. 实验要求

（1）了解文件权限及作用；

（2）掌握文件权限的设置；

（3）掌握文件权限的管理与应用；

（4）掌握文件基本操作；

（5）理解文件系统管理文件的实现机制。

2. 实验内容

1） 验证实验：

① 在用户主目录下创建目录 test，进入 test 目录创建空文件 file1。并以长格形式显示文件信息，注意文件的权限和所属用户和组；

② 在目录 test 中, 新建文件 file2，设置文件权限为 r--r-----。再新建文件 test2，查看其属性，要求设置权限为-rw-------；

③ 查看目录 test 及其中文件的所属用户和组。 把目录 test 及其下的所有文件的所有者改成 bin，所属组改成 daemon。查看设置结果。 删除目录 test 及其下的文件;

④ 查看/etc/inittab 文件的权限属性，并指出该文件的所有者以及文件所属组群;

⑤ 查找/etc 目录下以 i 开头的文件和目录；

⑥ 查找 root 用户所有以 t 开头的文件，并将其保存在/root/tmp 文件中；

⑦ 在/root 下建立/etc/fstab 的符号链接文件；

⑧ 新建文件 test，分别为其建立硬链接文件和符号链接文件。指出硬链接文件的索引号与符号链接文件的索引号的差异；

⑨ 在/usr 目录下创建一个目录 usr_test 和文本文件 test，并建立一个 test 文件的链接 test02。通过修改 test 文件中的内容查看 test 和 test02 中内容的情况，并分析原因。

2） 编程实验，实现一个简单的文件系统，具体要求如下：

① 内存中开辟一块虚拟磁盘空间作为文件存储分区，在其上实现一个简单的基于多级目录的单用户单任务系统中的文件系统；

② 在退出该文件系统的使用时，虚拟文件系统以一个文件的方式保存到磁盘中，以便下次可以把它恢复到内存的虚拟存储空间；

③ 能根据绝对路径和相对路径进行文件查找。

3. 实验结果

（1）①

```
ldd@ldd-virtual-machine:~$ mkdir test
ldd@ldd-virtual-machine:~$ cd test
ldd@ldd-virtual-machine:~/test$ touch file1
ldd@ldd-virtual-machine:~/test$ ll
总用量 8
drwxr-xr-x  2 ldd ldd 4096 12月 21 10:45 ./
drwxr-xr-x 18 ldd ldd 4096 12月 21 10:45 ../
-rw-r--r--  1 ldd ldd    0 12月 21 10:45 file1
```

②

```
ldd@ldd-virtual-machine:~/test$ chmod 440 file2
ldd@ldd-virtual-machine:~/test$ chmod 600 test2
ldd@ldd-virtual-machine:~/test$ ll
总用量 8
drwxr-xr-x  2 ldd ldd 4096 12月 21 11:02 ./
drwxr-xr-x 18 ldd ldd 4096 12月 21 10:45 ../
-rw-r--r--  1 ldd ldd    0 12月 21 10:45 file1
-r--r-----  1 ldd ldd    0 12月 21 11:02 file2
-rw-------  1 ldd ldd    0 12月 21 11:02 test2
```

③

```
ldd@ldd-virtual-machine:~$ su root
密码：
su：认证失败
ldd@ldd-virtual-machine:~$ sudo passwd root
[sudo] ldd 的密码：
输入新的 UNIX 密码：
重新输入新的 UNIX 密码：
passwd：已成功更新密码
ldd@ldd-virtual-machine:~$ su root
密码：
root@ldd-virtual-machine:/home/ldd# chown -R bin:daemon test/
root@ldd-virtual-machine:/home/ldd# ls -lh
-rwx------ 1 ldd ldd      43 11月 14 20:36 ten.sh
drwxr-xr-x 2 bin daemon 4.0K 12月 21 11:02 test
root@ldd-virtual-machine:/home/ldd# rm -rf test
```

④

```
ldd@ldd-virtual-machine:~$ su root
密码:
root@ldd-virtual-machine:/home/ldd#  ls -li /etc/inittab
ls: 无法访问'/etc/inittab': 没有那个文件或目录
root@ldd-virtual-machine:/home/ldd#  ls -li /etc
总用量 1088
262153 drwxr-xr-x  3 root root    4096 7月  25  2018 acp
```

⑤

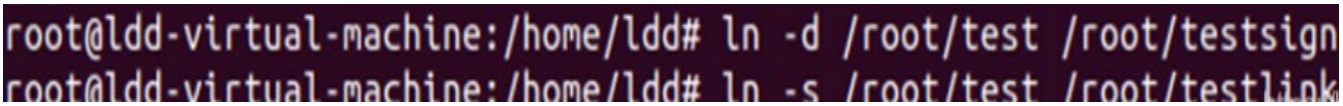
```
ldd@ldd-virtual-machine:/etc$ cd
ldd@ldd-virtual-machine:~$ su root
密码:
root@ldd-virtual-machine:/home/ldd# find /etc/i*
/etc/ifplugd
/etc/ifplugd/action.d
/etc/ifplugd/action.d/action_wpa
/etc/init
/etc/init/anacron.conf
/etc/init/whoopsie.conf
/etc/init.d
/etc/init.d/irqbalance
/etc/init.d/plymouth
/etc/init.d/grub-common
/etc/init.d/cron
/etc/init.d/kerneloops
/etc/init.d/pppd-dns
/etc/init.d/procps
/etc/init.d/saned
/etc/init.d/cups
/etc/init.d/acpid
/etc/init.d/apparmor
```

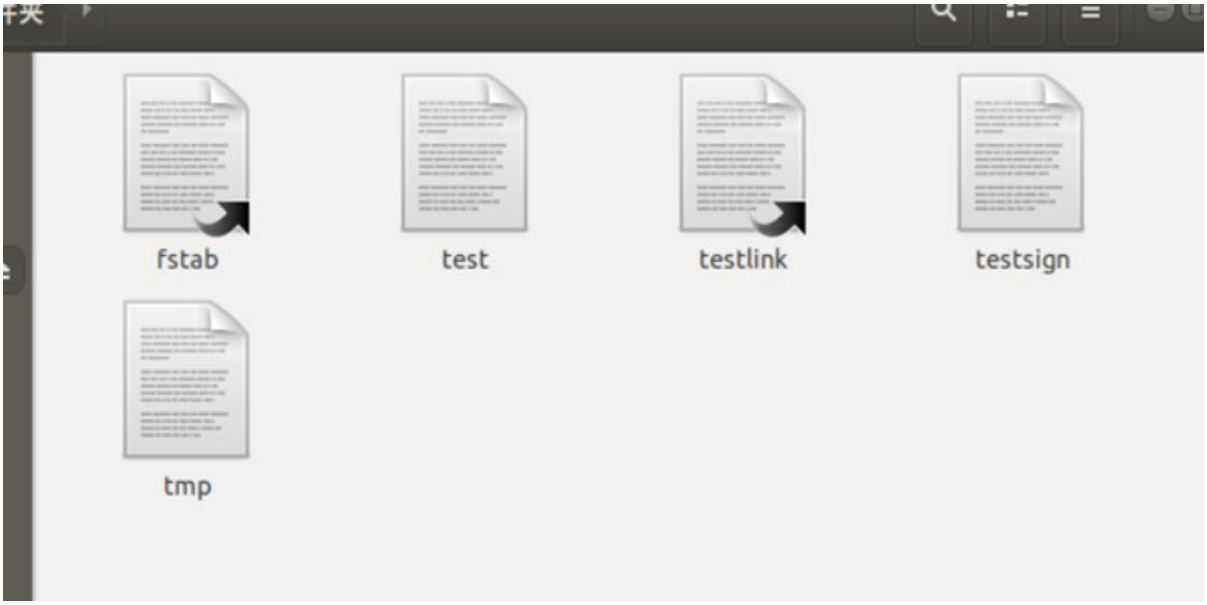
⑥

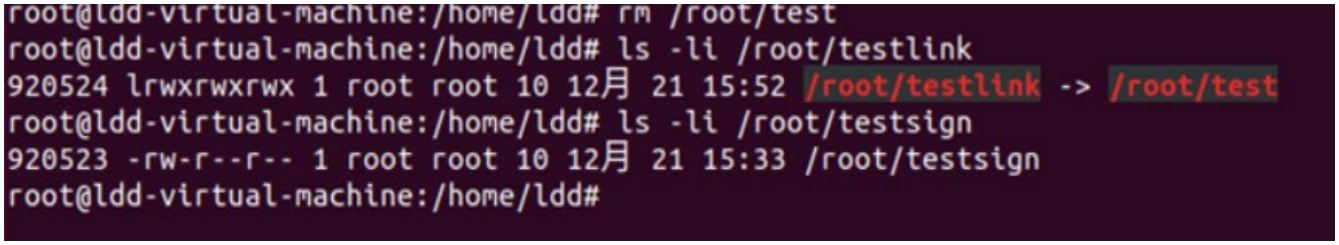
```
ldd@ldd-virtual-machine:~$ su root
密码:
root@ldd-virtual-machine:/home/ldd# find /root/t*
/root/tmp
root@ldd-virtual-machine:/home/ldd# find /root/t* > /root/tmp
root@ldd-virtual-machine:/home/ldd# cat /root/tmp
/root/tmp
root@ldd-virtual-machine:/home/ldd#
```

⑦

⑧







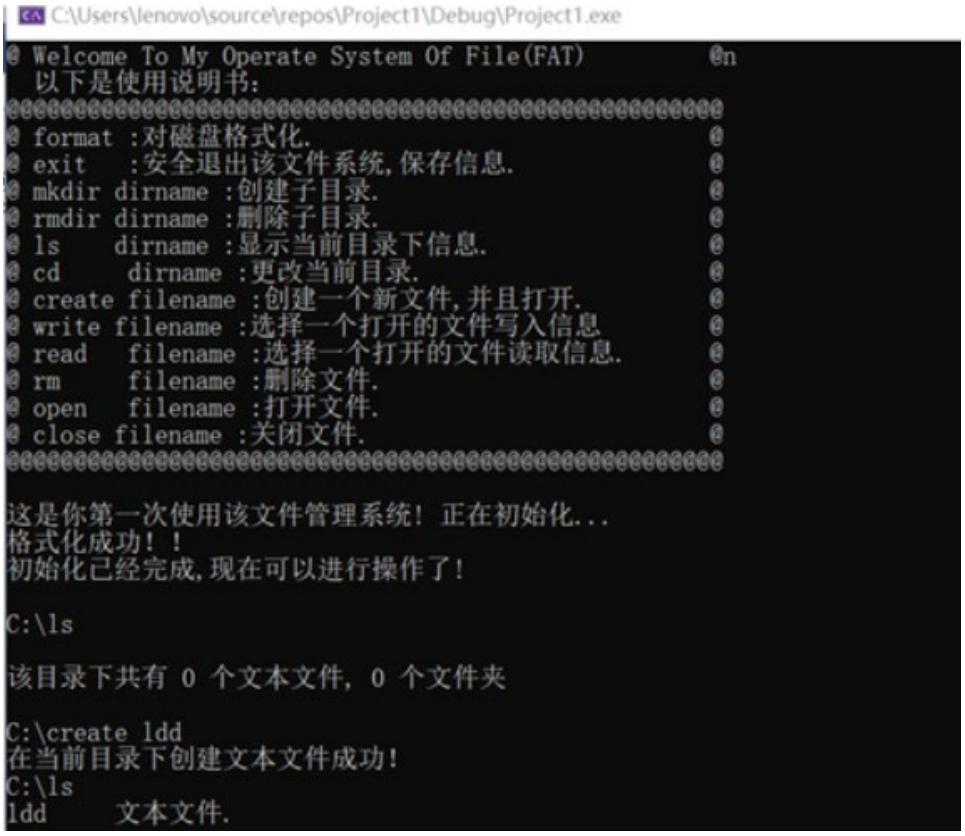
差异：硬链接原文件和链接文件公用一个 iNode 号，说明他们是同一个文件，而软连接原文件和链接文件拥有不同的 iNode 号，表明他们是两个不同的文件。由于 Linux 下的文件是通过索引节点（iNode）来识别文件，硬链接可以认为是一个指针，指向文件索引节点的指针，系统并不为它重新分配 iNode。每添加一个硬链接，文件的链接数就加 1.

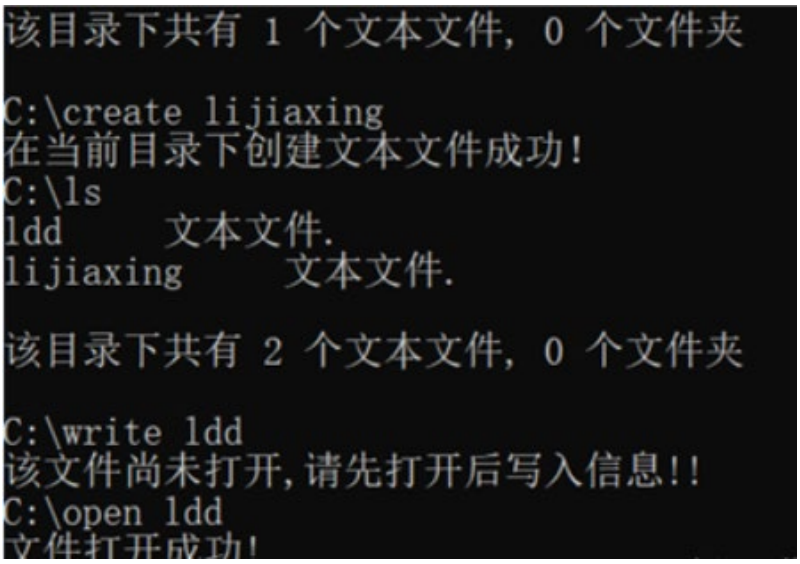
⑨

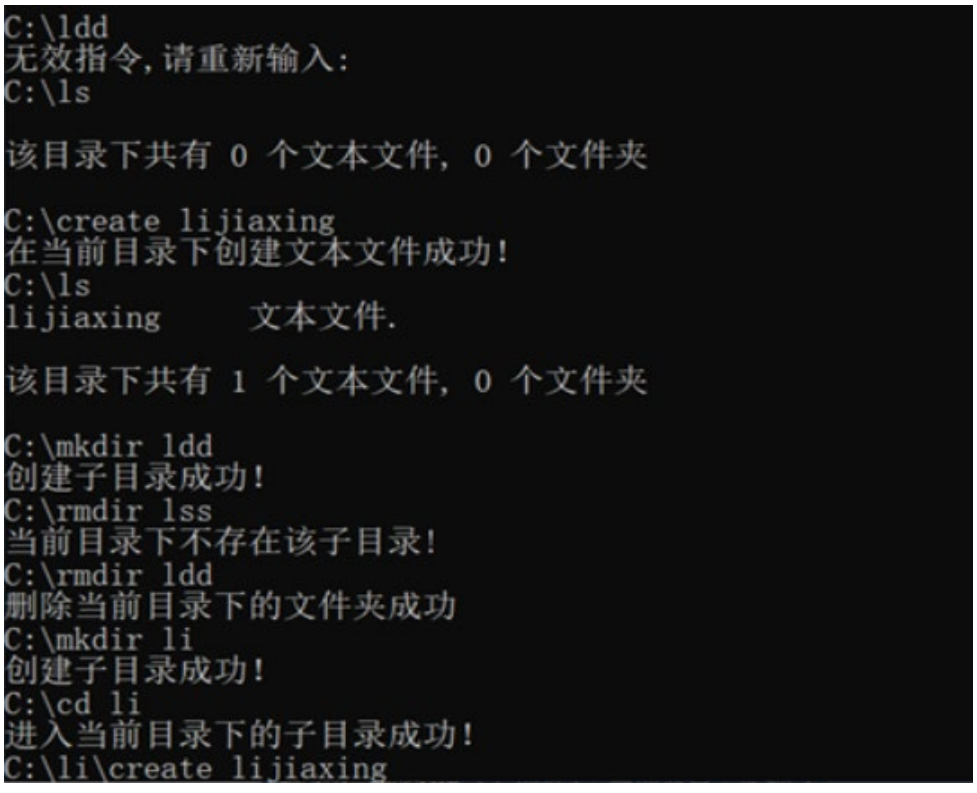
```
root@ldd-virtual-machine:/home/ldd# ls
a       copy.c            l       lock.o  signal.c  公共的  文档
a.out   examples.desktop  l.c     sg      string    模板    下载
b       file1             lock    sg.c    string.c  视频    音乐
copy    fileq             lock.c  signal  ten.sh    图片    桌面
root@ldd-virtual-machine:/home/ldd# mkdir user
root@ldd-virtual-machine:/home/ldd#  cd user
root@ldd-virtual-machine:/home/ldd/user# mkdir user_test
root@ldd-virtual-machine:/home/ldd/user# cd user_test
root@ldd-virtual-machine:/home/ldd/user/user_test# vi test.txt
root@ldd-virtual-machine:/home/ldd/user/user_test# ls
test.txt
root@ldd-virtual-machine:/home/ldd/user/user_test# link test.txt test02
root@ldd-virtual-machine:/home/ldd/user/user_test# cat test.txt
my name is li jiaxing
root@ldd-virtual-machine:/home/ldd/user/user_test# cat test02
my name is li jiaxing
root@ldd-virtual-machine:/home/ldd/user/user_test#
```

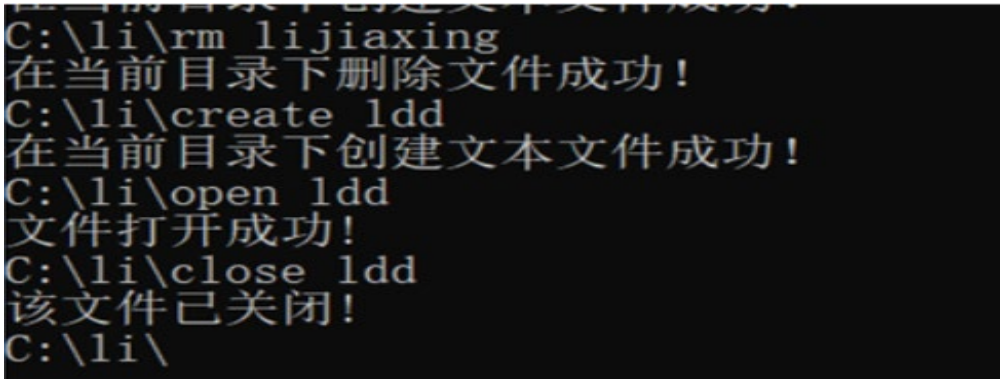
原因：test02 与 test.txt 显示的内容是一样的，当 test.txt 文件里的内容改变时，test02 显示出来的与 test.txt 中的内容是一致的。建立一个 test 文件的链接 test02 就相当于 windows 的创建快捷方式，test02 就是 test.txt 的一个索引。

（2）




该目录下共有 1 个文本文件，0 个文件夹
C:\create lijiaxing
在当前目录下创建文本文件成功!
C:\ls
ldd 文本文件.
lijiaxing 文本文件.
该目录下共有 2 个文本文件，0 个文件夹
C:\write ldd
该文件尚未打开,请先打开后写入信息!!
C:\open ldd


C:\ldd
无效指令,请重新输入:
C:\ls
该目录下共有 0 个文本文件，0 个文件夹
C:\create lijiaxing
在当前目录下创建文本文件成功!
C:\ls
lijiaxing 文本文件.
该目录下共有 1 个文本文件，0 个文件夹
C:\mkdir ldd
创建子目录成功!
C:\rmdir lss
当前目录下不存在该子目录!
C:\rmdir ldd
删除当前目录下的文件夹成功
C:\mkdir li
创建子目录成功!
C:\cd li
进入当前目录下的子目录成功!
C:\li\create lijiaxing


C:\li\rm lijiaxing
在当前目录下删除文件成功!
C:\li\create ldd
在当前目录下创建文本文件成功!
C:\li\open ldd
文件打开成功!
C:\li\close ldd
该文件已关闭!
C:\li\

代码：#include <stdio.h>
#include <memory.h>
#include <string>
#include <iostream>
using namespace std;
#define GENERAL 1//1 代表普通文件 2 代表目录文件 0 表示空文件
#define DIRECTORY 2
#define Zero 0
struct FCB
{
    char fname[16]; //文件名
    char type; // 0 空文件 1 目录文件 2 空文件
    int size;   //文件大小
    int fatherBlockNum;   //当前的父目录盘块号
    int currentBlockNum;   //当前的盘块
    void initialize()
    {
        strcpy(fname, "\0");
        type = Zero;
        size = 0;
        fatherBlockNum = currentBlockNum = 0;
    }
};
const char* FilePath = "D:\\";/*常量设置*/
const int BlockSize = 512;   //盘块大小
const int OPEN_MAX = 5;   //能打开最多的文件数
const int BlockCount = 128;   //盘块数
const int DiskSize = BlockSize * BlockCount;   //磁盘大小
const int BlockFcbCount = BlockSize / sizeof(FCB);//目录文件的最多 FCB 数
int OpenFileCount = 0; // 统计当前打开文件数目
struct OPENLIST   //用户文件打开表
{
    int files;   //当前打开文件数
    FCB f[OPEN_MAX];   //FCB 拷贝
    OPENLIST()
    {
        files = 0;
        for (int i = 0; i < OPEN_MAX; i++) {
            f[i].fatherBlockNum = -1;//为分配打开
            f[i].type = GENERAL;
        }
    }
};
struct dirFile/*----------目录文件结构----------*/

```
{
    struct FCB fcb[BlockFcbCount];
    void init(int _FatherBlockNum, int _CurrentBlockNum, char* name)//父块号，当前块号，
目录名
    {
        strcpy(fcb[0].fname, name); //本身的 FCB
        fcb[0].fatherBlockNum = _FatherBlockNum;
        fcb[0].currentBlockNum = _CurrentBlockNum;
        fcb[0].type = DIRECTORY;        //标记目录文件
        for (int i = 1; i < BlockFcbCount; i++) {
            fcb[i].fatherBlockNum = _CurrentBlockNum; //标记为子项
            fcb[i].type = Zero;       //  标记为空白项
        }
    }
};
struct
DISK/*************************************************************************/
{
    int FAT1[BlockCount];        //FAT1
    int FAT2[BlockCount];        //FAT2
    struct dirFile root;       //根目录
    char data[BlockCount - 3][BlockSize];
    void format() {
        memset(FAT1, 0, BlockCount);        //FAT1
        memset(FAT2, 0, BlockCount);        //FAT2
        FAT1[0] = FAT1[1] = FAT1[2] = -2; //0,1,2 盘块号依次代表 FAT1,FAT2,根目录区
        FAT2[0] = FAT2[1] = FAT2[2] = -2;   //FAT 作备份
        root.init(2, 2,(char*) "C:\\");//根目录区
        memset(data, 0, sizeof(data));//数据区
    }
};
FILE* fp;          //磁盘文件地址
char* BaseAddr;       //虚拟磁盘空间基地址
string currentPath = "C:\\";     //当前路径
int current = 2;       //当前目录的盘块号
string cmd;          //输入指令
struct DISK* osPoint;       //磁盘操作系统指针
char command[16];       //文件名标识
struct OPENLIST* openlist; //用户文件列表指针
int    format();
int    mkdir(char* sonfname);
int rmdir(char* sonfname);
int create(char* name);
int listshow();
```

```
int delfile(char* name);
int changePath(char* sonfname);
int write(char* name);
int exit();
int open(char* file);
int close(char* file);
int    read(char* file);
/*------------初始化-----------------------*/
int format()
{
	current = 2;
	currentPath = "C:\\";	//当前路径
	osPoint->format();//打开文件列表初始化
	delete openlist;
	openlist = new OPENLIST;
	/*-------保存到磁盘上--------*/
	fp = fopen(FilePath, "w+");
	fwrite(BaseAddr, sizeof(char), DiskSize, fp);
	fclose(fp);
	printf("格式化成功！！\n");
	return 1;
}
int    mkdir(char* sonfname)/*-----------------------创建子目录-------------------*/
{//判断是否有重名寻找空白子目录项寻找空白盘块号当前目录下增加该子目录项分配子目录盘块，并且初始化修改 fat 表
	int i, temp, iFAT;
	struct dirFile* dir;		//当前目录的指针
	if (current == 2) //  根目录
		dir = &(osPoint->root);
	else
		dir = (struct dirFile*)(osPoint->data[current - 3]);
	/*--------为了避免该目录下同名文件夹--------*/
	for (i = 1; i < BlockFcbCount; i++) {
		if (dir->fcb[i].type == DIRECTORY && strcmp(dir->fcb[i].fname, sonfname) == 0) {
			printf("该文件夹下已经有同名的文件夹存在了!\n");
			return 0;
		}
	}
	for (i = 1; i < BlockFcbCount; i++) {//查找空白 fcb 序号
		if (dir->fcb[i].type == Zero)
			break;
	}
	if (i == BlockFcbCount) {
		printf("该目录已满!请选择新的目录下创建!\n");
```

```
            return 0;
        }
        temp = i;
        for (i = 3; i < BlockCount; i++) {
            if (osPoint->FAT1[i] == 0)
                break;
        }
        if (i == BlockCount) {
            printf("磁盘已满!\n");
            return 0;
        }
        iFAT = i;
        /*-------------接下来进行分配----------*/
        osPoint->FAT1[iFAT] = osPoint->FAT2[iFAT] = 2;     //2 表示分配给下级目录文件
        //填写该分派新的盘块的参数
        strcpy(dir->fcb[temp].fname, sonfname);
        dir->fcb[temp].type = DIRECTORY;
        dir->fcb[temp].fatherBlockNum = current;
        dir->fcb[temp].currentBlockNum = iFAT;
        //初始化子目录文件盘块
        dir = (struct dirFile*)(osPoint->data[iFAT - 3]);     //定位到子目录盘块号
        dir->init(current, iFAT, sonfname);//iFAT 是要分配的块号，这里的 current 用来指要分配的
块的父块号
        printf("创建子目录成功！\n");
        return 1;
}
int rmdir(char* sonfname)/*-------删除当前目录下的文件夹--------*/
{
        int i, temp, j;//确保当前目录下有该文件,并记录下该 FCB 下标
        struct dirFile* dir;          //当前目录的指针
        if (current == 2)
            dir = &(osPoint->root);
        else
            dir = (struct dirFile*)(osPoint->data[current - 3]);
        for (i = 1; i < BlockFcbCount; i++) {          //查找该目录文件
            if (dir->fcb[i].type == DIRECTORY && strcmp(dir->fcb[i].fname, sonfname) == 0) {
                break;
            }
        }
        temp = i;

        if (i == BlockFcbCount) {
            printf("当前目录下不存在该子目录!\n");
            return 0;
```

```
    }
    j = dir->fcb[temp].currentBlockNum;
    struct dirFile* sonDir;        //当前子目录的指针
    sonDir = (struct dirFile*)(osPoint->data[j - 3]);

    for (i = 1; i < BlockFcbCount; i++) { //查找子目录是否为空目录
        if (sonDir->fcb[i].type != Zero)
        {
            printf("该文件夹为非空文件夹,为确保安全，请清空后再删除!\n");
            return 0;
        }
    }
    /*开始删除子目录操作*/
    osPoint->FAT1[j] = osPoint->FAT2[j] = 0;        //fat 清空
    char* p = osPoint->data[j - 3];          //格式化子目录
    memset(p, 0, BlockSize);
    dir->fcb[temp].initialize();                //回收子目录占据目录项
    printf("删除当前目录下的文件夹成功\n");
    return 1;
}
/*-----------在当前目录下创建文本文件---------------*/
int create(char* name) {
    int i, iFAT;//temp,
    int emptyNum = 0, isFound = 0;             //空闲目录项个数
    struct dirFile* dir;        //当前目录的指针
    if (current == 2)
        dir = &(osPoint->root);
    else
        dir = (struct dirFile*)(osPoint->data[current - 3]);
    for (i = 1; i < BlockFcbCount; i++)//查看目录是否已满//为了避免同名的文本文件
    {
        if (dir->fcb[i].type == Zero && isFound == 0) {
            emptyNum = i;
            isFound = 1;
        }
        else if (dir->fcb[i].type == GENERAL && strcmp(dir->fcb[i].fname, name) == 0) {
            printf("无法在同一目录下创建同名文件!\n");
            return 0;
        }
    }
    if (emptyNum == 0) {
        printf("已经达到目录项容纳上限，无法创建新目录!\n");
        return 0;
    }
```

```
    for (i = 3; i < BlockCount; i++)//查找 FAT 表寻找空白区，用来分配磁盘块号 j
    {
        if (osPoint->FAT1[i] == 0)
            break;
    }
    if (i == BlockCount) {
        printf("磁盘已满!\n");
        return 0;
    }
    iFAT = i;
    /*------进入分配阶段---------*///分配磁盘块
    osPoint->FAT1[iFAT] = osPoint->FAT2[iFAT] = 1;
    /*-----------接下来进行分配----------*///填写该分派新的盘块的参数
    strcpy(dir->fcb[emptyNum].fname, name);
    dir->fcb[emptyNum].type = GENERAL;
    dir->fcb[emptyNum].fatherBlockNum = current;
    dir->fcb[emptyNum].currentBlockNum = iFAT;
    dir->fcb[emptyNum].size = 0;
    char* p = osPoint->data[iFAT - 3];
    memset(p, 4, BlockSize);
    printf("在当前目录下创建文本文件成功！\n");
    return 1;
}
/*-------查询子目录------------*/
int listshow()
{
    int i, DirCount = 0, FileCount = 0;
    //搜索当前目录
    struct dirFile* dir;        //当前目录的指针
    if (current == 2)
        dir = &(osPoint->root);
    else
        dir = (struct dirFile*)(osPoint->data[current - 3]);
    for (i = 1; i < BlockFcbCount; i++) {
        if (dir->fcb[i].type == GENERAL) {    //查找普通文件
            FileCount++;
            printf("%s        文本文件.\n", dir->fcb[i].fname);
        }
        if (dir->fcb[i].type == DIRECTORY) {    //查找目录文件
            DirCount++;
            printf("%s        文件夹.\n", dir->fcb[i].fname);
        }
    }
    printf("\n 该目录下共有  %d  个文本文件, %d  个文件夹\n\n", FileCount, DirCount);
```

```
    return 1;
}
/*---------在当前目录下删除文件-----------*/
int delfile(char* name)
{
    int i, temp, j;
    //确保当前目录下有该文件,并且记录下它的 FCB 下标
    struct dirFile* dir;        //当前目录的指针
    if (current == 2)
        dir = &(osPoint->root);
    else
        dir = (struct dirFile*)(osPoint->data[current - 3]);
    for (i = 1; i < BlockFcbCount; i++) //查找该文件
    {
        if (dir->fcb[i].type == GENERAL && strcmp(dir->fcb[i].fname, name) == 0) {
            break;
        }
    }
    if (i == BlockFcbCount) {
        printf("当前目录下不存在该文件!\n");
        return 0;
    }
    int k;
    for (k = 0; k < OPEN_MAX; k++) {
        if ((openlist->f[k].type = GENERAL) &&
            (strcmp(openlist->f[k].fname, name) == 0)) {
            if (openlist->f[k].fatherBlockNum == current) {
                break;
            }
            else {
                printf("该文件未在当前目录下!\n");
                return 0;
            }
        }
    }
    if (k != OPEN_MAX) {
        close(name);
    }
    //从打开列表中删除     /*开始删除文件操作*/
    temp = i;
    j = dir->fcb[temp].currentBlockNum;      //查找盘块号 j
    osPoint->FAT1[j] = osPoint->FAT2[j] = 0;        //fat1,fat2 表标记为空白
    char* p = osPoint->data[j - 3];
    memset(p, 0, BlockSize); //清除原文本文件的内容
```

```
        dir->fcb[temp].initialize();       //type=0;         //标记该目录项为空文件
        printf("在当前目录下删除文件成功！\n");
        return 1;
    }
    /*--------------进入当前目录下的子目录--------------*/
    int changePath(char* sonfname)
    {
        struct dirFile* dir;         //当前目录的指针
        if (current == 2)
            dir = &(osPoint->root);
        else
            dir = (struct dirFile*)(osPoint->data[current - 3]);
        /*回到父目录*/
        if (strcmp(sonfname, "..") == 0) {
            if (current == 2) {
                printf("你现已经在根目录下!\n");
                return 0;
            }
            current = dir->fcb[0].fatherBlockNum;
            currentPath = currentPath.substr(0, currentPath.size() - strlen(dir->fcb[0].fname) - 1);
            return 1;
        }
        /*进入子目录*///确保当前目录下有该目录,并且记录下它的 FCB 下标
        int i, temp;
        for (i = 1; i < BlockFcbCount; i++) {         //查找该文件
            if (dir->fcb[i].type == DIRECTORY && strcmp(dir->fcb[i].fname, sonfname) == 0) {
                temp = i;
                break;
            }
        }
        if (i == BlockFcbCount) {
            printf("不存在该目录!\n");
            return 0;
        }
        //修改当前文件信息
        current = dir->fcb[temp].currentBlockNum;
        currentPath = currentPath + dir->fcb[temp].fname + "\\";
        printf("进入当前目录下的子目录成功！\n");
        return 1;
    }
    int exit() {//保存到磁盘上 C:\myfiles
        //将所有文件都关闭/*--------System exit----------------------*/
        fp = fopen(FilePath, "w+");
        fwrite(BaseAddr, sizeof(char), DiskSize, fp);
```

```
    fclose(fp);
    //释放内存上的虚拟磁盘
    free(osPoint);
    //释放用户打开文件表
    delete openlist;
    printf("退出文件系统成功！\n\n");
    return 1;
}
int write(char* name)/*-------------在指定的文件里记录信息----------------*/
{
    int i;
    char* startPoint, * endPoint;
    //在打开文件列表中查找 file(还需要考虑同名不同目录文件的情况!!!)
    for (i = 0; i < OPEN_MAX; i++) {
        if (strcmp(openlist->f[i].fname, name) == 0) {
            if (openlist->f[i].fatherBlockNum == current) {
                break;
            }
            else {
                printf("该文件处于打开列表中，本系统只能改写当前目录下文件!\n");
                return 0;
            }
        }
    }
    if (i == OPEN_MAX) {
        printf("该文件尚未打开,请先打开后写入信息!!\n");
        return 0;
    }
    int active = i;
    int fileStartNum = openlist->f[active].currentBlockNum - 3;
    startPoint = osPoint->data[fileStartNum];
    endPoint = osPoint->data[fileStartNum + 1];
    printf("请输入文本以 Ctrl D 号结束:\t");
    char input;
    while (((input = getchar()) != 4)) {
        if (startPoint < endPoint - 1) {
            *startPoint++ = input;
        }
        else {
            printf("达到单体文件最大容量！");
            *startPoint++ = 4;
            break;
        }
    }
```

```
    return 1;
}
int   read(char* file)/*---------选择一个打开的文件读取信息----------*/
{
    int i, fileStartNum;
    char* startPoint, * endPoint;
    //struct dirFile *dir;
    //在打开文件列表中查找 file(还需要考虑同名不同目录文件的情况!!!)
    for (i = 0; i < OPEN_MAX; i++) {
        if (strcmp(openlist->f[i].fname, file) == 0) {
            if (openlist->f[i].fatherBlockNum == current) {
                break;
            }
            else {
                printf("该文件处于打开列表中，本系统只能阅读当前目录下文件!\n");
                return 0;
            }
        }
    }
    if (i == OPEN_MAX) {
        printf("该文件尚未打开,请先打开后读取信息!\n");
        return 0;
    }
    int active = i;//计算文件物理地址
    fileStartNum = openlist->f[active].currentBlockNum - 3;
    startPoint = osPoint->data[fileStartNum];
    endPoint = osPoint->data[fileStartNum + 1];
    printf("该文件的内容为:   ");
    while ((*startPoint) != 4 && (startPoint < endPoint)) {
        putchar(*startPoint++);
    }
    printf("\n");
    return 1;
}
int open(char* file)//打开文件/*当前目录下添加一个打开文件*/
{
    int i, FcbIndex;
    //确保没有打开过该文件 = 相同名字 + 相同目录
    for (i = 0; i < OPEN_MAX; i++) {
        if (openlist->f[i].type == GENERAL && strcmp(openlist->f[i].fname, file) == 0
            && openlist->f[i].fatherBlockNum == current)
        {
            printf("该文件已经被打开!\n");
            return 0;
```

```
            }
        }
        //确保有空的打开文件项
        if (openlist->files == OPEN_MAX) {
            printf("打开文件数目达到上限!无法再打开新文件.\n");
            return 0;
        }
        //确保当前目录下有该文件,并且记录下它的 FCB 下标
        struct dirFile* dir;          //当前目录的指针
        if (current == 2)
            dir = &(osPoint->root);
        else
            dir = (struct dirFile*)(osPoint->data[current - 3]);
        for (i = 1; i < BlockFcbCount; i++) {          //查找该文件
            if (dir->fcb[i].type == GENERAL && strcmp(dir->fcb[i].fname, file) == 0) {
                FcbIndex = i;
                break;
            }
        }
        if (i == BlockFcbCount) {
            printf("当前目录下不存在该文件!\n");
            return 0;
        }
        //装载新文件进入打开文件列表,(FCB 信息，文件数++)
        openlist->f[OpenFileCount] = dir->fcb[FcbIndex]; //FCB 拷贝
        openlist->files++;
        printf("文件打开成功!\n");
        OpenFileCount++;
        return 1;
    }
    int close(char* file)
    {
        //释放该文件所占内存//释放用户打开文件列表表项
        int i;
        //在打开文件列表中查找  file(还需要考虑同名不同目录文件的情况!!!)
        for (i = 0; i < OPEN_MAX; i++) {
            if ((openlist->f[i].type = GENERAL) &&
                (strcmp(openlist->f[i].fname, file) == 0)) {
                if (openlist->f[i].fatherBlockNum == current) {
                    break;
                }
                else {
                    printf("该文件已打开，但未在当前目录下，无法关闭!\n");
                    return 0;
```

```
            }
        }
    }
    if (i == OPEN_MAX) {
        printf("该文件未在打开列表中!\n");
        return 0;
    }
    int active = i;
    openlist->files--;
    openlist->f[active].initialize();
    OpenFileCount--;
    printf("该文件已关闭!\n");
    return 1;
}
int main()
{
    printf("@ Welcome To My Operate System Of File(FAT)              @n");
    printf("\n   以下是使用说明书：\n");//使用说明书
    printf("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n");
    printf("@ format :对磁盘格式化.                                  @ \n");
    printf("@ exit     :安全退出该文件系统,保存信息.                 @ \n");
    printf("@ mkdir dirname :创建子目录.                             @ \n");
    printf("@ rmdir dirname :删除子目录.                             @ \n");
    printf("@ ls       dirname :显示当前目录下信息.                  @ \n");
    printf("@ cd        dirname :更改当前目录.                       @ \n");
    printf("@ create filename :创建一个新文件,并且打开.              @ \n");
    printf("@ write filename :选择一个打开的文件写入信息             @ \n");
    printf("@ read      filename :选择一个打开的文件读取信息.        @ \n");
    printf("@ rm         filename :删除文件.                         @ \n");
    printf("@ open      filename :打开文件.                          @\n");
    printf("@ close filename :关闭文件.                              @\n");
    printf("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n\n");
    openlist = new OPENLIST;//创建用户文件打开表
    BaseAddr = (char*)malloc(DiskSize);//申请虚拟空间并且初始化
    osPoint = (struct DISK*)(BaseAddr);//虚拟磁盘初始化
    if ((fp = fopen(FilePath, "r")) != NULL) {//加载磁盘文件
        fread(BaseAddr, sizeof(char), DiskSize, fp);
        printf("加载磁盘文件( %s )成功,现在可以进行操作了!\n\n", FilePath);
    }
    else {
        printf("这是你第一次使用该文件管理系统!\t 正在初始化...\n");
        format();
```

```
        printf("初始化已经完成,现在可以进行操作了!\n\n");
    }
    while (1) {
        cout << currentPath;
        cin >> cmd;
        if (cmd == "format") {
            format();
        }
        else if (cmd == "mkdir") {
            cin >> command;
            mkdir(command);
        }
        else if (cmd == "rmdir") {
            cin >> command;
            rmdir(command);
        }
        else if (cmd == "ls") {
            listshow();
        }
        else if (cmd == "cd") {
            cin >> command;
            changePath(command);
        }
        else if (cmd == "create") {
            cin >> command;
            create(command);
        }
        else if (cmd == "write") {
            cin >> command;
            write(command);
        }
        else if (cmd == "read") {
            cin >> command;
            read(command);
        }
        else if (cmd == "rm") {
            cin >> command;
            delfile(command);
        }
        else if (cmd == "open") {
            cin >> command;
            open(command);
        }
        else if (cmd == "close") {
```

```
            cin >> command;
            close(command);
        }
        else if (cmd == "exit") {
            exit();
            break;
        }
        else cout << "无效指令,请重新输入:" << endl;
    }
    printf("Thank you for using my file system!\n");
    return 1;
}
```

4. 实验小结

通过本次实验，我了解文件权限及作用，掌握文件权限的设置，掌握文件权限的管理与应用，掌握文件基本操作，理解文件系统管理文件的实现机制。