FILE SYSTEM

Q:

1.自顶向下实现功能时，考虑write命令中接受到终端命令ESC需要退出，那么就需要一个可以接受到esc命令的功能

```
int getch(void)
{
        int c=0;
        struct termios org_opts, new_opts;
        int res=0;
        //-----  store old settings -----------
        res=tcgetattr(STDIN_FILENO, &org_opts);
        assert(res==0);
        //---- set new terminal parms --------
        memcpy(&new_opts, &org_opts, sizeof(new_opts));
        new_opts.c_lflag &= ~(ICANON | ECHO | ECHOE | ECHOK | ECHONL | ECHOPRT | ECHOKE | ICRNL);
        tcsetattr(STDIN_FILENO, TCSANOW, &new_opts);
        c=getchar();
            //------  restore old settings ---------
        res=tcsetattr(STDIN_FILENO, TCSANOW, &org_opts);assert(res==0);
        return c;
}
```

在`csdn`博客中寻找到了一种方法。用于从终端获取一个字符而不回显（不在终端上显示用户输入的字符）。

```
int getch()
{
    int ch;
    struct termios oldt, newt;
    tcgetattr(STDIN_FILENO,&oldt);//获取与终端相关的参数
    newt=oldt;
    newt.c_lflag &= ~(ECHO|ICANON);
    tcsetattr(STDIN_FILENO,TCSANOW,&newt);
    ch=getchar();
    tcsetattr(STDIN_FILENO,TCSANOW,&oldt);
    return ch;
}
```

此函数用于实现使用write命令时输入esc可以终止输入循环；

2. 考虑顶层用户接口shell的实现，使用无限循环，哈希表实现命令的匹配和执行；
3. 密码字段的存储位置是否应该重新设计一个数据结构，可能破坏文件系统的结构；考虑在已有的数据结构中的空闲字段用来存储密码；
4. 读取用户输入时总是出错，发现应该使用一个`getchar()`读出回车，再进行内容的读取；

5. 考虑使用三个文件，`init.h`用来定义数据结构和宏；`ext2_func.h`用来声明定义的所有函数；`ext2.h`主文件中实现所有功能；
6. 用文件模拟硬盘存储文件系统，使用`fread(),fwrite(),fseek()`,需要找到当前的物理位置；需要根据逻辑地址，计算出seek所需的物理地址；
7. 初始化后发现文件系统发生错误，因为未将索引指针回到根目录，导致了索引的混乱；
8. 关于时间的打印；使用localtime和asctime函数；
9. 读写数据时，对文件中物理地址寻找的困难，需要考虑数据块；前6个块直接索引，后面间接索引的数据的块号和物理地址偏移量的计算。

构建数据结构

过程：

1.**首先构建需要的数据结构，放入`init.h`文件中；**

组描述符，索引结点，目录文件等；

```
//不考虑组和其他用户，只考虑所有者的权限
#define DIR_R_MODE 0b0000001000000100
#define DIR_W_MODE 0b0000001000000010
#define DIR_RW_MODE 0b0000001000000110

#define FILE_R_MODE 0b0000000100000100
#define FILE_W_MODE 0b0000000100000010
#define FILE_X_MODE 0b0000000100000001
#define FILE_RW_MODE 0b0000000100000110
#define FILE_RWX_MODE 0b0000000100000111
//组描述符
struct ext2_group_desc{
    char bg_volume_name[16]; //卷名
    uint16_t bg_block_bitmap; //保存块位图的块号
    uint16_t bg_inode_bitmap; //保存索引结点位图的块号
    uint16_t bg_inode_table; //索引结点表的起始块号
    uint16_t bg_free_blocks_count; //本组空闲块的个数
    uint16_t bg_free_inodes_count; //本组空闲索引结点的个数
    uint16_t bg_used_dirs_count; //本组目录的个数
    char bg_pad[4];//填充0xff
};
//索引结点
struct ext2_inode
{
    uint16_t i_mode; //文件类型及访问权限,
    uint16_t i_blocks; //文件的数据块个数
    uint32_t i_size; //文件大小(字节)
    uint64_t i_atime; //访问时间
    uint64_t i_ctime; //创建时间
    uint64_t i_mtime; //修改时间
    uint64_t i_dtime; //删除时间
    uint16_t i_block[8]; //指向数据块的指针
    char i_pad[8]; //填充1(0xff)
```

```
};
//目录文件
struct ext2_dir_entry
{
    uint16_t inode; //索引结点号
    uint16_t rec_len; //目录项长度
    uint8_t name_len; //文件名长度
    uint8_t file_type; //文件类型
    char name[MAX_NAME_LEN]; //文件名
};
//一些宏定义
#define blocks 4611 //1+1+1+512+4096
#define blocksize 512 //每块字节数
#define inodesize 64 //索引长度
#define data_begin_block 515 //数据开始块，1+1+1+512
#define inode_begin_block 3 //索引结点开始块，1+1+1
#define dirsize sizeof(struct ext2_dir_entry) //目录体长度
#define PATH "FS.txt" //文件系统
```

顶层Shell设计

2.**考虑主函数main的设计**

需要维护一个当前的目录或文件的索引结点，初始化文件系统，并为用户提供shell界面；

考虑用户login功能的实现，密码错误即退出文件系统，正确才开启shell；

```
int main() {
    struct ext2_inode current;
    printf("Authorized users only. All activities may be monitored and reported.\n");
    initialize_ext2(&current);
    if(login()){
        printf("Wrong password!\n");
        quit_display();
        return 1;
    }
    else printf("\n\n\n\n                Welcome to Ext2 File System!                \n\n\n\n");
    shell(current);
    return 0;
}
```

main函数大致实现即是如此；

考虑用户密码的维护，只有一个组描述符，剩余的480字节可以分一些给密码字段。

将组描述符中填充字段修改为密码字段，设置12个字节的密码位；

```
char bg_password[12];
```

3.**先考虑主函数需要的登录函数login的实现；**

使用`strcmp`函数判断输入是否与密码符合

```
int login(){
    char password[12];
    printf("Please input the password:");
    scanf("%s",password);
    getchar();
    if(strcmp(groupDesc.bg_password,password)==0)
        return 0;
    else
        return 1;
}
```

4.**考虑一个用户层命令password，用于修改密码；**

用户应当输入正确的旧密码才能修改密码；先对用户输入的旧密码判断正误；

输入新密码后，将其录入到组描述符的密码字段，同时更新文件中的组描述符的内容；

```
printf("Please input the old password\n");
scanf("%s",password);
getchar();
if(strcmp(password,groupDesc.bg_password)!=0){
    printf("False Password!\n");
    return 1;
}
```

```
while(1){
    printf("Please input the new password(12 words max):\n");
    scanf("%s",password);
    getchar();
    while(1){
        printf("Modify the password?[Y/N]");
        scanf("%c",&ch);
        getchar();
        if(ch=='N'||ch=='n'){
            printf("The modification has canceled!\n");
            return 1;
        }
        else if(ch=='Y'||ch=='y'){
            strcpy(groupDesc.bg_password,password);
            FILE *fp=NULL;
            while(fp==NULL)
                fp= fopen(PATH,"r+");
            fseek(fp,0,SEEK_SET);
            fwrite(&groupDesc,sizeof(struct ext2_group_desc),1,fp);//更新组描述符
            fclose(fp);
            return 0;
        }
```

```
//修改密码password命令
int Password(){
    char password[12];
    char ch;
    printf("Please input the old password\n");
    scanf("%s",password);
    getchar();
    if(strcmp(password,groupDesc.bg_password)!=0){
        printf("False Password!\n");
        return 1;
    }
    while(1){
        printf("Please input the new password(12 words max):\n");
        scanf("%s",password);
        getchar();
        while(1){
            printf("Modify the password?[Y/N]");
            scanf("%c",&ch);
            getchar();
            if(ch=='N'||ch=='n'){
                printf("The modification has canceled!\n");
                return 1;
            }
            else if(ch=='Y'||ch=='y'){
                strcpy(groupDesc.bg_password,password);
                FILE *fp=NULL;
                while(fp==NULL) fp= fopen(PATH,"r+");
                fseek(fp,0,SEEK_SET);//将文件指针移回最前面
                fwrite(&groupDesc,sizeof(struct ext2_group_desc),1,fp);//更新组描
述符
                fclose(fp);
                return 0;
            }
```

```
            else{
                printf("error input!\n");
            }
        }
    }
}
```

5. **考虑结束函数的设计；quit_display**

```
void quit_display(){
    printf("This File System is closed successfully!\n");
}
```

6. **考虑文件系统的初始化**

   需要考虑文件存在时，将文件中的组描述符，索引结点都读取出来；若文件系统不存在则需要我们格式化一个新的；

```
fseek(fp,0,SEEK_SET);
fread(&groupDesc,sizeof(struct ext2_group_desc),1,fp);
fseek(fp,inode_begin_block*blocksize,SEEK_SET);
fread(&inode,inodesize,1,fp);
fclose(fp);
```

分别移动文件指针到组描述符开始处和索引结点开始处进行读取；

7. **考虑格式化文件系统format函数的设计；**

   分别对所有的内容都进行初始化；

   先将所有4611块的内容都置为0；

```
for(i=0;i<blocks;i++) {//初始化所有的4611块为0
    fseek(fp,i*blocksize,SEEK_SET);//移动指针的位置
    fwrite(&zero, sizeof(zero),1,fp);//一次写入512个字节，全部置为0
}
```

对组描述符结构的所有内容进行分别初始化：

```
//初始化组描述符
strcpy(groupDesc.bg_volume_name,"my_ext2"); //初始化卷名
groupDesc.bg_block_bitmap=1;//保存数据块位图的块号为1
groupDesc.bg_inode_bitmap=2;//保存索引结点位图的块号为2
groupDesc.bg_inode_table=3;//第一个索引结点表的块号
groupDesc.bg_free_blocks_count=4095; //除去一个初始化目录，空闲块减一
groupDesc.bg_free_inodes_count=4095; //空闲索引结点减一
groupDesc.bg_used_dirs_count=1; //一个初始化目录
strcpy(groupDesc.bg_password,"123456");//将密码值初始化为123456
//将组描述符放入第一个块号
fseek(fp,0*blocksize,SEEK_SET);
fwrite(&groupDesc, sizeof(struct ext2_group_desc),1,fp);
```

对后面的数据块位图和索引结点初始化；

数据块第一块已使用，将其置为1，剩余4095块都为0；第一个索引结点也已使用，位图同数据块；

zero数组最前面的位置为1，后面均为0；

```
//初始化数据块位图和索引结点位图
zero[0]=0x80000000; //十六进制一共4个字节，最高位为1，表示第一个数据块已经用了
fseek(fp,1*blocksize,SEEK_SET); //第二块为数据块位图
fwrite(&zero,sizeof(zero),1,fp);
fseek(fp,2*blocksize,SEEK_SET); //第三块为索引结点位图
fwrite(&zero,sizeof(zero),1,fp);
```

根据索引结点的数据结构来初始化索引结点表；第一个直接索引指向了第一个数据块，其他7个都未使用，初始化为-1；

```
//初始化索引结点表
inode.i_mode=2;//表示这是一个目录type为2
inode.i_blocks=1;//刚开始所有的目录只占用一个数据块
inode.i_size=dirsize*2; //所有目录文件加起来的文件大小，刚开始只有当前目录和上一个目录文件
inode.i_ctime=now;
inode.i_atime=now;
inode.i_mtime=now;
inode.i_dtime=0;
memset(inode.i_block,-1, sizeof(inode.i_block));//初始化索引指针为-1
inode.i_block[0]=0;//第一个直接索引指向第一个数据块
fseek(fp,3*blocksize,SEEK_SET);//移动文件指针到第四块
fwrite(&inode,sizeof(struct ext2_inode),1,fp);//在第四块(索引结点表第一块)中写入一个索引结点
```

初始化第一个数据块；存放的是目录文件；包含当前目录“."和上一级目录"..“；

```
//在第一个数据块中写入两个目录体
//初始化当前目录 "."
dir.inode=0; //表示是第一个索引结点号
dir.rec_len= dirsize;//为了简单，所有的目录项长度都是固定的，为32个字节
dir.name_len=1;//当前目录名为".",长度为1
dir.file_type=2;//目录文件type为2
strcpy(dir.name,".");
fseek(fp,data_begin_block*blocksize,SEEK_SET);//移动文件指针到数据块开始处
fwrite(&dir,dirsize,1,fp);//将当前目录写入数据块
//初始化一个上一个目录 ".."
dir.inode=0; //表示是第一个索引结点号
dir.rec_len= dirsize;
dir.name_len=2;
dir.file_type=2;
strcpy(dir.name,"..");
fseek(fp,data_begin_block*blocksize+dirsize,SEEK_SET);//移动文件指针到上面刚写完的后面
fwrite(&dir,dirsize,1,fp);//写入上一级目录
fclose(fp);
```

8. **考虑实现文件系统提供的一系列用户命令入口——shell**；

   设置一个无限循环，来获取用户输入的命令；

   设置一个哈希表，来通过数字执行用户输入的命令；

   设置命令字符串，通过`scanf`读取用户的输入；

```
printf("\n[wzy@ext2fs %s]# ",current_dir);//显示命令接受字段，并显示当前所在目录
scanf("%s",cmd);
getchar();
int i;
for(i=0;i<14;i++){
    if(strcmp(cmd,hash_t[i])==0) break;//返回与hash_t匹配命令的序号值
}
```

需要给用户反馈当前的目录名，考虑设计新函数获取当前目录名；

```
//根据当前指针所在获取当前目录
get_dir_name(current_dir,current);
```

9. **create和delete//mkdir和rmdir**

创建和删除文件；

```
if(i==0||i==1){//create/delete file

    scanf("%s",msg_2);
    getchar();

    //对创建和删除分别处理
    if(i==0){
        //反馈创建是否成功
        if(Create(1,&current,msg_2)==1) printf("%s can't be successfully created!\n",msg_2);
        else printf("%s is created successfully\n",msg_2);
    }
    else{
        //反馈删除是否成功
        if(Delete(1,&current,msg_2)==1) printf("%s can't be successfully deleted\n",msg_2);
        else printf("%s is deleted successfully\n",msg_2);
    }
}
```

分别考虑设计`creat`和`delete`函数，并根据type值灵活处理文件和目录；

在shell中调用两个函数的接口；

```
else if(i==11||i==12){//mkdir/rmdir

    scanf("%s",msg_2);
    getchar();

    //对创建和删除分别处理
    if(i==11){
        //反馈创建是否成功
        if(Create(2,&current,msg_2)==1) printf("%s can't be successfully created!\n",msg_2);
        else printf("%s is created successfully\n",msg_2);
    }
    else{
        //反馈删除是否成功
        if(Delete(2,&current,msg_2)==1) printf("%s can't be successfully deleted\n",msg_2);
        else printf("%s is deleted successfully\n",msg_2);
    }
}
```

10. **实现cd命令**

将用户输入的路径信息获取到path数组中，并区分分隔符和路径信息；

读取到某个目录名时，使当前的索引指针进入目录，若发生错误则需要返回原目录；

将两个分隔符中间的目录名单独取出，并设计一个open函数测试目录或文件是否存在；

```
//cd
else if(i==2){
    scanf("%s",msg_2);//读取用户输入的路径
    getchar();
    int p=0;
    int q=0;
    node=current;//保存当前的目录信息，以便发生错误可以返回原目录
    while(q<=strlen(msg_2)){
        path[p]=msg_2[q];//将读取到的信息按位存入路径path中
        if(path[p]=='/'){//当出现分隔符时
            //第一个为/，表示要跳到根目录去
            if(q==0) initialize(&current);
            else if(p==0){
                printf("path input error!\n");
                current=node;//回到原来的目录
                break;
            }
            else{//已经读取完毕
                path[p]='\0';
                if(Open(&current,path)==1){
                    printf("path input error!\n");
                    current=node;//回到原来的目录
                    break;
                }
            }
            p=0;//重置
        }
        else if(path[p]=='\0'){//读到最后一位
            if(p==0) break;
            if(Open(&current,path)==1){
                printf("path input error!\n");
                current=node;//回到原来的目录
                break;
            }
        }
        else p++;

        q++;
    }
}
```

11. **close**

    考虑设计一个关闭目录的函数，并反馈是否关闭成功；和cd..实现相同的功能；

    在shell中只需先用接口实现即可；

```
else if(i==3){//close
    if(Close(&current)==1){
        printf("Something went wrong!\n");
        break;
    }
}
```

11. **read&write**

    考虑设计一个read函数，通过传入当前目录和文件名来读取文件内容；

    并做出失败反馈；反馈权限；考虑设计检查权限和修改权限函数；

    考虑设计一个write函数，通过传入当前目录和文件名来写入文件内容；

    并做出失败反馈；同样通过权限访问控制；

```
else if(i==4){//read
    scanf("%s",msg_2);
    getchar();
    if(check_auth(&current,msg_2)==1||check_auth(&current,msg_2)==4){
        if(Read(&current,msg_2)==1) printf("Sorry! Failed read\n");
    }
    else {
        printf("Sorry! Insufficient permissions.\n");
    }
}
else if(i==5){//write
    scanf("%s",msg_2);
    getchar();
    if(check_auth(&current,msg_2)==0||check_auth(&current,msg_2)==1){
        if(Write(&current,msg_2)==1) printf("Sorry! Failed Written\n");
    }
    else{
        printf("Sorry! Insufficient permissions.\n");
    }
}
```

12. **password**

    上面已设计了password函数接口，并在函数中处理了失败反馈，直接调用函数即可；

13. **format**

    上面已设计format函数接口，只需要在shell中提示用户是否进行格式化操作即可，并需要将目前的索引指针返回到根目录；格式化前需要输入密码；

```
else if(i==7){//format
    while(1){
        printf("FORMAT?\n");
        printf("[Y/N]: ");
        scanf("%c",&msg_1);//读取用户输入的单字符
        getchar();
        if(msg_1=='N'||msg_1=='n') break;
        else if(msg_1=='Y'||msg_1=='y'){
            if(login()==1) break;
            format();//调用format函数
            ret_root(&current);//初始化索引指针到根目录
            printf("Format success!\n");
            break;
        }
        else{
            printf("Please input [Y/N]: \n");//输入了其他invalid内容
        }
    }
    printf("Wrong password! Format Failed.");
}
```

14. **quit**

    只需要退出shell的无限循环即可退出此文件系统；

    并添加一些信息反馈；

```
else if(i==8){//quit
    quit_display();
    return;
}
```

15. **Ls**

    设计一个通过当前索引结点列出当前目录所有项的函数ls；

16. **help:**

    对命令的格式和功能提供帮助和提示

17. **chmod：**

    对文件的权限进行修改，考虑设计一个chmod函数；

```
else if(i==13){
    scanf("%s",msg_2);
    getchar();
    char auth[10]={0};
    scanf("%s",auth);
    getchar();
    if(chmod(&current,msg_2,auth)==1) printf("Sorry! Modification is Failed or Not exist.\n");
}
```

顶层函数接口实现

18. **create**

    1. **查找是否存在相同文件：**
    2. **为新文件或目录寻找空的索引结点：**
    3. **读取当前目录的 "." 目录体信息：**
    4. **根据文件类型初始化新的索引结点：**
    5. **写入当前目录和上一级目录的目录体信息：**
    6. **将新的索引结点写入索引结点表：**
    7. **将新的目录体信息写入当前目录：**
    8. **更新当前目录的索引结点信息：**

```
int Create(int type,struct ext2_inode *current,char *name){
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    u_int32_t block_location;//块位置
    u_int32_t node_location;//索引结点位置
    u_int32_t dir_entry_location;//目录体位置

    struct ext2_inode anode;
    struct ext2_dir_entry aentry,bentry;
    time_t now;
    time(&now);
    for(int i=0;i<current->i_size/dirsize;i++){//搜索当前目录索引结点，查看是否存在相同的文件
        fseek(fp, dir_entry_position(i*dirsize,current->i_block),SEEK_SET);
        fread(&aentry,dirsize,1,fp);//读取目录体
        if((aentry.file_type==type)&&(strcmp(aentry.name,name)==0)) return 1;//有相同文件，退出
    }

    node_location=Find_Empty_inode();//为新创建的文件或者目录寻找一个空的索引结点
    fseek(fp, dir_entry_position(0,current->i_block),SEEK_SET);
    fread(&bentry,dirsize,1,fp);//取出当前目录索引结点中指向的数据块中的"."的目录体
    if(type==1){//文件
        anode.i_mode=1;
        anode.i_size=0;//暂时没有内容
        anode.i_blocks=0;//没有为它分配数据块
        anode.i_atime=now;
        anode.i_ctime=now;
        anode.i_mtime=now;
        anode.i_dtime=0;
        memset(anode.i_block,-1,sizeof(anode.i_block));//全部初始化为-1，表示没有数据块指针
        for(int i=0;i<8;i++)
            anode.i_pad[i]=(char)0xff;//没有任何含义，只是为了填充
    }
    else{//创建目录
        anode.i_mode=2;
        anode.i_blocks=1;//需要一个数据块来存放创建的目录，包括当前目录体和上一个目录体
        anode.i_size=dirsize*2;
        anode.i_atime=now;
        anode.i_ctime=now;
        anode.i_mtime=now;
        anode.i_dtime=0;
```

```
        memset(anode.i_block,-1,sizeof(anode.i_block));
        block_location=Find_Empty_block();//申请一个空的数据块
        anode.i_block[0]=block_location;
        for(int i=0;i<8;i++) anode.i_pad[i]=(char)0xff;//没有任何含义，只是为了
填充
        //表示当前目录，指向新创建的索引结点
        aentry.inode=node_location;
        aentry.rec_len= dirsize;
        aentry.name_len=1;
        aentry.file_type=2;
        strcpy(aentry.name,".");

 fseek(fp,data_begin_block*blocksize+block_location*blocksize,SEEK_SET);
        fwrite(&aentry,dirsize,1,fp);
        //表示上一级目录
        aentry.inode=bentry.inode;//指向当前目录索引结点(current)的"."目录体所在的
索引结点
        aentry.rec_len=dirsize;
        aentry.name_len=2;
        aentry.file_type=2;
        strcpy(aentry.name,"..");
        fwrite(&aentry,dirsize,1,fp);
        //一个空条目,用于对剩下的空间进行格式化
        aentry.inode=0;//表示为空
        aentry.rec_len=dirsize;
        aentry.name_len=0;
        aentry.file_type=0;//不知道是目录还是文件
        strcpy(aentry.name,"");
        fwrite(&aentry,dirsize,14,fp);//一个数据块512个字节，一个目录体32个字节，一
共有16个目录体，除去当前目录和上一级目录，还可以写14个目录体
    }
    //往索引结点表写入新的索引结点
    fseek(fp,inode_begin_block*blocksize+node_location*inodesize,SEEK_SET);
    fwrite(&anode,inodesize,1,fp);
    //往current索引结点指向的数据块写入目录体，表示是文件还是目录
    aentry.inode=node_location;
    aentry.rec_len=dirsize;
    aentry.name_len= strlen(name);
    if(type==1) aentry.file_type=1;
    else aentry.file_type=2;
    strcpy(aentry.name,name);
    fseek(fp, Find_empty_entry(current),SEEK_SET);
    fwrite(&aentry,dirsize,1,fp);
    //更新current索引结点
    fseek(fp,inode_begin_block*blocksize+bentry.inode*inodesize,SEEK_SET);
    fwrite(current,inodesize,1,fp);
    fclose(fp);
    return 0;
}
```

19. **delete**

    1. **文件打开：**

- `FILE *fp = NULL; while(fp == NULL) fp = fopen(PATH, "r+");`：在一个循环中，尝试打开文件系统的文件。如果文件指针 `fp` 为 `NULL`，则一直循环尝试打开文件，直到成功打开。

2. **查找目录体位置：**
   - `for (i = 0; i < current->i_size/dirsize; i++) { ... }`：遍历当前目录的目录项，检查是否存在同名的文件或目录。
   - `fread(&delete_entry, dirsize, 1, fp);`：读取目录体。
   - 如果找到目标文件或目录，记录其在当前目录中的相对位置。
3. **读取目标文件或目录的索引结点信息：**
   - `fseek(fp, inode_begin_block * blocksize + delete_entry.inode * inodesize, SEEK_SET);`：将文件指针移动到索引结点表中的目标索引结点的位置。
   - `fread(&delete_inode, inodesize, 1, fp);`：读取目标索引结点的信息。
4. **删除目录或文件：**
   - 如果是目录：
     - 检查目录是否为空，如果不为空，则输出错误信息；否则继续删除目录。
     - 调用 `Delete_block` 函数删除目录的数据块。
     - 调用 `Delete_inode` 函数删除目录的索引结点。
     - 更新当前目录的信息，包括减小 `i_size` 和删除相应的目录体。
   - 如果是文件：
     - 删除文件直接指向的块。
     - 删除一级索引中的数据块。
     - 删除二级索引中的数据块。
     - 调用 `Delete_inode` 函数删除文件的索引结点。
     - 更新当前目录的信息，包括减小 `i_size` 和删除相应的目录体。
5. **更新当前目录的信息：**
   - `fseek(fp, data_begin_block * blocksize + current->i_block[0] * blocksize, SEEK_SET);`：将文件指针移动到当前目录的数据块中的第一个目录体的位置。
   - `fread(&temp_entry, dirsize, 1, fp);`：读取当前目录体的信息。
   - `fseek(fp, inode_begin_block * blocksize + temp_entry.inode * inodesize, SEEK_SET);`：将文件指针移动到当前目录的索引结点在索引结点表中的位置。
   - `fwrite(current, inodesize, 1, fp);`：更新当前目录的索引结点信息。
6. **关闭文件：**
   - `fclose(fp);`：关闭文件。
7. **返回成功或失败：**
   - `return 0;`：表示删除成功。
   - `return 1;`：表示删除失败，通常是因为未找到目标文件或目录。

```
int Delete(int type,struct ext2_inode *current,char *name){
    FILE *fp=NULL;
    while(fp==NULL)
        fp= fopen(PATH,"r+");
    int dir_entry_location;//表示寻找到的目录体在当前目录中的相对位置
    struct ext2_inode delete_inode;
    struct ext2_dir_entry temp_entry,delete_entry,empty_entry;
    //一个空条目
```

```
    empty_entry.inode=0;
    empty_entry.rec_len=dirsize;
    empty_entry.name_len=0;
    strcpy(empty_entry.name, "");
    empty_entry.file_type=0;
    int i;
    for (i = 0; i < current->i_size/dirsize;i++) {
        fseek(fp, dir_entry_position(i*dirsize,current->i_block),SEEK_SET);
        fread(&delete_entry, dirsize, 1, fp);
        if((strcmp(delete_entry.name, name) == 0) && (delete_entry.file_type
== type))
        {
            dir_entry_location=i;
            break;
        }
    }
    if(i<current->i_size/dirsize)
    {

 fseek(fp,inode_begin_block*blocksize+delete_entry.inode*inodesize,SEEK_SET)
;
        fread(&delete_inode,inodesize,1,fp);
        //如果要删除的是文件夹
        if(type==2)
        {
            if(delete_inode.i_size>2*dirsize)
                printf("The folder is not empty!\n");
            else
            {
                Delete_block(delete_inode.i_block[0]);
                Delete_inode(delete_entry.inode);
                current->i_size=current->i_size-dirsize;//从目录的数据块中删去一
个目录体
                int last_entry_location= dir_entry_position(current-
>i_size,current->i_block);
                fseek(fp,last_entry_location,SEEK_SET);//找到当前目录的最后一个
条目
                fread(&temp_entry,dirsize,1,fp);//读取最后一个条目
                fseek(fp,last_entry_location,SEEK_SET);//找到当前目录的最后一个
条目
                fwrite(&empty_entry,dirsize,1,fp);//清空最后一个条目
                last_entry_location-=data_begin_block*blocksize;//在数据块中的
绝对位置
                if(last_entry_location%blocksize==0)//如果最后一个条目刚好是一个
数据块的起始位置，删除这个数据块
                {
                    Delete_block(last_entry_location/blocksize);
                    current->i_blocks--;
                    if(current->i_blocks==6)//正好是一级子索引的第一个指针
                        Delete_block(current->i_block[6]);
                    else if(current->i_blocks==128+6)//正好是二级子索引的第一个指
针
                    {
                        fseek(fp,data_begin_block*blocksize+current-
>i_block[7]*blocksize,SEEK_SET);
```

```
                        int block_location;
                        fread(&block_location,sizeof(u_int32_t),1,fp);
                        Delete_block(block_location);
                        Delete_block(current->i_block[7]);
                    }
                    else if((current->i_blocks-128-6)%128==0)
                    {
                        fseek(fp,data_begin_block*blocksize+current-
>i_block[7]*blocksize+((current->i_blocks-128-
6)/128)*sizeof(u_int32_t),SEEK_SET);
                        int block_location;
                        fread(&block_location,sizeof(u_int32_t),1,fp);
                        Delete_block(block_location);
                    }
                }
                if(dir_entry_location*dirsize<current->i_size)//如果删除的条目
不是最后一条，直接用最后一个条目覆盖,如果删除的是最后一个，也不用覆盖了
                {
                    fseek(fp,
dir_entry_position(dir_entry_location*dirsize,current->i_block),SEEK_SET);
                    fwrite(&temp_entry,dirsize,1,fp);
                }
                printf("The %s is deleted!\n",name);
            }
        }
        //删除文件
        else
        {
            //删除直接指向的块
            for(int j=0;j<6;j++)
            {
                if(delete_inode.i_blocks==0)
                    break;
                Delete_block(delete_inode.i_block[j]);
                delete_inode.i_blocks--;
            }
            //删除一级索引中的数据块
            if(delete_inode.i_blocks>0)
            {
                int block_location;

 fseek(fp,data_begin_block*blocksize+delete_inode.i_block[6]*blocksize,SEEK_
SET);
                for(int j=0;j<128;j++)
                {
                    if(delete_inode.i_blocks==0)
                        break;
                    fread(&block_location,sizeof(u_int32_t),1,fp);
                    Delete_block(block_location);
                    delete_inode.i_blocks--;
                }
                Delete_block(delete_inode.i_block[6]);//删除一级索引
            }
            //二级索引
            if(delete_inode.i_blocks>0)
```

```
            {
                int block_location_first;
                for(int j=0;j<128;j++)
                {

 fseek(fp,data_begin_block*blocksize+delete_inode.i_block[7]*blocksize+j*siz
eof(u_int32_t),SEEK_SET);
                    fread(&block_location_first, sizeof(u_int32_t), 1, fp);
                    fseek(fp,data_begin_block*blocksize +
block_location_first * blocksize, SEEK_SET);
                    int block_location_second;
                    for(int k=0;k<128;k++)
                    {
                        if(delete_inode.i_blocks==0)
                            break;

 fread(&block_location_second,sizeof(u_int32_t),1,fp);
                        Delete_block(block_location_second);
                        delete_inode.i_blocks--;
                    }
                    Delete_block(block_location_first);
                    if(delete_inode.i_blocks==0)
                        break;
                }
                Delete_block(delete_inode.i_block[7]);
            }
            Delete_inode(delete_entry.inode);
            int last_entry_location;
            current->i_size=current->i_size-dirsize;
            last_entry_location= dir_entry_position(current->i_size,current-
>i_block);//找到最后的条目
            fseek(fp,last_entry_location,SEEK_SET);
            fread(&temp_entry,dirsize,1,fp);
            fseek(fp,last_entry_location,SEEK_SET);
            fwrite(&empty_entry,dirsize,1,fp);
            last_entry_location-=data_begin_block*blocksize;
            if(last_entry_location%blocksize==0)//如果最后一个条目刚好是一个数据块
的起始位置，删除这个数据块
            {
                Delete_block(last_entry_location/blocksize);
                current->i_blocks--;
                if(current->i_blocks==6)//正好是一级子索引的第一个指针
                    Delete_block(current->i_block[6]);
                else if(current->i_blocks==128+6)//正好是二级子索引的第一个指针
                {
                    fseek(fp,data_begin_block*blocksize+current-
>i_block[7]*blocksize,SEEK_SET);
                    int block_location;
                    fread(&block_location,sizeof(u_int32_t),1,fp);
                    Delete_block(block_location);
                    Delete_block(current->i_block[7]);
                }
                else if((current->i_blocks-128-6)%128==0)
                {
```

```
                    fseek(fp,data_begin_block*blocksize+current-
>i_block[7]*blocksize+((current->i_blocks-128-
6)/128)*sizeof(u_int32_t),SEEK_SET);
                    int block_location;
                    fread(&block_location,sizeof(u_int32_t),1,fp);
                    Delete_block(block_location);
                }
            }
            if(dir_entry_location*dirsize<current->i_size)//如果删除的条目不是最
后一条，直接用最后一个条目覆盖,如果删除的是最后一个，也不用覆盖了
            {
                fseek(fp,
dir_entry_position(dir_entry_location*dirsize,current->i_block),SEEK_SET);
                fwrite(&temp_entry,dirsize,1,fp);
            }
        }
        fseek(fp,data_begin_block*blocksize+current-
>i_block[0]*blocksize,SEEK_SET);
        fread(&temp_entry,dirsize,1,fp);//表示当前目录

 fseek(fp,inode_begin_block*blocksize+temp_entry.inode*inodesize,SEEK_SET);
        fwrite(current,inodesize,1,fp);//更新current信息
    }
    else
    {
        fclose(fp);
        return 1;
    }
    fclose(fp);
    return 0;
}
```

20. **read**
    1. **文件打开**：
        - `FILE *fp = NULL; while(fp == NULL) fp = fopen(PATH, "r+");`：在一个循环中，尝试打开文件系统的文件。如果文件指针 `fp` 为 `NULL`，则一直循环尝试打开文件，直到成功打开。
    2. **查找目标文件**：
        - `for (int i = 0; i < (current->i_size/32); i++) { ... }`：遍历当前目录的目录项，查找目标文件。
        - `fread(&dir, sizeof(struct ext2_dir_entry), 1, fp);`：读取目录项。
        - 如果找到目标文件，检查其类型是否为文件。
    3. **读取文件内容**：
        - `fseek(fp, inode_begin_block * blocksize + dir.inode * inodesize, SEEK_SET);`：将文件指针移动到目标文件的索引结点位置。
        - `fread(&node, sizeof(struct ext2_inode), 1, fp);`：读取目标文件的索引结点信息。
        - 遍历文件的数据块，读取文件内容，并根据需要打印到屏幕上。
    4. **更新访问时间**：
        - `time_t now; time(&now);`：获取当前时间。

- `node.i_atime = now;`：更新目标文件的访问时间。
- `fseek(fp, inode_begin_block * blocksize + dir.inode * inodesize, SEEK_SET);`：将文件指针移动到目标文件的索引结点位置。
- `fwrite(&node, inodesize, 1, fp);`：将更新后的索引结点信息写回磁盘。

5. **关闭文件：**
   - `fclose(fp);`：关闭文件。
6. **返回成功或失败：**
   - `return 0;`：表示读取文件成功。
   - `return 1;`：表示读取文件失败.

```
int Read(struct ext2_inode *current,char *name){
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    for(int i=0;i<(current->i_size/32);i++){//其实和Open函数很像，只是要的是文件
        fseek(fp, dir_entry_position(i*32,current->i_block),SEEK_SET);
        fread(&dir,sizeof(struct ext2_dir_entry),1,fp);
        if(strcmp(dir.name,name)==0){
            if(dir.file_type==1){
                time_t now;
                struct ext2_inode node;
                char content;

 fseek(fp,inode_begin_block*blocksize+dir.inode*inodesize,SEEK_SET);
                fread(&node,sizeof(struct ext2_inode),1,fp);
                for(int j=0;j<node.i_size;j++){
                    fseek(fp,
dir_entry_position(j,node.i_block),SEEK_SET);//每次都找一个字节，我感觉这样的效率不高，可以优化一下
                    fread(&content,sizeof(char),1,fp);
                    if(content==0x0D)//0x0D（asc码是13） 指的是“回车”   \r是把光标置于本行行首
                        printf("\n");
                    else
                        printf("%c",content);
                }
                printf("\n");
                time(&now);
                node.i_atime=now;

 fseek(fp,inode_begin_block*blocksize+dir.inode*inodesize,SEEK_SET);
                fwrite(&node, inodesize,1,fp);//更新原来的索引结点
                fclose(fp);
                return 0;
            }
        }
    }
    fclose(fp);
    return 1;
}
```

21. **write**

1. **文件打开：**
   - `FILE *fp = NULL; while(fp == NULL) fp = fopen(PATH, "r+");`：在一个循环中，尝试打开文件系统的文件。如果文件指针 `fp` 为 `NULL`，则一直循环尝试打开文件，直到成功打开。
2. **查找目标文件：**
   - `for (int i = 0; i < (current->i_size/32); i++) { ... }`：遍历当前目录的目录项，查找目标文件。
   - `fread(&dirEntry, dirsize, 1, fp);`：读取目录项。
   - 如果找到目标文件，检查其类型是否为文件。
3. **获取输入字符并写入文件：**
   - `str = getch();`：获取用户输入的字符。
   - `while(str != 27) { ... }`：在一个循环中，检查用户输入的字符是否为 ASCII 码为 27（ESC键，表示退出写入）。
4. **判断是否需要申请新的数据块：**
   - `(node.i_size % blocksize == 0) || (node.i_size == 0)`：判断是否当前数据块已满，或者刚开始需要一个数据块。
   - `Add_block(&node, (node.i_size) / blocksize);`：如果需要，调用 `Add_block` 函数来为文件申请新的数据块。
   - `node.i_blocks++;`：更新文件的数据块数。
5. **写入字符到文件：**
   - `fseek(fp, dir_entry_position(node.i_size, node.i_block), SEEK_SET);`：将文件指针移动到文件数据块的末尾。
   - `fwrite(&str, sizeof(char), 1, fp);`：将字符写入文件。
   - `node.i_size = node.i_size + sizeof(char);`：更新文件大小。
   - 如果字符为回车符（`0x0d`），在屏幕上打印换行符。
6. **更新文件的修改时间和访问时间：**
   - `time_t now; time(&now);`：获取当前时间。
   - `node.i_mtime = now; node.i_atime = now;`：更新文件的修改时间和访问时间。
7. **更新文件索引结点信息：**
   - `fseek(fp, inode_begin_block * blocksize + dirEntry.inode * inodesize, SEEK_SET);`：将文件指针移动到文件的索引结点位置。
   - `fwrite(&node, inodesize, 1, fp);`：将更新后的索引结点信息写回磁盘。
8. **关闭文件：**
   - `fclose(fp);`：关闭文件。
9. **返回成功或失败：**
   - `return 0;`：表示写入文件成功。
   - `return 1;`：表示写入文件失败.

```
int Write(struct ext2_inode *current,char *name){
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    struct ext2_dir_entry dirEntry;
    struct ext2_inode node;
    time_t now;
    char str;
    while(1){
        int i;
```

```
        for(i=0;i<(current->i_size/32);i++){//遍历所有的目录体
            fseek(fp, dir_entry_position(i*dirsize,current-
>i_block),SEEK_SET);
            fread(&dirEntry,dirsize,1,fp);
            if(strcmp(dirEntry.name,name)==0){
                if(dirEntry.file_type==1){//是文件类型

 fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
                    fread(&node,inodesize,1,fp);
                    break;
                }
            }
        }
        if(i<(current->i_size/32))//有文件存在
            break;
        printf("There isn't this file,please create it first\n");
        return 1;
    }
    str=getch();
    while(str!=27){
        printf("%c",str);
        if((node.i_size%blocksize==0)||(node.i_size==0)){//写入的数据占满一个数
据块了，需要申请新的数据块;或者刚开始需要一个数据块
            Add_block(&node,(node.i_size)/blocksize);
            node.i_blocks++;
        }
        fseek(fp, dir_entry_position(node.i_size,node.i_block),SEEK_SET);
        fwrite(&str,sizeof(char),1,fp);
        node.i_size=node.i_size+sizeof(char);
        if(str==0x0d) printf("%c",0x0a);
        str=getch();
        if(str==27) break;
    }
    time(&now);
    node.i_mtime=now;
    node.i_atime=now;
    fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
    fwrite(&node,inodesize,1,fp);
    fclose(fp);
    printf("\n");
    return 0;
}
```

22. **ls**

    1. **文件打开：**
        - `FILE *fp = NULL; while(fp == NULL) fp = fopen(PATH, "r+");`：在一个循环中，尝试打开文件系统的文件。如果文件指针 `fp` 为 `NULL`，则一直循环尝试打开文件，直到成功打开。
    2. **遍历目录项并获取文件信息：**
        - `for (int i = 0; i < (current->i_size/dirsize); i++) { ... }`：遍历当前目录的目录项。
        - `fread(&dirEntry, dirsize, 1, fp);`：读取目录项。

- `fseek(fp, inode_begin_block * blocksize + dirEntry.inode * inodesize, SEEK_SET);`：将文件指针移动到文件的索引结点位置。
- `fread(&node, inodesize, 1, fp);`：读取文件的索引结点信息。

3. **格式化时间字符串：**
   - `char timestr[150];`：创建一个字符串数组来存储格式化后的时间字符串。
   - `strcat(timestr, asctime(localtime((const time_t *) &node.i_ctime)));`：使用 `asctime` 函数将创建时间追加到时间字符串。
   - `strcat(timestr, asctime(localtime((const time_t *) &node.i_atime)));`：将访问时间追加到时间字符串。
   - `strcat(timestr, asctime(localtime((const time_t *) &node.i_mtime)));`：将修改时间追加到时间字符串。
   - `for(int j=0;j<strlen(timestr)-1;j++){ if(timestr[j]=='\n') timestr[j]='\t'; }`：将时间字符串中的换行符替换为制表符。
4. **打印文件信息：**
   - 根据文件类型（文件或目录）打印相应的信息。
   - 包括文件和目录的权限

```
void Ls(struct ext2_inode *current){
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    struct ext2_dir_entry dirEntry;
    struct ext2_inode node;
    char time[150];

 printf("Type\t\tAuthorization\t\tFileName\tCreateTime\t\t\tLastAccessTime\t
\t\tModifyTime\n");
    for(int i=0;i<(current->i_size/dirsize);i++){
        fseek(fp, dir_entry_position(i*dirsize,current->i_block),SEEK_SET);
        fread(&dirEntry,dirsize,1,fp);

 fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
        fread(&node,inodesize,1,fp);
        strcpy(time,"");
        strcat(time, asctime(localtime((const time_t *) &node.i_ctime)));
        strcat(time, asctime(localtime((const time_t *) &node.i_atime)));
        strcat(time, asctime(localtime((const time_t *) &node.i_mtime)));
        for(int j=0;j<strlen(time)-1;j++){
            if(time[j]=='\n')
                time[j]='\t';
        }
        //uint16_t j=0x04;
        char ch[]={'d','r','w','x','-','-','-','-','-','-','\0'};
        if(node.i_mode==262||node.i_mode==518) {
            ch[3]='-';
        }
        else if(node.i_mode==261||node.i_mode==517){
            ch[2]='-';
        }
        else if(node.i_mode==260||node.i_mode==516){
            ch[3]='-';
            ch[2]='-';
        }
```

```
        else if(node.i_mode==259||node.i_mode==515){
            ch[1]='-';
        }
        else if(node.i_mode==258||node.i_mode==514){
            ch[1]='-';
            ch[3]='-';
        }
        else if(node.i_mode==257||node.i_mode==513){
            ch[1]='-';
            ch[2]='-';
        }
        //printf("%d\n",node.i_mode);
        if(dirEntry.file_type==1) {
            ch[0]='-';
            printf("File\t\t%s\t\t%s\t\t%s",ch,dirEntry.name,time);
        }
        else printf("Directory\t%s\t\t%s\t\t%s",ch,dirEntry.name,time);
    }
    fclose(fp);
}
```

23. **close**

实现了将当前目录的访问时间更新为当前时间，然后通过打开上一级目录实现了关闭当前目录的功能。在文件系统中，通常会通过更新访问时间等信息来维护文件和目录的状态。这种关闭目录的操作可能会用于用户在文件系统中导航时，返回到上一级目录。

```
int Close(struct ext2_inode *current){
    time_t now;
    struct ext2_dir_entry bentry;
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    time(&now);
    current->i_atime=now;
    fseek(fp, dir_entry_position(0,current->i_block),SEEK_SET);//寻找当前目录索引结点指向的第一个数据块中的第一个目录体
    fread(&bentry,sizeof(struct ext2_dir_entry),1,fp);//第一个目录体就是当前目录".",里面存放了当前目录索引结点的位置
    fseek(fp,inode_begin_block*blocksize+bentry.inode*sizeof(struct ext2_inode),SEEK_SET);//寻找当前目录所在的索引结点在硬盘的位置
    fwrite(current,sizeof(struct ext2_inode),1,fp);
    fclose(fp);
    return Open(current,"..");//打开上一级目录
}
```

24. **chmod：**

实现了对文件或目录访问权限的修改；

```
int chmod(struct ext2_inode *current,char *name,char *auth){
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    struct ext2_dir_entry dirEntry;
```

```
    struct ext2_inode node;
    time_t now;
    int i;
    for(i=0;i<(current->i_size/dirsize);i++){
        fseek(fp, dir_entry_position(i*dirsize,current->i_block),SEEK_SET);
        fread(&dirEntry,dirsize,1,fp);
        //
fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
        // fread(&node,inodesize,1,fp);
        if(strcmp(dirEntry.name,name)==0){
            //if(dirEntry.file_type==1){//是文件类型

 fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
            fread(&node,inodesize,1,fp);
            break;
        }
    }
    if(i>=(current->i_size/32)) return 1;
    uint16_t j=0x00;

    if(auth[1]=='r') j=j|0b0000000000000100;
    if(auth[2]=='w') j=j|0b0000000000000010;
    if(auth[3]=='x') j=j|0b0000000000000001;

    if(dirEntry.file_type==1){
        j=j|0b0000000100000000;
        node.i_mode=j;
    }
    else{
        j=j|0b0000001000000000;
        node.i_mode=j;
    }
    //printf("%d\n",node.i_mode);
    time(&now);
    node.i_mtime=now;
    node.i_atime=now;
    fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);//更
新索引结点信息
    fwrite(&node,inodesize,1,fp);
    printf("The authorization has been modified!\n");
    fclose(fp);
    return 0;
}
```

底层函数设计

24. **open**用于传入文件名和当前索引结点指针判断目录中是否存在以此为名的文件；

实现了打开目录的功能，通过输入目录名称（`name`），在当前目录下查找该目录，并将目录的`ext2_inode`结构体信息存储到传入的`struct ext2_inode *current`中；

如果遍历完所有目录项都没有找到匹配的目录，或者找到的目录不是目录类型，则在关闭文件之前返回1表示打开目录失败。

- `fclose(fp);`：关闭文件。
- `return 1;`：返回1表示打开目录失败。返回0则说明打开成功；

```
int Open(struct ext2_inode *current,char *name){
    FILE *fp=NULL;
    while(fp==NULL){
        fp= fopen(PATH,"r+");
    }
    for(int i=0;i<(current->i_size/32);i++) {//一个目录数据结构的大小为32个字节，计算一共有多少个目录体
        fseek(fp, dir_entry_position(i*32,current->i_block),SEEK_SET);//定位当前目录索引结点指向的数据块中第i个目录体
        fread(&dir,sizeof(struct ext2_dir_entry),1,fp);
        if(strcmp(dir.name,name)==0){
            if(dir.file_type==2){//查找类型为目录的文件
                fseek(fp,inode_begin_block*blocksize+dir.inode*sizeof(struct ext2_inode),SEEK_SET);//从第一个索引结点出发，定位该目录所在的索引结点
                fread(current,sizeof(struct ext2_inode),1,fp);
                fclose(fp);
                return 0;//表示成功找到
            }
        }
    }
    fclose(fp);
    return 1;//表示查找失败
}
```

25. get_dir_name

获取给定目录的名称，即通过传入的 `struct ext2_inode` 结构体 `node` 找到该目录的上一级目录，并获取该目录在上一级目录中的名称。

```
void get_dir_name(char *name,struct ext2_inode node){
    struct ext2_dir_entry entry;
    struct ext2_inode current=node;
    u_int16_t i_node;
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    //current指向上一个目录
    Open(&current,"..");
    //在当前目录中搜索 "."目录体，记录所在的索引号
    for(int i=0;i<node.i_size/32;i++){
        fseek(fp, dir_entry_position(i*dirsize,node.i_block),SEEK_SET);
        fread(&entry,dirsize,1,fp);
        if(strcmp(entry.name,".")==0){
            i_node=entry.inode;
            break;
        }
    }
    //在上一级目录搜索索引号为i_node的目录体，获得当前目录的名字
```

```
    for(int i=0;i<current.i_size/32;i++){
        fseek(fp, dir_entry_position(i*dirsize,current.i_block),SEEK_SET);
        fread(&entry,dirsize,1,fp);
        if(entry.inode==i_node){
            strcpy(name,entry.name);
            return;
        }
    }
}
```

26. dir_entry_position

此函数接收目录项在文件系统中的相对位置 `dir_entry_begin` 和索引结点的直接或间接索引块数组 `i_block` 作为参数，并返回目录项在文件系统中的绝对位置。在这个函数中，通过不同的情况处理，计算出目录项在文件系统中的准确位置。

```
int dir_entry_position(int dir_entry_begin, u_int16_t i_block[8]) {
    int dir_blocks = dir_entry_begin / 512; // 一个数据块需要512个字节，计算目录项位于哪个数据块
    int block_offset = dir_entry_begin % 512; // 计算在数据块内的偏移量

    FILE *fp = NULL;
    int dir_entry_position;

    // 前六个直接索引块，第七个一级索引，第八个二级索引
    if (dir_blocks <= 5) {
        dir_entry_position = data_begin_block * blocksize + i_block[dir_blocks] * blocksize + block_offset;
        // 在数据块开始位置向后移动直接索引的数据块所在位置，在向后移动数据块内的偏移量
    } else {
        while (fp == NULL) fp = fopen(PATH, "r+");

        dir_blocks = dir_blocks - 6; // 超出了直接索引的范围，需要去一级子索引寻找
        if (dir_blocks < 128) {
            // 一级子索引最多能指向(512/4)=128个数据块，因为指针是int(4字节)
            // 一级子索引只用一个数据块(512字节)，整个数据块中存放了128个指向其他数据块的指针
            int position;
            fseek(fp, data_begin_block * blocksize + i_block[6] * blocksize + dir_blocks * 4, SEEK_SET);
            // 寻找一级子索引中指向所需数据块的指针的位置
            fread(&position, sizeof(int), 1, fp);
            dir_entry_position = data_begin_block * blocksize + position * blocksize + block_offset;
        } else {
            // 超出一级子索引的范围，需要减去一级子索引的范围，二级子索引最多可寻找128*128个数据块
            int position_first;
            int position_second;
            dir_blocks = dir_blocks - 128;

            fseek(fp, data_begin_block * blocksize + i_block[7] * blocksize + (dir_blocks / 128) * 4, SEEK_SET);
```

```
            fread(&position_first, sizeof(int), 1, fp);

            fseek(fp, data_begin_block * blocksize + position_first * blocksize
+ (dir_blocks % 128) * 4, SEEK_SET);
            fread(&position_second, sizeof(int), 1, fp);

            dir_entry_position = data_begin_block * blocksize + position_second
* blocksize + block_offset;
        }
        fclose(fp);
    }

    return dir_entry_position;
}
```

27. **ext2_new_inode**

此函数通过搜索 inode 位图找到第一个未分配的索引结点，更新相关的位图和描述符，并返回该索引结点的位置。

```
u_int32_t ext2_new_inode() {
    FILE *fp = NULL;
    u_int32_t zero[blocksize / 4];

    while (fp == NULL) fp = fopen(PATH, "r+");

    fseek(fp, 2 * blocksize, SEEK_SET); // 索引结点位图
    fread(zero, blocksize, 1, fp); // zero 中存储索引结点位图

    for (u_int32_t i = last_allco_inode; i < (last_allco_inode + blocksize / 4);
i++) {
        // 从上一个分配的结点开始搜索，并且会循环搜索，最后一个找不到会重新回到第一个元素开始搜
索
        if (zero[i % (blocksize / 4)] != 0xffffffff) {
            u_int32_t j = 0x80000000, k = zero[i % (blocksize / 4)];

            // 找到第一个空的索引结点
            for (u_int32_t p = 0; p < 32; p++) {
                if ((j & k) == 0x00000000) {
                    // 如果为0，表示偏移量为 p 的索引结点是空的
                    zero[i % (blocksize / 4)] = zero[i % blocksize / 4] | j; //
更新 inode 位图，添加一个 1
                    groupDesc.bg_free_inodes_count--; // 可分配的索引结点减一

                    fseek(fp, 0, SEEK_SET);
                    fwrite(&groupDesc, sizeof(struct ext2_group_desc), 1, fp);
// 更新组描述符

                    fseek(fp, 2 * blocksize, SEEK_SET);
                    fwrite(zero, blocksize, 1, fp); // 更新 inode 位图

                    last_allco_inode = i % (blocksize / 4);
```

```
                    fclose(fp);
                    return (i % (blocksize / 4)) * 32 + p; // 返回索引结点的位置
                } else {
                    j = j / 2; // 算术向右移位，唯一的1向右移位一次
                }
            }
        }
    }

    fclose(fp);
    return -1; // 没有找到空的索引结点
}
```

28. **ext2_alloc_block**

用于查找并返回一个未分配的数据块的位置;

```
u_int32_t ext2_alloc_block() {
    FILE *fp = NULL;
    u_int32_t zero[blocksize / 4];

    while (fp == NULL) fp = fopen(PATH, "r+");

    fseek(fp, 1 * blocksize, SEEK_SET);
    fread(zero, blocksize, 1, fp);

    for (u_int32_t i = last_allco_block; i < (last_allco_block + blocksize / 4);
i++) {
        // 从上一个分配的结点开始搜索，并且会循环搜索，最后一个找不到会重新回到第一个元素开始搜
索
        if (zero[i % (blocksize / 4)] != 0xffffffff) {
            u_int32_t j = 0x80000000, k = zero[i % (blocksize / 4)];

            // 找到第一个空的数据块
            for (u_int32_t p = 0; p < 32; p++) {
                if ((j & k) == 0x00000000) {
                    // 如果为0，表示偏移量为 p 的数据块是空的
                    zero[i % (blocksize / 4)] = zero[i % blocksize / 4] | j; //
更新 block 位图，添加一个 1
                    groupDesc.bg_free_blocks_count--; // 可分配的数据块减一

                    fseek(fp, 0, SEEK_SET);
                    fwrite(&groupDesc, sizeof(struct ext2_group_desc), 1, fp);
// 更新组描述符

                    fseek(fp, 1 * blocksize, SEEK_SET);
                    fwrite(zero, blocksize, 1, fp); // 更新 block 位图

                    last_allco_block = i % (blocksize / 4);
                    fclose(fp);
                    return (i % (blocksize / 4)) * 32 + p; // 返回数据块的位置
                } else {
                    j = j / 2; // 算术向右移位，唯一的 1 向右移位一次
```

```
                }
            }
        }
    }

    fclose(fp);
    return -1; // 没有找到空的数据块
}
```

29. **ext2_free_inode**

用于删除指定位置的索引结点;

```
void ext2_free_inode(int position) {
    u_int32_t zero[blocksize / 4];
    FILE *fp = NULL;

    while (fp == NULL) fp = fopen(PATH, "r+");

    fseek(fp, 2 * blocksize, SEEK_SET);
    fread(zero, blocksize, 1, fp);

    u_int32_t j = 0x80000000;

    // 定位到偏移量为 position%32 的位置
    for (int i = 0; i < position % 32; i++) j = j / 2;

    zero[position / 32] = zero[position / 32] ^ j; // 异或操作，会将对应的 1 置 0，其他位不受影响

    fseek(fp, 2 * blocksize, SEEK_SET);
    fwrite(zero, blocksize, 1, fp);

    fclose(fp);
}
```

30. **ext2_free_blocks**

用于删除指定位置的数据块;

```
void ext2_free_blocks(int position) {
    u_int32_t zero[blocksize / 4];
    FILE *fp = NULL;

    while (fp == NULL) fp = fopen(PATH, "r+");

    fseek(fp, 1 * blocksize, SEEK_SET);
    fread(zero, blocksize, 1, fp);
```

```
    u_int32_t j = 0x80000000;

    // 定位到偏移量为 position%32 的位置
    for (int i = 0; i < position % 32; i++) j = j / 2;

    zero[position / 32] = zero[position / 32] ^ j; // 异或操作，会将对应的 1 置 0，其他位不受影响

    fseek(fp, 1 * blocksize, SEEK_SET);
    fwrite(zero, blocksize, 1, fp);

    fclose(fp);
}
```

31. **ext2_empty_dir**

查找给定索引结点（`current`）中的下一个空目录项的位置;

```
int ext2_empty_dir(struct ext2_inode *current) {
    FILE *fp = NULL;

    while (fp == NULL) fp = fopen(PATH, "r+");

    int location;  // 目录体的绝对地址
    u_int32_t block_location;
    int temp = blocksize / sizeof(u_int32_t);  // 每个block可以存放的指针数量

    if (current->i_size % blocksize == 0) {
        // 当前的文件大小正好占用了所有可用的数据块，因为目录体占用32个字节，
        // 一个数据块最多存放512/32=16个目录体，超出就要申请新的数据块
        Add_block(current, current->i_blocks);  // 申请一个新的数据块
        current->i_blocks++;
    }

    if ((current->i_blocks - 1) < 6) {
        // 前六个直接索引
        location = data_begin_block * blocksize + current->i_block[current->i_blocks - 1] * blocksize + current->i_size % blocksize;
    }
    else if ((current->i_blocks - 1 - 6) < 128) {
        // 一级索引
        fseek(fp, data_begin_block * blocksize + current->i_block[6] * blocksize + (current->i_blocks - 1 - 6) * sizeof(u_int32_t), SEEK_SET);
        fread(&block_location, sizeof(u_int32_t), 1, fp);
        location = data_begin_block * blocksize + block_location * blocksize + current->i_size % blocksize;
    }
    else {
        // 二级索引
        int remain_block;
        remain_block = current->i_blocks - 1 - 6 - 128;
```

```
        // 寻找第一级索引指针的位置
        fseek(fp, data_begin_block * blocksize + current->i_block[7] * blocksize
+ (remain_block / 128) * sizeof(u_int32_t), SEEK_SET);
        fread(&block_location, sizeof(u_int32_t), 1, fp);

        remain_block = remain_block % 128;  // 求偏移量
        fseek(fp, data_begin_block * blocksize + block_location * blocksize +
remain_block * sizeof(u_int32_t), SEEK_SET);
        fread(&block_location, sizeof(u_int32_t), 1, fp);

        location = data_begin_block * blocksize + block_location * blocksize +
current->i_size % blocksize;
    }

    current->i_size = current->i_size + dirsize;  // 文件大小加一个目录体的大小
    fclose(fp);
    return location;
}
```

32. **Add_block**

用于为给定的索引结点（`current`）分配一个数据块

```
void Add_block(struct ext2_inode *current, int block_number) {
    FILE *fp = NULL;

    while (fp == NULL) fp = fopen(PATH, "r+");

    // 直接索引
    if (block_number < 6) {
        current->i_block[block_number] = Find_Empty_block();
    }
    else {
        block_number = block_number - 6;

        if (block_number == 0) {
            // 为一级子索引找一个空的数据块来存放指针，最多可以存放512/4=128个
            current->i_block[6] = Find_Empty_block();

            fseek(fp, data_begin_block * blocksize + current->i_block[6] *
blocksize, SEEK_SET);
            u_int32_t empty_block = Find_Empty_block();
            fwrite(&empty_block, sizeof(u_int32_t), 1, fp); // 往数据块里写入第一个
指针
        }
        else if (block_number < 128) {
            // 在一级子索引指向的第一个数据块写指针
            u_int32_t empty_block = Find_Empty_block();

            fseek(fp, data_begin_block * blocksize + current->i_block[6] *
blocksize + block_number * sizeof(u_int32_t), SEEK_SET);
            fwrite(&empty_block, sizeof(u_int32_t), 1, fp);
        }
```

```
        else {
            block_number = block_number - 128;

            if (block_number == 0) {
                // 写入二级索引的第一个数据块指针
                current->i_block[7] = Find_Empty_block();

                fseek(fp, data_begin_block * blocksize + current->i_block[7] *
blocksize, SEEK_SET);
                u_int32_t empty_block = Find_Empty_block();
                fwrite(&empty_block, sizeof(u_int32_t), 1, fp); // 往数据块写入第一
个第二级指针，一共可以写入128个第二级指针

                fseek(fp, data_begin_block * blocksize + empty_block *
blocksize, SEEK_SET);
                empty_block = Find_Empty_block();
                fwrite(&empty_block, sizeof(u_int32_t), 1, fp); // 第一级索引的第一
个指针->第一个二级索引->第一个数据块指针
            }
            else if (block_number % 128 == 0) {
                // 需要更新一个二级索引，顺便放一个数据块指针
                fseek(fp, data_begin_block * blocksize + current->i_block[7] *
blocksize + (block_number / 128) * 4, SEEK_SET);
                u_int32_t empty_block = Find_Empty_block();
                fwrite(&empty_block, sizeof(u_int32_t), 1, fp); // 写入一个二级索引
指针

                fseek(fp, data_begin_block * blocksize + empty_block *
blocksize, SEEK_SET);
                empty_block = Find_Empty_block();
                fwrite(&empty_block, sizeof(u_int32_t), 1, fp); // 往数据块写入一个
数据块指针
            }
            else {
                // 往二级索引的地方写入新的数据块指针
                fseek(fp, data_begin_block * blocksize + current->i_block[7] *
blocksize + (block_number / 128) * 4, SEEK_SET);
                u_int32_t block_read;
                fread(&block_read, sizeof(u_int32_t), 1, fp);

                fseek(fp, data_begin_block * blocksize + block_read * blocksize
+ (block_number % 128) * 4, SEEK_SET);
                u_int32_t empty_block = Find_Empty_block();
                fwrite(&empty_block, sizeof(u_int32_t), 1, fp);
            }
        }
    }

    fclose(fp);
}
```

33. **check_auth:**

读写文件时，检查文件权限是否可行；

```
int check_auth(struct ext2_inode *current,char *name){
    FILE *fp=NULL;
    while(fp==NULL) fp= fopen(PATH,"r+");
    struct ext2_dir_entry dirEntry;
    struct ext2_inode node;
    time_t now;
    int i;
    for(i=0;i<(current->i_size/dirsize);i++){
        fseek(fp, dir_entry_position(i*dirsize,current->i_block),SEEK_SET);
        fread(&dirEntry,dirsize,1,fp);
        //
fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
        // fread(&node,inodesize,1,fp);
        if(strcmp(dirEntry.name,name)==0){
            //if(dirEntry.file_type==1){//是文件类型

 fseek(fp,inode_begin_block*blocksize+dirEntry.inode*inodesize,SEEK_SET);
            fread(&node,inodesize,1,fp);
            break;
        }
    }
    //read-1
    //write-0
    //read+write-4
    //not found-3
    //none-2
    if(i>=(current->i_size/32)) return 3;//not found
    if(node.i_mode==263||node.i_mode==519){//rwx
        return 4;
    }
    else if(node.i_mode==262||node.i_mode==518){//rw-
        return 4;
    }
    else if(node.i_mode==261||node.i_mode==517){//r-x
        return 1;
    }
    else if(node.i_mode==260||node.i_mode==516){//r--
        return 1;
    }
    else if(node.i_mode==259||node.i_mode==515){//-wx
        return 0;
    }
    else if(node.i_mode==258||node.i_mode==514){//-w-
        return 0;
    }
    else if(node.i_mode==257||node.i_mode==513){//--x
        return 2;
    }
    fclose(fp);
}
```

运行结果

初始界面：

```
[root@kp-test01 test]# gcc -o ext2 ext2.c
[root@kp-test01 test]# ./ext2
Authorized users only. All activities may be monitored and reported.
Please input your password:123456



                    Welcome to Ext2 File System!



[wzy@ext2fs .]#
```

ls：其中含有以前创建的一些文件或目录

```
[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023
File            --wx------             lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 12:47:10 2023        Mon Nov 27 12:47:10 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023
```

cd：进入hhh目录中，并通过..返回上一级目录和/返回根目录

```
[wzy@ext2fs .]# cd hhh

[wzy@ext2fs hhh]# cd /

[wzy@ext2fs .]# cd hhh

[wzy@ext2fs hhh]# cd ..

[wzy@ext2fs .]#
```

create：

```
[wzy@ext2fs .]# create aaa
aaa is created successfully

[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023
File            --wx------             lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 12:47:10 2023        Mon Nov 27 12:47:10 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023
File            -rwx------             aaa             Mon Nov 27 14:14:03 2023        Mon Nov 27 14:14:03 2023        Mon Nov 27 14:14:03 2023
```

mkdir:

```
[wzy@ext2fs .]# mkdir bbb
bbb is created successfully

[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023
File            --wx------             lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 12:47:10 2023        Mon Nov 27 12:47:10 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023
File            -rwx------             aaa             Mon Nov 27 14:14:03 2023        Mon Nov 27 14:14:03 2023        Mon Nov 27 14:14:03 2023
Directory       drw-------              bbb             Mon Nov 27 14:15:11 2023        Mon Nov 27 14:15:11 2023        Mon Nov 27 14:15:11 2023
```

delete:

```
[wzy@ext2fs .]# delete aaa
aaa is deleted successfully

[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023
File            --wx------             lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 12:47:10 2023        Mon Nov 27 12:47:10 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023
Directory       drw-------              bbb             Mon Nov 27 14:15:11 2023        Mon Nov 27 14:15:11 2023        Mon Nov 27 14:15:11 2023
```

rmdir：

```
[wzy@ext2fs .]# rmdir bbb
The bbb is deleted!
bbb is deleted successfully

[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023
File            --wx------              lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 12:47:10 2023        Mon Nov 27 12:47:10 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023
```

close：关闭当前的目录

```
[wzy@ext2fs .]# cd hhh

[wzy@ext2fs hhh]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023        Mon Nov 27 12:41:58 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023

[wzy@ext2fs hhh]# close

[wzy@ext2fs .]#
```

password：

```
[wzy@ext2fs .]# password
Please input the old password
123456
Please input the new password(12 words max):
123456
Modify the password?[Y/N]y
```

read：访问控制，使用read读取lll内容，再用ls命令查看访问时间和修改时间；

修改访问权限以便可以成功访问。

```
[wzy@ext2fs .]# read lll
Sorry! Insufficient permissions.

[wzy@ext2fs .]# chmod lll -r--------
The authorization has been modified!

[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 14:19:10 2023        Mon Nov 27 12:41:58 2023
File            -r--------              lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 14:21:52 2023        Mon Nov 27 14:21:52 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023

[wzy@ext2fs .]# read lll
sfdsfd
fds
fd
```

write：

```
[wzy@ext2fs .]# write lll
Sorry! Insufficient permissions.

[wzy@ext2fs .]# chmod lll -rw-------
The authorization has been modified!

[wzy@ext2fs .]# ls
Type            Authorization           FileName        CreateTime                      LastAccessTime                  ModifyTime
Directory       drw-------              .               Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              ..              Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023        Mon Nov 27 12:05:25 2023
Directory       drw-------              hhh             Mon Nov 27 12:41:58 2023        Mon Nov 27 14:19:10 2023        Mon Nov 27 12:41:58 2023
File            -rw-------              lll             Mon Nov 27 12:42:20 2023        Mon Nov 27 14:23:33 2023        Mon Nov 27 14:23:33 2023
Directory       drw-------              kkk             Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023        Mon Nov 27 12:48:25 2023

[wzy@ext2fs .]# write lll
fefefesfesgggg
```

format：使用格式化命令，并使用ls命令查看是否format成功；

```
[wzy@ext2fs .]# mkdir kkk
kkk is created successfully

[wzy@ext2fs .]# ls
Type            Authorization          FileName        CreateTime                    LastAccessTime                ModifyTime
Directory       drw-------             .               Mon Nov 27 14:38:13 2023      Mon Nov 27 14:38:13 2023      Mon Nov 27 14:38:13 2023
Directory       drw-------             ..              Mon Nov 27 14:38:13 2023      Mon Nov 27 14:38:13 2023      Mon Nov 27 14:38:13 2023
Directory       drw-------             kkk             Mon Nov 27 14:40:56 2023      Mon Nov 27 14:40:56 2023      Mon Nov 27 14:40:56 2023

[wzy@ext2fs .]# format
FORMAT?
[Y/N]: y
Please input your password:123456
Format success!

[wzy@ext2fs .]# ls
Type            Authorization          FileName        CreateTime                    LastAccessTime                ModifyTime
Directory       drw-------             .               Mon Nov 27 14:41:15 2023      Mon Nov 27 14:41:15 2023      Mon Nov 27 14:41:15 2023
Directory       drw-------             ..              Mon Nov 27 14:41:15 2023      Mon Nov 27 14:41:15 2023      Mon Nov 27 14:41:15 2023

[wzy@ext2fs .]#
```

help:

```
[wzy@ext2fs .]# help
------------------------commands-------------------------

There are 14 commands you can choose as follows:
create: create [File name]
Create a file or directory whose name need to be provided.

delete: delete [File name]
Delete a file or directory whose name need to be provided.

cd: cd [Path]
Enter the directory of the given path.

close: close [none]
Close the current directory

read: read [File name]
Read the file content of the file name given.

write: write [File name]
Write content to the file with the given file name.

password: password [none]
Modify your password.

format: format [none]
Format this File System.

quit: quit [none]
Quit this File System.

ls: ls [none]
List all the contents under the current directory.

help: help [none]
Provide help.
mkdir: mkdir [File name]
Create a directory whose name need to be provided.

rmdir: rmdir [File name]
Delete a directory whose name need to be provided.
```

quit:

```
[wzy@ext2fs .]# quit
This File System is closed successfully!
[root@kp-test01 test]#
```