实验 3 文件系统

```
#include <stdio.h>
#include "string.h"
#include "stdlib.h"
#include "time.h"
#include <sys/ioctl.h>
#include <termios.h>
#include <unistd.h>            /* for STDIN_FILENO */
#define blocks 4611            // 1+1+1+512+4096,总块数
#define blocksiz 512           //每块字节数
#define inodesiz 64            //索引长度
#define data_begin_block 515   //数据开始块
#define dirsiz 32              //目录体长度
#define EXT2_NAME_LEN 15       //文件名长度
#define PATH "vdisk"           //文件系统
typedef struct ext2_group_desc //组描述符 68 字节
{
    char bg_volume_name[16];  //卷名
    int bg_block_bitmap;      //保存块位图的块号
    int bg_inode_bitmap;      //保存索引结点位图的块号
    int bg_inode_table;       //索引结点表的起始块号
    int bg_free_blocks_count; //本组空闲块的个数
    int bg_free_inodes_count; //本组空闲索引结点的个数
    int bg_used_dirs_count;   //本组目录的个数
    char psw[16];             //password
    char bg_pad[24]; //填充(0xff)
} ext2_group_desc;

typedef struct ext2_inode //索引节点 64 字节
```

```
{
    int i_mode;     //文件类型及访问权限 1:普通文件，2:目录
    int i_blocks;   //文件内容占用的数据块个数
    int i_size;     //大小(字节)
    time_t i_atime; //访问时间
    time_t i_ctime; //创建时间
    time_t i_mtime; //修改时间
    time_t i_dtime; //删除时间
    int i_block[8]; //指向数据块的指针
    char i_pad[24]; //填充 1(0xff)
} ext2_inode;

typedef struct ext2_dir_entry //目录体 32 字节
{
    int inode;                //索引节点号
    int rec_len;              //目录项长度
    int name_len;             //文件名长度
    int file_type;            //文件类型(1:普通文件，2:目录…)
    char name[EXT2_NAME_LEN]; //文件名
    char dir_pad;             //填充
} ext2_dir_entry;

/*定义全局变量*/
ext2_group_desc group_desc; //组描述符
ext2_inode inode;
ext2_dir_entry dir;                  //目录体（目录体，内容可能是文件也可能是目
录）

FILE *f;                             /*文件指针*/
unsigned int last_allco_inode = 0; //上次分配的索引节点号
```

```
unsigned int last_allco_block = 0; //上次分配的数据块号
/******************/

int getch() // 使用方法，在需要不显示输入的是什么的地方调用，返回值为用户输入的字符。
{
    int ch;
    struct termios oldt, newt;
    tcgetattr(STDIN_FILENO, &oldt); // 用来获取终端参数，成功返回零；失败返回非零
    newt = oldt;
    newt.c_lflag &= ~(ECHO | ICANON);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
}
/********格式化文件系统**********/
/*
 * 初始化组描述符
 * 初始化数据块位图
 * 初始化索引节点位图
 * 初始化索引节点表 -添加一个索引节点
 * 第一个数据块中写入当前目录和上一目录
 */
int initialize(ext2_inode *cu);
int format(ext2_inode *current)
{
    FILE *fp = NULL;
    int i;
```

```
    unsigned int zero[blocksiz / 4]; //零数组，用来初始化块为 0
    time_t now;
    time(&now);
    while (fp == NULL)
        fp = fopen(PATH, "w+");
    for (i = 0; i < blocksiz / 4; i++)
        zero[i] = 0;
    for (i = 0; i < blocks; i++) //初始化所有 4611 块为 0
    {
        fseek(fp, i * blocksiz, SEEK_SET);
        fwrite(&zero, blocksiz, 1, fp);
    }
    //初始化组描述符
    strcpy(group_desc.bg_volume_name, "Volume_name"); //初始化卷名为 abcd
    group_desc.bg_block_bitmap = 1;             //保存块位图的块号
    group_desc.bg_inode_bitmap = 2;             //保存索引节点位图的块号
    group_desc.bg_inode_table = 3;              //索引节点表的起始块号
    group_desc.bg_free_blocks_count = 4095;     //除去一个初始化目录。空闲数据块
的个数
    group_desc.bg_free_inodes_count = 4095;
    group_desc.bg_used_dirs_count = 1;
    strcpy(group_desc.psw, "123");
    fseek(fp, 0, SEEK_SET);
    fwrite(&group_desc, sizeof(ext2_group_desc), 1, fp); //第一块为组描述符

    //初始化数据块位图和索引节点位图，第一位置为 1
    zero[0] = 0x80000000;
    fseek(fp, 1 * blocksiz, SEEK_SET);
    fwrite(&zero, blocksiz, 1, fp); //第二块为块位图，块位图的第一位为 1
    fseek(fp, 2 * blocksiz, SEEK_SET);
```

```
    fwrite(&zero, blocksiz, 1, fp); //第三块为索引位图，索引节点位图的第一位为
1

    //初始化索引节点表，添加一个索引节点
    inode.i_mode = 2;
    inode.i_blocks = 1;
    inode.i_size = 64;
    inode.i_ctime = now;
    inode.i_atime = now;
    inode.i_mtime = now;
    inode.i_dtime = 0;
    fseek(fp, 3 * blocksiz, SEEK_SET);
    fwrite(&inode, sizeof(ext2_inode), 1, fp); //第四块开始为索引节点表
    //向第一个数据块写 当前目录
    dir.inode = 0;
    dir.rec_len = 32;   //默认目录体为32字节
    dir.name_len = 1;
    dir.file_type = 2;
    strcpy(dir.name, "."); //当前目录
    fseek(fp, data_begin_block * blocksiz, SEEK_SET);
    fwrite(&dir, sizeof(ext2_dir_entry), 1, fp);
    //当前目录之后写 上一目录
    dir.inode = 0;      //因为是根目录所以上一目录就是当前目录
    dir.rec_len = 32;
    dir.name_len = 2;
    dir.file_type = 2;
    strcpy(dir.name, ".."); //上一目录
    fseek(fp, data_begin_block * blocksiz + dirsiz, SEEK_SET);
    fwrite(&dir, sizeof(ext2_dir_entry), 1, fp); //第data_begin_block+1 =516 块
开始为数据
```

```
    //current = &inode;
    initialize(current);    //将指针指向根目录
    // last_allco_inode = 0; //上次分配的索引节点号
    // last_allco_block = 0; //上次分配的数据块号

    // current->i_mode = 2;
    // current->i_blocks = 1;
    // current->i_size = 64;
    // current->i_ctime = now;
    // current->i_atime = now;
    // current->i_mtime = now;
    // current->i_dtime = 0;

    printf("\n!!!!!!!!inode.i_size:%d\n", inode.i_size);
    fclose(fp);
    return 0;
}

//返回目录的起始存储位置，每个目录 32 字节
int dir_entry_position(int dir_entry_begin, int i_block[8]) // dir_entry_begin
目录体的相对开始字节
{
    int dir_blocks = dir_entry_begin / 512;   // 存储目录需要的块数
    int block_offset = dir_entry_begin % 512; // 块内偏移字节数
    int a;
    FILE *fp = NULL;
    if (dir_blocks <= 5) //前六个直接索引
        return data_begin_block * blocksiz + i_block[dir_blocks] * blocksiz +
block_offset;
```

```
    else //间接索引
    {
        while (fp == NULL)
            fp = fopen(PATH, "r+");
        dir_blocks = dir_blocks - 6;
        if (dir_blocks < 128) //一个块 512 字节，一个 int 为 4 个字节 一级索引有
512/4= 128 个
        {
            int a;
            fseek(fp, data_begin_block * blocksiz + i_block[6] * blocksiz +
dir_blocks * 4, SEEK_SET);
            fread(&a, sizeof(int), 1, fp);
            return data_begin_block * blocksiz + a * blocksiz + block_offset;
        }
        else //二级索引
        {
            dir_blocks = dir_blocks - 128;
            fseek(fp, data_begin_block * blocksiz + i_block[7] * blocksiz +
dir_blocks / 128 * 4, SEEK_SET);
            fread(&a, sizeof(int), 1, fp);
            fseek(fp, data_begin_block * blocksiz + a * blocksiz + dir_blocks %
128 * 4, SEEK_SET);
            fread(&a, sizeof(int), 1, fp);
            return data_begin_block * blocksiz + a * blocksiz + block_offset;
        }
        fclose(fp);
    }
}
/*在当前目录 打开一个目录
current 指向新打开的当前目录（ext2_inode）
```

```
*/
int Open(ext2_inode *current, char *name)
{
    FILE *fp = NULL;
    int i;
    while (fp == NULL)
        fp = fopen(PATH, "r+");
    for (i = 0; i < (current->i_size / 32); i++)
    {
        fseek(fp, dir_entry_position(i * 32, current->i_block), SEEK_SET); //定
位目录的偏移位置
        fread(&dir, sizeof(ext2_dir_entry), 1, fp);
        if (!strcmp(dir.name, name))
        {
            if (dir.file_type == 2) //目录
            {
                fseek(fp, 3 * blocksiz + dir.inode * sizeof(ext2_inode),
SEEK_SET);
                fread(current, sizeof(ext2_inode), 1, fp);
                fclose(fp);
                return 0;
            }
        }
    }
    fclose(fp);
    return 1;
}
/********关闭当前目录**********/
/*
关闭时仅修改最后访问时间
```

```
返回时 打开上一目录 作为当前目录
*/
int Close(ext2_inode *current)
{
    time_t now;
    ext2_dir_entry bentry;
    FILE *fout;
    fout = fopen(PATH, "r+");
    time(&now);
    current->i_atime = now; //修改最后访问时间
    fseek(fout, (data_begin_block + current->i_block[0]) * blocksiz, SEEK_SET);
    fread(&bentry, sizeof(ext2_dir_entry), 1, fout); // current's dir_entry

    fseek(fout, 3 * blocksiz + (bentry.inode) * sizeof(ext2_inode), SEEK_SET);
    fwrite(current, sizeof(ext2_inode), 1, fout);   //写入文件系统中
    fclose(fout);
    return Open(current, "..");
}
/*
read file content from directory 'current' in file 'name'
*/
int Read(ext2_inode *current, char *name)
{
    FILE *fp = NULL;
    int i;
    while (fp == NULL)
        fp = fopen(PATH, "r+");
    for (i = 0; i < (current->i_size / 32); i++) //遍历当前目录的目录项，
ext2_inode *current 指向当前目录，每个目录项 32 字节
    {
```

```
        fseek(fp, dir_entry_position(i * 32, current->i_block), SEEK_SET);  //返回目录项的起始存储位置
        fread(&dir, sizeof(ext2_dir_entry), 1, fp);
        if (!strcmp(dir.name, name))    // 比较文件名是否相同
        {
            if (dir.file_type == 1) // 如果文件名相同，且文件类型是文件的话（文件类型有目录和文件两种）
            {
                time_t now;
                ext2_inode node;
                char content_char;
                fseek(fp, 3 * blocksiz + dir.inode * sizeof(ext2_inode), SEEK_SET); //根据目录体中保存的索引节点号，找到文件的 inode 位置
                fread(&node, sizeof(ext2_inode), 1, fp); // original inode，node 为文件的 inode 信息

                i = 0;
                for (i = 0; i < node.i_size; i++) // 读出大小为 i_size 的文件，一次读一个 char
                {
                    fseek(fp, dir_entry_position(i, node.i_block), SEEK_SET); //根据指向数据块的文件指针 i_block，将默认的读写指针移动到文件的数据块中
                    fread(&content_char, sizeof(char), 1, fp);
                    if (content_char == 0xD) //0xD (ascii——回车\n)
                        printf("\n");
                    else
                        printf("%c", content_char);
                }
                printf("\n");
                time(&now);
```

```
                node.i_atime = now;  // 修改访问时间
                fseek(fp,  3  *  blocksiz  +  dir.inode  *  sizeof(ext2_inode),
SEEK_SET);
                fwrite(&node, sizeof(ext2_inode), 1, fp); // update inode 将修
改写入文件系统中
                fclose(fp);
                return 0;
            }
        }
    }
    fclose(fp);
    return 1;
}
//寻找空索引
int FindInode()
{
    FILE *fp = NULL;
    unsigned int zero[blocksiz / 4];
    int i;
    while (fp == NULL)
        fp = fopen(PATH, "r+");
    fseek(fp, 2 * blocksiz, SEEK_SET); // inode 位图
    fread(zero, blocksiz, 1, fp);       // zero 保存索引节点位图
    // unsigned int zero[128] ,每个 int4 字节，共 128 个，故一共能表示 128*4*32 位，
=512*8 没问题!

    for (i = last_allco_inode; i < (last_allco_inode + blocksiz / 4); i++) //一
个 inode 号是 int 存储，故为 4 字节，
```

// last_allco_inode + blocksiz / 4 其实 i 的绝对数值已经超出索引节点位图的存储范围，但是因为要判断 last_allco_inode 之前的索引节点有无空闲，所以后面计算的时候%取余即可。

```
{
    if (zero[i % (blocksiz / 4)] != 0xffffffff) //当还有空闲的索引节点时；一个 int4 字节，4*8=32 位
    //i % (blocksiz / 4)是某个索引节点号， zero[i % (blocksiz / 4)]表示 inode 位图中的某段 32 位区域 （128*4*32）一共有 128 个这样的区域
    {
        unsigned int j = 0x80000000, k = zero[i % (blocksiz / 4)], l = i;
        for (i = 0; i < 32; i++)
        {
            if (!(k & j)) // & 按位与，再取非，如果结果不为 0，说明第 i 位有空闲，否则，j = j / 2， 考察下一位是否空闲
            {
                zero[l % (blocksiz / 4)] = zero[l % (blocksiz / 4)] | j; //如果空闲，将此位置 1

                group_desc.bg_free_inodes_count -= 1; //索引节点数减 1
                fseek(fp, 0, 0); //移动到起始位置——组描述符所在块
                fwrite(&group_desc, sizeof(ext2_group_desc), 1, fp); //更新组描述符（索引节点数目信息）

                fseek(fp, 2 * blocksiz, SEEK_SET);
                fwrite(zero, blocksiz, 1, fp); //更新 inode 位图，zero 存储的是整个 inode 位图，所以直接更新即可

                last_allco_inode = l % (blocksiz / 4);
                fclose(fp);
                return l % (blocksiz / 4) * 32 + i; // 返回空闲的 inode 号
```

```
                }
                else
                    j = j / 2;  // 考察下一位
            }
        }
    }
    fclose(fp);
    return -1;
}
//寻找空 block
int FindBlock()
{
    FILE *fp = NULL;
    unsigned int zero[blocksiz / 4];
    int i;
    while (fp == NULL)
        fp = fopen(PATH, "r+");
    fseek(fp, 1 * blocksiz, SEEK_SET);
    fread(zero, blocksiz, 1, fp); // zero 保存块位图
    for (i = last_allco_block; i < (last_allco_block + blocksiz / 4); i++)
    {
        if (zero[i % (blocksiz / 4)] != 0xffffffff)
        {
            unsigned int j = 0X80000000, k = zero[i % (blocksiz / 4)], l = i;
            for (i = 0; i < 32; i++)
            {
                if (!(k & j))
                {
                    zero[l % (blocksiz / 4)] = zero[l % (blocksiz / 4)] | j;
                    group_desc.bg_free_blocks_count -= 1; //块数减 1
```

```
                    fseek(fp, 0, 0);
                    fwrite(&group_desc, sizeof(ext2_group_desc), 1, fp);
                    fseek(fp, 1 * blocksiz, SEEK_SET);
                    fwrite(zero, blocksiz, 1, fp);
                    last_allco_block = 1 % (blocksiz / 4);
                    fclose(fp);
                    return 1 % (blocksiz / 4) * 32 + i;
                }
                else
                    j = j / 2;
            }
        }
    }
    fclose(fp);
    return -1;
}
//删除 inode，更新 inode 节点位图
void DelInode(int len) //len 是 inode 号，是一个 unsigned int 值
{
    unsigned int zero[blocksiz / 4], i;
    int j;
    f = fopen(PATH, "r+");
    fseek(f, 2 * blocksiz, SEEK_SET);
    fread(zero, blocksiz, 1, f);
    i = 0x80000000;
    for (j = 0; j < len % 32; j++)
        i = i / 2;
    zero[len / 32] = zero[len / 32] ^ i;
    fseek(f, 2 * blocksiz, SEEK_SET);
    fwrite(zero, blocksiz, 1, f);
```

```
    fclose(f);
}
//删除 block 块，更新块位图
void DelBlock(int len)
{
    unsigned int zero[blocksiz / 4], i;
    int j;
    f = fopen(PATH, "r+");
    fseek(f, 1 * blocksiz, SEEK_SET);
    fread(zero, blocksiz, 1, f);
    i = 0x80000000;
    for (j = 0; j < len % 32; j++)
        i = i / 2;
    zero[len / 32] = zero[len / 32] ^ i;
    fseek(f, 1 * blocksiz, SEEK_SET);
    fwrite(zero, blocksiz, 1, f);
    fclose(f);
}
void add_block(ext2_inode *current, int i, int j) // 空间不够，故增加一个数据块
来存放内容
{
    FILE *fp = NULL;
    while (fp == NULL)
        fp = fopen(PATH, "r+");
    if (i < 6) //直接索引
    {
        current->i_block[i] = j;
    }
    else
    {
```

```
        i = i - 6;
        if (i == 0)
        {
            current->i_block[6] = FindBlock();
            fseek(fp, data_begin_block * blocksiz + current->i_block[6] *
blocksiz, SEEK_SET);
            fwrite(&j, sizeof(int), 1, fp);
        }
        else if (i < 128) //一级索引
        {
            fseek(fp, data_begin_block * blocksiz + current->i_block[6] *
blocksiz + i * 4, SEEK_SET);
            fwrite(&j, sizeof(int), 1, fp);
        }
        else //二级索引
        {
            i = i - 128;
            if (i == 0)
            {
                current->i_block[7] = FindBlock();
                fseek(fp, data_begin_block * blocksiz + current->i_block[7] *
blocksiz, SEEK_SET);
                i = FindBlock();
                fwrite(&i, sizeof(int), 1, fp);
                fseek(fp, data_begin_block * blocksiz + i * blocksiz, SEEK_SET);
                fwrite(&j, sizeof(int), 1, fp);
            }
            if (i % 128 == 0)
            {
```

```
                fseek(fp, data_begin_block * blocksiz + current->i_block[7] *
blocksiz + i / 128 * 4, SEEK_SET);
                i = FindBlock();
                fwrite(&i, sizeof(int), 1, fp);
                fseek(fp, data_begin_block * blocksiz + i * blocksiz, SEEK_SET);
                fwrite(&j, sizeof(int), 1, fp);
            }
            else
            {
                fseek(fp, data_begin_block * blocksiz + current->i_block[7] *
blocksiz + i / 128 * 4, SEEK_SET);
                fread(&i, sizeof(int), 1, fp);
                fseek(fp, data_begin_block * blocksiz + i * blocksiz + i % 128
* 4, SEEK_SET);
                fwrite(&j, sizeof(int), 1, fp);
            }
        }
    }
}
// 为当前目录寻找一个空目录体
int FindEntry(ext2_inode *current)
{
    FILE *fout = NULL;
    int location;       //条目的绝对地址
    int block_location; //块号
    int temp;           //每个 block 可以存放的 INT 数量
    int remain_block;   //剩余块数
    location = data_begin_block * blocksiz;
    temp = blocksiz / sizeof(int);
    fout = fopen(PATH, "r+");
```

```
    if (current->i_size % blocksiz == 0) //一个 BLOCK 使用完后增加一个块
    {
        add_block(current, current->i_blocks, FindBlock());
        current->i_blocks++;
    }
    if (current->i_blocks < 6) //前 6 个块直接索引
    {
        location += current->i_block[current->i_blocks - 1] * blocksiz;
        location += current->i_size % blocksiz;
    }
    else if (current->i_blocks < temp + 5) //一级索引
    {
        block_location = current->i_block[6];
        fseek(fout, (data_begin_block + block_location) * blocksiz +
(current->i_blocks - 6) * sizeof(int), SEEK_SET);
        fread(&block_location, sizeof(int), 1, fout);
        location += block_location * blocksiz;
        location += current->i_size % blocksiz;
    }
    else //二级索引
    {
        block_location = current->i_block[7];
        remain_block = current->i_blocks - 6 - temp;
        fseek(fout, (data_begin_block + block_location) * blocksiz +
(int)((remain_block - 1) / temp + 1) * sizeof(int), SEEK_SET);
        fread(&block_location, sizeof(int), 1, fout);
        remain_block = remain_block % temp;
        fseek(fout, (data_begin_block + block_location) * blocksiz +
remain_block * sizeof(int),
              SEEK_SET);
```

```
        fread(&block_location, sizeof(int), 1, fout);
        location += block_location * blocksiz;
        location += current->i_size % blocksiz + dirsiz;
    }
    current->i_size += dirsiz;
    fclose(fout);
    return location;
}
/*********创建文件或者目录*********/
/*
* type=1 创建文件
type=2 创建目录
* current 当前目录索引节点
* name 文件名或目录名
*/
int Create(int type, ext2_inode *current, char *name)
{
    FILE *fout = NULL;
    int i;
    int block_location;     // block location
    int node_location;      // node location
    int dir_entry_location; // dir entry location
    time_t now;
    ext2_inode ainode;
    ext2_dir_entry aentry, bentry; // bentry 保存当前系统的目录体信息
    time(&now);
    fout = fopen(PATH, "r+");
    node_location = FindInode(); // 寻找空索引

    // 检查是否存在重复文件或目录名称
```

```
    for (i = 0; i < current->i_size / dirsiz; i++)
    {
        fseek(fout,     dir_entry_position(i     *     sizeof(ext2_dir_entry),
current->i_block), SEEK_SET);
        fread(&aentry, sizeof(ext2_dir_entry), 1, fout);
        if (aentry.file_type == type && !strcmp(aentry.name, name))
            return 1;
    }

    fseek(fout, (data_begin_block + current->i_block[0]) * blocksiz, SEEK_SET);
    fread(&bentry, sizeof(ext2_dir_entry), 1, fout); // current's dir_entry
    if (type == 1)  //文件
    {
        ainode.i_mode = 1;
        ainode.i_blocks = 0; //文件暂无内容
        ainode.i_size = 0;   //初始文件大小为 0
        ainode.i_atime = now;
        ainode.i_ctime = now;
        ainode.i_mtime = now;
        ainode.i_dtime = 0;
        for (i = 0; i < 8; i++)
        {
            ainode.i_block[i] = 0;
        }
        for (i = 0; i < 24; i++)
        {
            ainode.i_pad[i] = (char)(0xff);
        }
    }
    else //目录
```

```
    {
        ainode.i_mode = 2;   //目录
        ainode.i_blocks = 1; //目录 当前和上一目录
        ainode.i_size = 64;  //初始大小 32*2=64 //一旦新建一个目录，该目录下就有"."和".."
        ainode.i_atime = now;
        ainode.i_ctime = now;
        ainode.i_mtime = now;
        ainode.i_dtime = 0;
        block_location = FindBlock();
        ainode.i_block[0] = block_location;
        for (i = 1; i < 8; i++)
        {
            ainode.i_block[i] = 0;
        }
        for (i = 0; i < 24; i++)
        {
            ainode.i_pad[i] = (char)(0xff);
        }
        //当前目录
        aentry.inode = node_location;
        aentry.rec_len = sizeof(ext2_dir_entry);
        aentry.name_len = 1;
        aentry.file_type = 2;
        strcpy(aentry.name, ".");
        printf(".dir created.\n");
        aentry.dir_pad = 0;
        fseek(fout, (data_begin_block + block_location) * blocksiz, SEEK_SET);
        fwrite(&aentry, sizeof(ext2_dir_entry), 1, fout);
        //上一级目录
```

```
    aentry.inode = bentry.inode;
    aentry.rec_len = sizeof(ext2_dir_entry);
    aentry.name_len = 2;
    aentry.file_type = 2;
    strcpy(aentry.name, "..");
    aentry.dir_pad = 0;
    fwrite(&aentry, sizeof(ext2_dir_entry), 1, fout);
    printf("..dir created.\n");
    //一个空条目
    aentry.inode = 0;
    aentry.rec_len = sizeof(ext2_dir_entry);
    aentry.name_len = 0;
    aentry.file_type = 0;
    aentry.name[EXT2_NAME_LEN] = 0;
    aentry.dir_pad = 0;
    fwrite(&aentry, sizeof(ext2_dir_entry), 14, fout); //清空数据块
}                                                    // end else
//保存新建 inode
fseek(fout, 3 * blocksiz + (node_location) * sizeof(ext2_inode), SEEK_SET);
fwrite(&ainode, sizeof(ext2_inode), 1, fout);
// 将新建 inode 的信息写入 current 指向的数据块
aentry.inode = node_location;
aentry.rec_len = dirsiz;
aentry.name_len = strlen(name);
if (type == 1)
{
    aentry.file_type = 1;
} //文件
else
{
```

```
        aentry.file_type = 2;
    } //目录
    strcpy(aentry.name, name);
    aentry.dir_pad = 0;
    dir_entry_location = FindEntry(current);
    fseek(fout, dir_entry_location, SEEK_SET); //定位条目位置
    fwrite(&aentry, sizeof(ext2_dir_entry), 1, fout);

    //保存 current 的信息,bentry 是 current 指向的 block 中的第一条
    //ext2_inode cinode;
    fseek(fout, 3 * blocksiz + (bentry.inode) * sizeof(ext2_inode), SEEK_SET);

    // fread(&cinode, sizeof(ext2_inode), 1, fout);
    // printf("after_cinode.i_size: %d\n", cinode.i_size);

    fwrite(current, sizeof(ext2_inode), 1, fout);
    fclose(fout);
    return 0;
}
/******************/
/*
 * write data to file 'name' in directory 'current'* if there isn't this file in
this directory ,remaind create a new one
 */
int Write(ext2_inode *current, char *name)
{
    FILE *fp = NULL;
    ext2_dir_entry dir;
    ext2_inode node;
    time_t now;
```

```
    char str;
    int i;
    while (fp == NULL)
        fp = fopen(PATH, "r+");
    while (1)
    {
        for (i = 0; i < (current->i_size / 32); i++)
        {
            fseek(fp, dir_entry_position(i * 32, current->i_block), SEEK_SET);
            fread(&dir, sizeof(ext2_dir_entry), 1, fp);
            if (!strcmp(dir.name, name))
            {
                if (dir.file_type == 1)
                {
                    fseek(fp, 3 * blocksiz + dir.inode * sizeof(ext2_inode),
SEEK_SET);
                    fread(&node, sizeof(ext2_inode), 1, fp);
                    break;
                }
            }
        }
        if (i < current->i_size / 32) // have file
            break;
        // Create(1,current,name); //have not file ,create a new file
        printf("There isn't this file,please create it first\n");
        return 0;
    }
    str = getch();

    while (str != 27) // 没有检测到ESC（ascii = 27）之前一直读
```

```
    {
        printf("%c", str);
        if (!(node.i_size % 512)) // 需要增加数据块
        {
            add_block(&node, node.i_size / 512, FindBlock());
            node.i_blocks += 1;
        }
        fseek(fp, dir_entry_position(node.i_size, node.i_block), SEEK_SET);
        fwrite(&str, sizeof(char), 1, fp);

        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, 3 * blocksiz + dir.inode * sizeof(ext2_inode), SEEK_SET);
    fwrite(&node, sizeof(ext2_inode), 1, fp);
    fclose(fp);
    printf("\n");
    return 0;
}
/**********ls 命令********/
/*
 * 列出当前目录的文件和目录
 */
```

```
void Ls(ext2_inode *current)
{
    ext2_dir_entry dir;
    int i, j;
    char timestr[150];
    ext2_inode node;
    f = fopen(PATH, "r+");

printf("Type\t\tFileName\tCreateTime\t\t\tLastAccessTime\t\t\tModifyTime\n");
    printf("\n!!!!!!!!current->i_size:%d\n", current->i_size);
    for (i = 0; i < current->i_size / 32; i++)
    {
        fseek(f, dir_entry_position(i * 32, current->i_block), SEEK_SET);
        fread(&dir, sizeof(ext2_dir_entry), 1, f);  // 读出目录项内容
        fseek(f, 3 * blocksiz + dir.inode * sizeof(ext2_inode), SEEK_SET);
        fread(&node, sizeof(ext2_inode), 1, f);     // 读出索引节点的内容
        strcpy(timestr, "");
        strcat(timestr, asctime(localtime(&node.i_ctime)));
        strcat(timestr, asctime(localtime(&node.i_atime)));
        strcat(timestr, asctime(localtime(&node.i_mtime)));
        for (j = 0; j < strlen(timestr) - 1; j++)
            if (timestr[j] == '\n')
            {
                timestr[j] = '\t';
            }
        if (dir.file_type == 1)
            printf("File\t\t%s\t\t%s", dir.name, timestr);
        else
            printf("Directory\t%s\t\t%s", dir.name, timestr);
```

```
    }
    fclose(f);
}
int initialize(ext2_inode *cu)
{
    f = fopen(PATH, "r+");
    fseek(f, 3 * blocksiz, 0);
    fread(cu, sizeof(ext2_inode), 1, f);
    fclose(f);
    return 0;
}
/*********修改文件系统密码*********/
/*
 * 修改成功返回 0
 * 修改不成功返回 1
 */
int Password()
{
    char psw[16], ch[10];
    printf("Please input the old password\n");
    scanf("%s", psw);
    if (strcmp(psw, group_desc.psw) != 0)
    {
        printf("Password error!\n");
        return 1;
    }
    while (1)
    {
        printf("Please input the new password:");
        scanf("%s", psw);
```

```
        while (1)
        {
            printf("Modify the password?[Y/N]");
            scanf("%s", ch);
            if (ch[0] == 'N' || ch[0] == 'n')
            {
                printf("You canceled the modify of your password\n");
                return 1;
            }
            else if (ch[0] == 'Y' || ch[0] == 'y')
            {
                strcpy(group_desc.psw, psw);
                f = fopen(PATH, "r+");
                fseek(f, 0, 0);
                fwrite(&group_desc, sizeof(ext2_group_desc), 1, f);
                fclose(f);
                return 0;
            }
            else
                printf("Meaningless command\n");
        }
    }
}
/*****************/
int login()
{
    char psw[16];
    printf("please input the password(init:123):");
    scanf("%s", psw);
    return strcmp(group_desc.psw, psw);
```

```
}
/******************/
void exitdisplay()
{
    printf("Thank you for using~ Byebye!\n");
    return;
}
/**************初始化文件系统*****************/
/*返回 1 初始化失败，返回 0 初始化成功*/
int initfs(ext2_inode *cu)
{
    f = fopen(PATH, "r+");
    if (f == NULL)
    {
        // char ch[20];/***********/
        char ch;
        int i;
        printf("File system couldn't be found. Do you want to create
one?\n[Y/N]");
        i = 1;
        while (i)
        {
            scanf("%c", &ch); /*******/
            switch (ch)
            {
            case 'Y':
            case 'y': /********/
                if (format(cu) != 0)
                    return 1;
                f = fopen(PATH, "r");
```

```
                i = 0;
                break;
            case 'N':
            case 'n': /*******/
                exitdisplay();
                return 1;
            default:
                printf("Sorry, meaningless command\n");
                break;
            }
        }
    }
    fseek(f, 0, SEEK_SET);
    fread(&group_desc, sizeof(ext2_group_desc), 1, f);
    fseek(f, 3 * blocksiz, SEEK_SET);
    fread(&inode, sizeof(ext2_inode), 1, f);
    fclose(f);
    initialize(cu);
    return 0;
}
/*********获取当前目录的目录名*********/
void getstring(char *cs, ext2_inode node)
{
    ext2_inode current = node;
    int i, j;
    ext2_dir_entry dir;
    f = fopen(PATH, "r+");
    Open(&current, ".."); // current 指向上一目录
    for (i = 0; i < node.i_size / 32; i++)
    {
```

```
        fseek(f, dir_entry_position(i * 32, node.i_block), SEEK_SET);
        fread(&dir, sizeof(ext2_dir_entry), 1, f);
        if (!strcmp(dir.name, "."))
        {
            j = dir.inode;
            break;
        }
    }
    for (i = 0; i < current.i_size / 32; i++)
    {
        fseek(f, dir_entry_position(i * 32, current.i_block), SEEK_SET);
        fread(&dir, sizeof(ext2_dir_entry), 1, f);
        if (dir.inode == j)
        {
            strcpy(cs, dir.name);
            return;
        }
    }
}
/*******在当前目录删除目录或者文件***********/
int Delet(int type, ext2_inode *current, char *name)
{
    FILE *fout = NULL;
    int i, j, t, k, flag;
    // int Nlocation,Elocation,Blocation,
    int Blocation2, Blocation3;
    int node_location, dir_entry_location, block_location,e_location;
    int block_location2, block_location3;
    ext2_inode cinode, tmpinode,einode;
    ext2_dir_entry bentry, centry, dentry,eentry;
```

```
    //一个空条目
    dentry.inode = 0;
    dentry.rec_len = sizeof(ext2_dir_entry);
    dentry.name_len = 0;
    dentry.file_type = 0;
    strcpy(dentry.name, "");
    dentry.dir_pad = 0;

    fout = fopen(PATH, "r+");
    t = (int)(current->i_size / dirsiz); //总条目数
    flag = 0;                            //是否找到文件或目录
    for (i = 0; i < t; i++)
    {
        dir_entry_location = dir_entry_position(i * dirsiz, current->i_block);
        fseek(fout, dir_entry_location, SEEK_SET);
        fread(&centry, sizeof(ext2_dir_entry), 1, fout);
        if ((strcmp(centry.name, name) == 0) && (centry.file_type == type))
        {
            flag = 1;
            j = i;
            break;
        }
    }
    if (flag)
    {
        node_location = centry.inode;   //inode 号
        fseek(fout, 3 * blocksiz + node_location * sizeof(ext2_inode), SEEK_SET);
//定位 INODE 位置
        fread(&cinode, sizeof(ext2_inode), 1, fout);
        block_location = cinode.i_block[0];
```

```
        //删文件夹
        if (type == 2)
        {
             while (cinode.i_size > 2 * dirsiz)
            {
                fseek(fout,dir_entry_position(cinode.i_size-
dirsiz,cinode.i_block),0);
                fread(&eentry, sizeof(ext2_dir_entry), 1, fout);
                Delet(eentry.file_type,&cinode,eentry.name);
            }
            {
                DelBlock(block_location);
                DelInode(node_location);
                dir_entry_location = dir_entry_position(current->i_size-dirsiz,
current->i_block); //找到 current 指向条目的最后一条
                fseek(fout, dir_entry_location, SEEK_SET);
                fread(&centry, dirsiz, 1, fout); //将最后一条条目存入 centry
                fseek(fout, dir_entry_location, SEEK_SET);
                fwrite(&dentry, dirsiz, 1, fout);                    //清空该位置
                dir_entry_location -= data_begin_block * blocksiz; //在数据中的
相对位置
                //如果这个位置刚好是一个块的起始位置，则删掉这个块（如果删除的文
件夹刚好是给目录体多分配的一个数据块的第一个目录项，则应该释放这个块）
                if (dir_entry_location % blocksiz == 0)
                {
                    DelBlock((int)(dir_entry_location / blocksiz));
                    current->i_blocks--;         //所用数据块数-1
                    if (current->i_blocks == 6) //如果只剩下 6 个直接索引，则删
除那个一级子索引
                        DelBlock(current->i_block[6]);
```

```
                    else if (current->i_blocks == (blocksiz / sizeof(int) + 6))
                    {
                        int a;
                        fseek(fout,     data_begin_block     *     blocksiz     +
current->i_block[7] * blocksiz, SEEK_SET);
                        fread(&a, sizeof(int), 1, fout);
                        DelBlock(a);
                        DelBlock(current->i_block[7]);
                    }
                    else if (!((current->i_blocks - 6 - blocksiz / sizeof(int)) %
(blocksiz / sizeof(int))))
                    {
                        int a;
                        fseek(fout,     data_begin_block     *     blocksiz     +
current->i_block[7] * blocksiz + ((current->i_blocks - 6 - blocksiz / sizeof(int))
/ (blocksiz / sizeof(int))), SEEK_SET);
                        fread(&a, sizeof(int), 1, fout);
                        DelBlock(a);
                    }
                }
                current->i_size -= dirsiz;
                if (j * dirsiz < current->i_size) //删除的条目如果不是最后一条，
用 centry 覆盖（将最后一个条目覆盖到删除的条目中）
                {
                    dir_entry_location   =   dir_entry_position(j   *   dirsiz,
current->i_block);
                    fseek(fout, dir_entry_location, SEEK_SET);
                    fwrite(&centry, dirsiz, 1, fout);
                }
                printf("The %s is deleted!", name);
```

```
            }
        }
        //删文件
        else
        {
            //删直接指向的块
            for (i = 0; i < 6; i++)
            {
                if (cinode.i_blocks == 0)
                {
                    break;
                }
                block_location = cinode.i_block[i];
                DelBlock(block_location);
                cinode.i_blocks--;
            }
            //删一级索引中的块
            if (cinode.i_blocks > 0) //（即初始情况下：i_blocks>=7）
            {
                block_location = cinode.i_block[6];
                fseek(fout, (data_begin_block + block_location) * blocksiz,
SEEK_SET);
                for (i = 0; i < blocksiz / sizeof(int); i++)
                {
                    if (cinode.i_blocks == 0)
                    {
                        break;
                    }
                    fread(&Blocation2, sizeof(int), 1, fout);
                    DelBlock(Blocation2);
```

```
                    cinode.i_blocks--;
                }
                DelBlock(block_location); // 删除一级索引
            }
            if (cinode.i_blocks > 0) //有二级索引存在
            {
                block_location = cinode.i_block[7];
                for (i = 0; i < blocksiz / sizeof(int); i++)
                {
                    fseek(fout, (data_begin_block + block_location) * blocksiz
+ i * sizeof(int), SEEK_SET);
                    fread(&Blocation2, sizeof(int), 1, fout);
                    fseek(fout, (data_begin_block + Blocation2) * blocksiz,
SEEK_SET);
                    for (k = 0; i < blocksiz / sizeof(int); k++)
                    {
                        if (cinode.i_blocks == 0)
                        {
                            break;
                        }
                        fread(&Blocation3, sizeof(int), 1, fout);
                        DelBlock(Blocation3);
                        cinode.i_blocks--;
                    }
                    DelBlock(Blocation2); //删除二级索引
                }
                DelBlock(block_location); // 删除一级索引
            }
```

```
            DelInode(node_location);
//删除文件的 inode
            dir_entry_location   =   dir_entry_position(current->i_size-dirsiz,
current->i_block); //找到 current（当前目录） 指向条目的最后一条
            //dir_entry_position 输入：相对位置，返回目录的起始存储位置

            printf("num:%d\n", current->i_size/dirsiz);

            fseek(fout, dir_entry_location, SEEK_SET);  //这里读取的位置应该是
第四个的末尾也就是五个长度-dirsiz
            fread(&centry, dirsiz, 1, fout); //将最后一条条目存入 centry

            printf("last entryname: %s\n", centry.name);
            printf("last entrynamelen: %d\n", centry.name_len);

            fseek(fout, dir_entry_location, SEEK_SET);
            fwrite(&dentry, dirsiz, 1, fout);                    //清空该位置
            dir_entry_location -= data_begin_block * blocksiz; //在数据中的位置
            //如果这个位置刚好是一个块的起始位置，则删掉这个块
            if (dir_entry_location % blocksiz == 0)
            {
                DelBlock((int)(dir_entry_location / blocksiz));
                current->i_blocks--;
                if (current->i_blocks == 6)
                    DelBlock(current->i_block[6]);
                else if (current->i_blocks == (blocksiz / sizeof(int) + 6))
                {
                    int a;
                    fseek(fout, data_begin_block * blocksiz + current->i_block[7]
* blocksiz, SEEK_SET);
```

```
                    fread(&a, sizeof(int), 1, fout);
                    DelBlock(a);
                    DelBlock(current->i_block[7]);
                }
                else if (!((current->i_blocks - 6 - blocksiz / sizeof(int)) %
(blocksiz / sizeof(int))))
                {
                    int a;
                    fseek(fout, data_begin_block * blocksiz + current->i_block[7]
* blocksiz + ((current->i_blocks - 6 - blocksiz / sizeof(int)) / (blocksiz /
sizeof(int))), SEEK_SET);
                    fread(&a, sizeof(int), 1, fout);
                    DelBlock(a);
                }
            }
            current->i_size -= dirsiz;  //更新当前目录的大小

            printf("num:%d\n", current->i_size/dirsiz);

            if (j * dirsiz < current->i_size) // 删除的条目如果不是最后一条，用
centry 覆盖被删除的条目
            {
                dir_entry_location    =    dir_entry_position(j    *    dirsiz,
current->i_block);
                fseek(fout, dir_entry_location, SEEK_SET);
                fwrite(&centry, dirsiz, 1, fout);
            }
        }
```

```
        fseek(fout,  (data_begin_block  +  current->i_block[0])  *  blocksiz,
SEEK_SET);
        fread(&bentry, sizeof(ext2_dir_entry), 1, fout); // current's dir_entry
//读操作写操作之后读写指针都会变

        // //
        // printf("\ntest name:%s\n", bentry.name);

        //fseek(fout,  3  *  blocksiz  +  (bentry.inode)  *  sizeof(ext2_inode),
SEEK_SET);  //当前目录的 inode
        //fread(&cinode, sizeof(ext2_inode), 1, fout);
        //printf("cinode.i_size: %d\n", cinode.i_size);
        // printf("current.i_size: %d\n", current->i_size);
        // //



        fseek(fout, 3 * blocksiz + (bentry.inode) * sizeof(ext2_inode), SEEK_SET);
//当前目录的 inode
        fwrite(current, sizeof(ext2_inode), 1, fout); //将 current 修改的数据写
回文件    //fwrite(&current, sizeof(ext2_inode), 1, fout);



        // // //
        // printf("current.i_size: %d\n", current->i_size);
        //  fseek(fout,  3  *  blocksiz  +  (bentry.inode)  *  sizeof(ext2_inode),
SEEK_SET);
        // fread(&tmpinode, sizeof(ext2_inode), 1, fout);
        // printf("tmpinode.i_size: %d\n", tmpinode.i_size);
    }
```

```
    else
    {
        fclose(fout);
        return 1;   //删除失败
    }
    fclose(fout);
    return 0;//成功删除
}

void pwd (char *str, ext2_inode *current){//显示当前目录的绝对路径
    FILE *fout = NULL;
    char string[100];
    char *slash = "/";

    int node_location, dir_entry_location, block_location;
    ext2_inode cinode;
    ext2_dir_entry pentry, centry;  //上级条目、当前条目

    fout = fopen(PATH, "r+");

    fseek(fout, (data_begin_block + current->i_block[0]) * blocksiz, SEEK_SET);
//定位到当前目录的.条目，即当前目录。
    fread(&centry, sizeof(ext2_dir_entry), 1, fout); // current's dir_entry  //
读操作写操作之后读写指针都会变
    //printf("\ndentry name:%s\n", dentry.name);

    fseek(fout, (data_begin_block + current->i_block[0]) * blocksiz + dirsiz,
SEEK_SET); //定位到当前目录的..条目，即上一级目录。
```

```
    fread(&pentry, sizeof(ext2_dir_entry), 1, fout); // current's dir_entry  //读操作写操作之后读写指针都会变
    //printf("\nbentry name:%s\n", bentry.name);

    fseek(fout, 3 * blocksiz + (pentry.inode) * sizeof(ext2_inode), SEEK_SET); //上一级目录的 inode
    fread(&cinode, sizeof(ext2_inode), 1, fout);

    getstring(string, cinode);
    //printf("上一个 string 为%s\n",string);
    //printf("\n%s ", string);

   //printf("\ni_size:%d\n", cinode.i_size);

    current = &cinode;
    if (centry.inode != pentry.inode){  //如果没有递归到根目录，继续递归
        // if (string != "."){ //如果当前目录的上一级是根目录，那么路径不应该加入".", 默认 home 绝对路径是"/"
        strcat(string, slash);
        strcat(string, str);
        strcpy(str, string);
        //printf("上一个 str 为%s\n",str);
        // }
        pwd(str, current);
    }

}

/*main shell*/
void shellloop(ext2_inode currentdir)
```

```
{
    char command[10], var1[10], var2[128], path[10];
    ext2_inode temp;
    int i, j;
    char currentstring[20];
    char
        ctable[14][10] = {"create", "delete", "cd", "close", "read", "write",
"password", "format", "exit", "login", "logout", "ls", "pwd", "help"};
    while (1)
    {
        getstring(currentstring, currentdir); //获取当前目录的目录名
        printf("\n[%s]> ", currentstring);
        scanf("%s", command);
        for (i = 0; i < 14; i++)
            if (!strcmp(command, ctable[i]))
                break;
        if (i == 0 || i == 1) //创建，删除 文件/目录 create f "" , delete f ""
        {
            scanf("%s", var1); /******/
            scanf("%s", var2);
            if (var1[0] == 'f')
                j = 1; //创建文件
            else if (var1[0] == 'd')
                j = 2; //创建目录
            else
            {
                printf("the first variant must be [f/d]");
                continue;
            }
            if (i == 0)
```

```
    {
        if (Create(j, &currentdir, var2) == 1)
            printf("Failed! %s can't be created\n", var2);
        else
            printf("Congratulations! %s is created\n", var2);
    }
    else
    {
        if (Delet(j, &currentdir, var2) == 1)
            printf("Failed! %s can't be deleted!\n", var2);
        else
            printf("Congratulations! %s is deleted!\n", var2);
    }
}
else if (i == 2) // open == cd change dir
{
    scanf("%s", var2);
    i = 0;
    j = 0;
    temp = currentdir;
    while (1)
    {
        path[i] = var2[j];
        if (path[i] == '/')
        {
            if (j == 0)
                initialize(&currentdir); // 将当前目录指针指向根目录/
            else if (i == 0) // 目录名中不能包含/
            {
                printf("path input error!\n");
```

```
                    break;
                }
                else //遇到'/'，说明要进入一个目录（每次只进入一级目录(调用Open)，故后面要将暂存的目录地址尾部设置为'\0'）
                {
                    path[i] = '\0';
                    if (Open(&currentdir, path) == 1)
                    {
                        printf("path input error!\n");
                        currentdir = temp;
                    }
                }
                i = 0; // 重新设置相对路径
            }
            else if (path[i] == '\0')
            {
                if (i == 0)
                    break;
                if (Open(&currentdir, path) == 1)
                {
                    printf("path input error!\n");
                    currentdir = temp;
                }
                break;
            }
            else
                i++;
            j++;
        }
    }
```

```
else if (i == 3) // close
{
    /*imagine the second variable suply number of layers to get out of*/
    scanf("%d", &i);
    for (j = 0; j < i; j++)
        if (Close(&currentdir) == 1)
        {
            printf("Warning! the number %d is too large\n", i);
            break;
        }
}
else if (i == 4) // read
{
    scanf("%s", var2);
    if (Read(&currentdir, var2) == 1)
        printf("Failed! The file can't be read\n");
}
else if (i == 5) // write
{
    printf("没有检测到ESC(ascii = 27)之前一直写数据\n");
    scanf("%s", var2);
    if (Write(&currentdir, var2) == 1)
        printf("Failed! The file can't be written\n");
}
else if (i == 6) // password
    Password();
else if (i == 7) // format
{
    while (1)
    {
```

```
                printf("Do you want to format the filesystem?\n It will be
dangerous to your data.\n");
                printf("[Y/N]");
                scanf("%s", var1);
                if (var1[0] == 'N' || var1[0] == 'n')
                    break;
                else if (var1[0] == 'Y' || var1[0] == 'y')
                {
                    format(&currentdir);
                    break;
                }
                else
                    printf("please input [Y/N]");
            }
        }
        else if (i == 8) // exit
        {
            while (1)
            {
                printf("Do you want to exit from filesystem?[Y/N]\n");
                scanf("%s", var2);
                if (var2[0] == 'N' || var2[0] == 'n')
                    break;
                else if (var2[0] == 'Y' || var2[0] == 'y')
                    return;
                else
                    printf("\nplease input [Y/N]\n");
            }
        }
        else if (i == 9) // login
```

```
    printf("Failed! You havn't logged out yet\n");
else if (i == 10) // logout
{
    while (i)
    {
        printf("Do you want to logout from filesystem?[Y/N]");
        scanf("%s", var1);
        if (var1[0] == 'N' || var1[0] == 'n')
            break;
        else if (var1[0] == 'Y' || var1[0] == 'y')
        {
            initialize(&currentdir);
            while (1)
            {
                printf("command: ");
                scanf("%s", var2);
                if (strcmp(var2, "login") == 0)
                {
                    if (login() == 0)
                    {
                        i = 0;
                        break;
                    }
                }
                else if (strcmp(var2, "exit") == 0)
                    return;
            }
        }
        else
            printf("please input [Y/N]");
```

```
            }
        }
        else if (i == 11) // ls
            Ls(&currentdir);
        else if (i == 12) {
            char string[100];
            for(int j=0;j<100;j++)
                string[j]=0;
            getstring(currentstring, currentdir); //获取当前目录的目录名
           // printf("\n%s\n ", currentstring);

            pwd(string, &currentdir);
            strcat(string, currentstring);
           // printf("这个 string 为%s\n",string);
            int i=0;
          for( i=0;string[i+1];i++)
            string[i]=string[i+1];
            string[i]=0;
            printf("%s\n",string);

        }
        else if (i == 13) {//help
            printf("
*************************************************************************
***\n");
            printf("   *                    An simulation of ext2 file system
*\n");
            printf("                                                          *
*\n");
```

```
            printf("              *     The     available     commands     are:
*\n");
            printf("    * 01.change dir    : cd+dir_name             02.create
dir     : create d+dir_name   *\n");
            printf("    * 03.create file   : create f+file_name     04.delete
dir     : delete d+dir_name   *\n");
            printf("    * 05.delete file   : delete f+file_name     06.read
file    : read+file_name      *\n");
            printf("    * 07.write  file   : write+file_name        08.absolute
path  : pwd                *\n");
            printf("    * 09.close  file   : close+num              10.change
pw      : password           *\n");
            printf("    * 11.list   items  : ls                       12.this
menu    : help               *\n");
            printf("              *     13.format     disk          :     format
14.exit           : exit               *\n");
            printf("          *    15.logout                        :    logout
*\n");
            printf("
*************************************************************************
***\n");
        }
        else
            printf("Failed! Command not available\n");
    }
}
int main()
{
    ext2_inode cu; /*current user*/
    printf("Hello! Welcome to Ext2_like file system!\n");
```

```
    if (initfs(&cu) == 1)
        return 0;
    if (login() != 0) /***************/
    {
        printf("Wrong password!It will terminate right away.\n");
        exitdisplay();
        return 0;
    }
    shellloop(cu);
    exitdisplay();
    return 0;
}
```