实验五 简单文件系统的实现

17042127 陶逸群

设计要求

- 实现多级目录的单用户的简单文件系统
- 提供format、mkdir、rmdir、ls、cd、create、open、close、write、read、rm、printfat、exit一组操作命令
- write命令实现截断写、填充写、追加写三种模式
- 实现读写多余一个磁盘块大小的数据
- 实现扇区大小可变
- 实现随机读写
- 以16进制打印fat表

相关说明

宏定义

```
#define END             0xffff  //fat表项占用
#define FREE            0x0000  //fat表项未占用
#define MAX_OPENFILE    10  //最多打开文件表项数
#define PATH_MAX_LENGTH 100  //路径最大长度
#define NAME_MAX_LENGTH 15  //文件名最大长度
#define PATH            "./sys"  //文件系统写入文件.sys中
#define MODIFIED '1'     //打开文件表项被修改
#define UNMODEFIED '0'  //打开文件表项未修改
#define ASSIGNED '1'   //打开文件表项被占用
#define UNASSIGNED '0' //打开文件表项被释放
#define DICTORY '0'  //fcb为文件夹
#define DATA '1'   //fcb为文件
#define EMPTY '0'  //文件、文件夹为空
#define UNEMPTY '1'//文件、文件夹不为空
```

数据结构

- fcb：文件信息

```
typedef struct FCB {
    char filename[8];//文件名
    char exname[4];//扩展名
    unsigned char attribute;//标识位 DATA('1')为数据文件 DICTORY('0')为文件夹
    struct tm time_info;//创建时间
    unsigned short first;//开始块
    unsigned long length;//占用大小(以BLOCK_SIZE递增)
    char free;//标识位 EMPTY('0')表示文件为空 UNEMPTY('1')表示文件不为空
    int len;//文件实际长
} fcb;
```

- fat：fat表项

```
typedef struct FAT {
    unsigned short id;
} fat;
```

- useropen：打开文件表项

```
typedef struct USEROPEN {
    fcb open_fcb;//打开的文件项目的信息
    char dir[80];//绝对路径
    int count;//读写指针
    char fcb_state;//标识是否被修改 MODIFIED('1')表示被修改 UNMODIFIED('0')表示未被修改
    char topenfile;//标识是否被占用 ASSIGNED('1')表示被占用 UNASSIGNED('0')表示未被占用
} useropen;
```

- block0：引导块

```
typedef struct BLOCK0 {
    int BLOCK_SIZE;//单个磁盘块大小
    int BLOCK_NUM;//磁盘块个数
    int SIZE;//文件系统总大小
    int FAT_NUM;//储存fat表需要的磁盘块数
    unsigned short root_start;//根目录开始块数
    unsigned char *start_block;//数据区开始指针
} block0;
```

全局变量

```
unsigned char *myvhard;//文件系统开始位置指针
useropen openfile_list[MAX_OPENFILE];//打开文件项目表
int curdir;//当前打开文件项目号
char current_dir[80];//当前路径
unsigned char *startp;//数据区开始指针
int BLOCK_SIZE;//单个磁盘块大小
int BLOCK_NUM;//磁盘块个数
int SIZE;//文件系统总大小
int FAT_NUM;//储存一张fat表需要的磁盘块数
int ROOT_START;//根目录开始块数
```

内存排布

引导块	fat表	根目录区	数据区
1块	FAT_NUM块	2块	BLOCK_NUM-FAT_NUM-3块

函数详解

- fcb_set：fcb文件信息设置

```
void fcb_set(fcb *f, const char *filename, const char *exname, unsigned char attribute, unsigned short first,unsigned long length, char ffree, int len);
```

- fcb_copy：fcb文件信息复制

```
void fcb_copy(fcb *dest, fcb *src);
```

- fat_format：fat表初始化

```
void fat_format();
```

将两张fat表中的每一项都设置成FREE（0x0000）。

- get_free：获取空闲块号

```
int get_free();
```

根据fat表获取空闲块号，如果获取失败返回-1。

- fat_allocate：fat表空间分配

根据起始块号和需要长度，调用get_free获取空闲块，利用两张fat表采用回退和提交操作分配fat表，无法分配成功（get_free返回-1）时回退。

```
int fat_allocate(unsigned short first, unsigned short length){
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
    int i,j;
    int allocated;
    //到达起始位置
    fat0 = fat0 + first;
    for (i = 0; i < length-1; i++) {
        allocated = get_free();
        if(allocated != -1){
            fat0->id = allocated;
        } else{
            fprintf(stderr, "系统已无空闲磁盘块\n");
            //回退操作
            fat0 = (fat *) (myvhard + BLOCK_SIZE);
            fat1 = (fat *) (myvhard + BLOCK_SIZE * (FAT_NUM + 1));
            for (j = 0; j < BLOCK_NUM; j++, fat0++, fat1++) {
                fat0->id = fat1->id;
            }
            return -1;
        }
        fat0 = (fat *) (myvhard + BLOCK_SIZE);
        fat0 = fat0 + allocated;
    }
    fat0->id = END;
    fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat1 = (fat *) (myvhard + BLOCK_SIZE * (FAT_NUM + 1));
    for (j = 0; j < BLOCK_NUM; j++, fat0++, fat1++) {
        fat1->id = fat0->id;
    }
    return 0;
}
```

- reclaim_space：释放fat表项目

```
void reclaim_space(int first){
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
    fat0 = fat0 + first;
    fat1 = fat1 + first;
    int offset;
    while (fat0->id != END) {
        offset = fat0->id - first;
        first = fat0->id;
        fat0->id = FREE;
        fat1->id = FREE;
        fat0 += offset;
        fat1 += offset;
    }
    fat0->id = FREE;
    fat1->id = FREE;
}
```

将从 `first` 块在 `fat` 表对应项目开始的 `fat` 表项置为 `FREE` ， `first` 的下一项为 `fat->id` 。

- openfile_set：打开项目设置

```
void openfile_set(int setdir,fcb *open_fcb, char dir[],int count, char fcbstate,
char topenfile);
```

- init_folder：文件夹初始化

```
void init_folder(int first, int second);
```

`first` 是要初始文件夹父目录的开始块号， `second` 是初始文件夹的开始块号，将 `second` 中的 `..` fcb的开始块号设置为 `second` ，除了 `..` 和 `.` ，将 `second` 中的其他fcb的 `free` 标识位都设置为 `EMPTY` 。

- find_free_space：寻找块中空闲位置

```
fcb *find_free_space(int first){
    int i;
    fcb *dir = (fcb *) (myvhard + BLOCK_SIZE * first);
    for (i = 0; i < BLOCK_SIZE / sizeof(fcb); i++, dir++) {
        if (dir->free == EMPTY) {
            return dir;
        }
    }
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat0 = fat0 + first;
    if(fat0->id == END){
        return NULL;
    } else{
        return find_free_space(fat0->id);
    }
}
```

`first` 为开始块号，返回 `free` 标识位为为 `EMPTY` 的fcb指针，若查找不到则返回 `NULL` 。首先查找 `first` 中是否存在 `free` 标识位为为 `EMPTY` 的fcb，查找不到则查阅fat表，到下一磁盘块查找，直到 `fat->id` 为 `END` 。

- get_fact_path：取得实际路径

```
//取得实际地址
void *get_fact_path(char *fact_path, const char *path) {
    //如果开头为\即是从根目录开始,就是其实际地址
    if (path[0] == '\\') {
        strcpy(fact_path, path);
        return 0;
    }
    char str[PATH_MAX_LENGTH];
    char *token,*final;
    //到当前目录
    memset(fact_path, '\0', PATH_MAX_LENGTH);
    strcpy(fact_path, current_dir);
    //划分
    strcpy(str, path);
    token = strtok(str, "/");
    do {
        //当前目录
        if (!strcmp(token, ".")) {
            continue;
        }
        if (!strcmp(token, "~")){
            continue;
        }
        //上层目录
        if (!strcmp(token, "..")) {
            //根目录的上层目录仍为根目录
            if (!strcmp(fact_path, "\\")) {
                continue;
            } else {
                //找到最后一个分割符出现位置
                final = strrchr(fact_path,'/');
                //最后的位置置成\0,到上层目录
                memset(final, '\0', 1);
                continue;
            }
        }
        strcat(fact_path, "/");
        strcat(fact_path, token);
    } while ((token = strtok(NULL, "/")) != NULL);
    return fact_path;
}
```

- get_useropenfd：取得未被占用的打开项目号

```
int get_useropenfd();
```

返回打开项目表中 `topenfile` 字段为 `UNASSIGNED` 的项目号，若全被打开占用，则返回 `-1` 。

- get_fullname：取得完整文件名

```
void get_fullname(char *fullname, fcb *fcb);
```

拼接文件名和后缀名。

- find_fcb：根据实际路径寻找文件或文件夹（调用find_fcb_r递归查找）
- find_fcb_r：递归查找文件或文件夹

```
//寻找文件或目录在块中位置  从根目录开始翻
fcb *find_fcb(const char *path, int *cnt) {
    char fact_path[PATH_MAX_LENGTH];
    get_fact_path(fact_path, path);
```

```
    if (strcmp(fact_path,"\\")==0) {
        *cnt = ROOT_START;
        return (fcb *) (myvhard + BLOCK_SIZE * ROOT_START);
    }
    char *token = strtok(fact_path, "/");
    token = strtok(NULL, "/");
    return find_fcb_r(token, ROOT_START,cnt);
}
//寻找文件或目录在块中位置  开始递归
fcb *find_fcb_r(char *token, int start,int *cnt) {
    int i, length = BLOCK_SIZE;
    char fullname[NAME_MAX_LENGTH] = "\0";
    fcb *root = (fcb *) (BLOCK_SIZE * start + myvhard);
    fcb *dir;
    block0 *init_block = (block0 *) myvhard;
    for (i = 0, dir = root; i < length / sizeof(fcb); i++, dir++) {
        if (dir->free == EMPTY) {
            continue;
        }
        //取得完整文件名
        get_fullname(fullname, dir);
        if (!strcmp(token, fullname)) {
            token = strtok(NULL, "/");
            if (token == NULL) {
                *cnt = start;
                return dir;
            }
            return find_fcb_r(token, dir->first,cnt);
        }
    }
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat0 = fat0 + start;
    if(fat0->id == END){
        *cnt = 0;
        return NULL;
    } else{
        return find_fcb_r(token, fat0->id,cnt);
    }
}
```

这两个函数返回对应文件名的fcb指针，并且修改 `cnt` 为父目录的 `first` 开始磁盘块号。

- startsys：启动文件系统

```
void startsys();
```

该函数首先读取引导块，完成全局变量的初始化，申请虚拟空间，初始化打开项目表等功能，若读取失败则调用 `format` 函数初始化文件系统。

- format：文件系统初始化

```
void format();
```

完成虚拟磁盘块的初始化，布局，初始化fat表，根目录的工作，该函数可更改磁盘块大小。

- cd_open：根据不同模式，改变路径或者打开文件

```
void cd_open(char *args,int mode);
```

`mode` 为 `DATA` 时则打开文件，为 `DICTORY` 时改变当前目录。

- pwd：遍历输出当前打开文件项目表的被占用项

```
void pwd();
```

- printfat：以16进制打印fat表

```
void printfat();
```

- read：取得一些必要参数（比如开始读取位置等）调用do_read随机读取文件
- do_read:随机读取文件

```
int do_read(int des_fd,int len,char *text){
    if((openfile_list[des_fd].count + len) > openfile_list[des_fd].open_fcb.len){
        fprintf(stderr, "读取超过文件总长\n");
        return -1;
    }
    // 计算逻辑块号和块内偏移量
    int block_num = openfile_list[des_fd].count / BLOCK_SIZE;
    int block_offset = openfile_list[des_fd].count % BLOCK_SIZE;
    char buf[BLOCK_SIZE];
    // 得到物理块号
    fat* fat1 = (fat*)(myvhard + BLOCK_SIZE);
    int cur_fat = openfile_list[des_fd].open_fcb.first;
    fat* fat_ptr = fat1 + openfile_list[des_fd].open_fcb.first;
    for(int i = 0; i < block_num; i++){
        cur_fat = fat_ptr->id;
        fat_ptr = fat1 + fat_ptr->id;
    }
    // 开始读取
    int cnt = 0;
    while(cnt < len){
        unsigned char* pos = (unsigned char*)(myvhard + BLOCK_SIZE*cur_fat);
        for(int i = 0; i < BLOCK_SIZE; ++i){
            buf[i] = pos[i];
        }
        for(; block_offset < BLOCK_SIZE; ++block_offset){
            text[cnt] = buf[block_offset];
            ++cnt;
            openfile_list[des_fd].count++;
            if(cnt == len){
                break;
            }
        }
        if(cnt < len){
            cur_fat = fat_ptr->id;
            fat_ptr = fat1 + fat_ptr->id;
            block_offset = 0;
        }
    }
    text[cnt] = '\0';
    return cnt;
}
void read(int des_fd){
    if(des_fd < 0 || des_fd >= MAX_OPENFILE){
        fprintf(stderr, "文件描述符错误\n");
        return;
    }
    int start;
    printf("请输入文件读取开始位置\n");
    scanf("%d",&start);
    int len;
    printf("请输入读取长度\n");
```

```
    scanf("%d",&len);
    char *text = malloc(len*2);
    if (openfile_list[des_fd].open_fcb.attribute == DICTORY){
        fprintf(stderr, "无法读入文件夹\n");
        return ;
    }
    if (openfile_list[des_fd].topenfile == UNASSIGNED){
        fprintf(stderr, "文件未打开\n");
        return ;
    }
    if (len > openfile_list[des_fd].open_fcb.len){
        fprintf(stderr, "读入长度大于文件总长\n");
        return ;
    }
    openfile_list[des_fd].count = start;
    int cnt = do_read(des_fd,  len, text);
    if(cnt == -1){
        fprintf(stderr, "读取文件错误\n");
        return ;
    }else{
        printf("%s\n", text);
        printf("共读取 %d B\n", cnt);
        getchar();
    }
}
```

- write：取得一些必要参数（比如开始写入位置、写入模式等）调用do_write随机写入文件

  这里将write的三个模式都转化为截断写。覆盖写和追加写的写入内容重新计算，后传入do_write。
- do_write:随机写入文件

```
void write(int des_fd,int mode){
    char aa;
    int len = 0;
    int str_len = 0;
    int len_sum;
    int str_start;
    printf("请输入要写入字节数\n");
    scanf("%d",&str_len);
    char* str = NULL;
    int cur;
    if(mode == 2){
        printf("请输入读写指针的位置\n");
        scanf("%d",&cur);
    } else{
        cur = openfile_list[des_fd].count;
    }
    if(mode == 1){
        str = malloc(str_len);
        len_sum = str_len;
    } else if(mode == 2){
        if((openfile_list[des_fd].open_fcb.length - cur) > str_len){
            len_sum = openfile_list[des_fd].open_fcb.length - cur + str_len;
            str_start = len_sum - cur - str_len;
        } else{
            len_sum = str_len + cur;
            str_start = len_sum;
        }
        str = malloc(len_sum*2);
    } else{
        len_sum = openfile_list[des_fd].open_fcb.length + str_len;
        str = malloc(len_sum*2);
    }
```

```
    if (openfile_list[des_fd].open_fcb.attribute == DICTORY){
        fprintf(stderr, "无法写入文件夹\n");
        return ;
    }
    if (openfile_list[des_fd].topenfile == UNASSIGNED){
        fprintf(stderr, "文件未打开\n");
        return ;
    }
    if(mode == 1){
        memset(str, '\0',str_len);
        getchar();
        printf("要随机填充吗？\n");
        if((aa=getchar())=='y'){
            for (int i = 0; i < str_len; ++i) {
                str[i] = (i%10)+'0';
            }
            getchar();
        } else{
            getchar();
            while (scanf("%c",&aa)!=EOF){
                str[len] = aa;
                len++;
                if(len>=str_len){
                    break;
                }
            }
        }
    } else if (mode == 2){
        openfile_list[des_fd].open_fcb.len = len_sum;
        openfile_list[des_fd].count = 0;
        memset(str, '\0',len_sum*2);
        do_read(des_fd,cur,str);
        char *input = malloc(str_len*2);
        memset(input, '\0',str_len*2);
        getchar();
        printf("要随机填充吗？\n");
        if((aa=getchar())=='y'){
            int i;
            for ( i = 0; i < str_len; ++i) {
                input[i] = (i%10)+'0';
            }
            input[i] = '\0';
            getchar();
        } else{
            getchar();
            while (scanf("%c",&aa)!=EOF){
                input[len] = aa;
                len++;
                if(len>=str_len){
                    break;
                }
            }
        }
        char *input_end = malloc(2*(len_sum-str_start));
        openfile_list[des_fd].count = str_start;
        do_read(des_fd,len_sum-str_start,input_end);
        strcat(str,input);
        free(input);
        strcat(str,input_end);
        free(input_end);
    } else{
        openfile_list[des_fd].count = 0;
        memset(str, '\0',len_sum*2);
        do_read(des_fd,openfile_list[des_fd].open_fcb.len,str);
        openfile_list[des_fd].open_fcb.len = len_sum;
```

```
98          char *input = malloc(str_len*2);
99          memset(input, '\0',str_len*2);
100         getchar();
101         printf("要随机填充吗？\n");
102         if((aa=getchar())=='y'){
103             int i;
104             for ( i = 0; i < str_len; ++i) {
105                 input[i] = (i%10)+'0';
106             }
107             input[i] = '\0';
108             getchar();
109         } else{
110             getchar();
111             while (scanf("%c",&aa)!=EOF){
112                 input[len] = aa;
113                 len++;
114                 if(len>=str_len){
115                     break;
116                 }
117             }
118         }
119         strcat(str,input);
120         free(input);
121     }
122     do_write(des_fd,str,len_sum,mode);
123 }
124 void do_write(int des_fd,char *str, int len,int mode){
125     int first = openfile_list[des_fd].open_fcb.first;
126     fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
127     fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
128     reclaim_space(first);
129     int cur;
130     int block_num = (int)((double)len/BLOCK_SIZE + 0.5);
131     int write = get_free();
132     first = write;
133     if (write == -1){
134         fprintf(stderr, "系统已无空闲磁盘块\n");
135         fat0 = (fat *) (myvhard + BLOCK_SIZE);
136         fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
137         memcpy(fat0, fat1, FAT_NUM*BLOCK_SIZE);
138     }
139     memcpy(myvhard+write*(BLOCK_SIZE),str,BLOCK_SIZE);
140     fcb *dir = find_fcb(openfile_list[des_fd].dir,&cur);
141     dir->first = write;
142     int old = write;
143     int i = 1;
144     fat0 = fat0 + old;
145     fat0->id = END;
146     block_num--;
147     while (i <= block_num){
148         write = get_free();
149         if (write == -1){
150             fprintf(stderr, "系统已无空闲磁盘块\n");
151             fat0 = (fat *) (myvhard + BLOCK_SIZE);
152             fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
153             memcpy(fat0, fat1, FAT_NUM*BLOCK_SIZE);
154         }
155         fat0->id = write;
156         fat0 = (fat *) (myvhard + BLOCK_SIZE);
157         fat0 = fat0 + write;
158         fat0->id = END;
159         memcpy(myvhard+write*(BLOCK_SIZE),str+BLOCK_SIZE*i,BLOCK_SIZE);
160         i++;
161     }
162     fat0 = (fat *) (myvhard + BLOCK_SIZE);
```

```
163       fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
164       memcpy(fat1, fat0, FAT_NUM*BLOCK_SIZE);
165       dir->length = (int)((double)len/BLOCK_SIZE + 0.5)*BLOCK_SIZE;
166       dir->len = len;
167       openfile_set(des_fd,dir,openfile_list[des_fd].dir,0,UNMODEFIED,ASSIGNED);
168   }
```

- close：关闭文件

```
1   void close(int fd);
```

`fd` 为要关闭文件在打开项目表中的编号。

- ls：遍历输出当前文件夹下所有文件夹和文件

```
1   void ls();
```

- rmdir_rm：根据不同模式，删除文件夹或文件

```
1   void rmdir_rm(char *args, int mode);
```

`mode` 为 `DATA` 时则删除文件，为 `DICTORY` 时删除文件夹。

- mkdir_create：根据不同模式，创建文件夹或文件

```
1   void mkdir_create(char *args,int mode);
```

`mode` 为 `DATA` 时则创建文件，为 `DICTORY` 时创建文件夹。

- my_exitsys：退出文件系统，写入文件。

```
1   void my_exitsys();
```

结果演示

- foramt，startsys测试

```
C:\Users\10059\CLionProjects\untitled13\cmake-build-debug\untitled13.exe
文件系统未安装将调用format函数安装文件系统
请输入每块磁盘块大小
1024
请输入磁盘块个数
1000
文件系统初始化成功
\>
```

- mkdir，ls测试

```
C:\Users\10059\CLionProjects\untitled13\cmake-build-debug\untitled13.exe
文件系统初始化成功
\>mkdir ss
 \>mkdir cc
 \>ls
 项目名为：.,扩展名为:dic,属性为0,长度为1024,修改时间为：2019年12月15日17时6分3秒
项目名为：..,扩展名为:dic,属性为0,长度为1024,修改时间为：2019年12月15日17时6分3秒
项目名为：ss,扩展名为:dic,属性为0,长度为1024,修改时间为：2019年12月15日17时8分25秒
项目名为：cc,扩展名为:dic,属性为0,长度为1024,修改时间为：2019年12月15日17时8分27秒
\>
```

- create，cd，open测试

```
\>mkdir \/ss/sss
 \>cd ss
 项目名为：       ss扩展名为:dic    路径为:\/ss修改时间为：2019年12月15日17时8分25秒
\/ss>cd sss
 项目名为：      sss扩展名为:dic    路径为:\/ss/sss修改时间为：2019年12月15日17时10分45秒
\/ss/sss>create test.c
 \/ss/sss>open test.c
 项目名为：     test扩展名为:c      路径为:\/ss/sss/test.c修改时间为：2019年12月15日19时31分11秒
\/ss/sss/test.c>
```

- write，read测试

```
\/ss/sss/test.c>write
 请选择写入模式(1.截断2.覆盖3.追加)
1
当前打开文件/文件夹表为
第0项,项目名为：.扩展名为:dic路径为:\修改时间为：2019年12月15日17时6分3秒
第1项,项目名为：ss扩展名为:dic路径为:\/ss修改时间为：2019年12月15日17时8分25秒
第2项,项目名为：sss扩展名为:dic路径为:\/ss/sss修改时间为：2019年12月15日17时10分45秒
第3项,项目名为：test扩展名为:c路径为:\/ss/sss/test.c修改时间为：2019年12月15日19时31分11秒
输入想要写入的项号
3
请输入要写入字节数
2048
要随机填充吗？
y
\/ss/sss/test.c>read
 当前打开文件/文件夹表为
第0项,项目名为：.扩展名为:dic路径为:\修改时间为：2019年12月15日17时6分3秒
第1项,项目名为：ss扩展名为:dic路径为:\/ss修改时间为：2019年12月15日17时8分25秒
```

第1项,项目名为：ss扩展名为:dic路径为:\/ss修改时间为：2019年12月15日17时8分25秒
第2项,项目名为：sss扩展名为:dic路径为:\/ss/sss修改时间为：2019年12月15日17时10分45秒
第3项,项目名为：test扩展名为:c路径为:\/ss/sss/test.c修改时间为：2019年12月15日19时31分11秒
输入想要读入的项号
3
请输入文件读取开始位置
1024
请输入读取长度
1024
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
4567890123456789012345678901234567890123456789012345678901234
共读取 1024 B
4567890123456789012345678901234567890123456789012345678901234
共读取 1024 B
\/ss/sss/test.c>*write*
请选择写入模式(1.截断2.覆盖3.追加)
2
当前打开文件/文件夹表为
第0项,项目名为：.扩展名为:dic路径为:\修改时间为：2019年12月15日17时6分3秒
第1项,项目名为：ss扩展名为:dic路径为:\/ss修改时间为：2019年12月15日17时8分25秒
第2项,项目名为：sss扩展名为:dic路径为:\/ss/sss修改时间为：2019年12月15日17时10分45秒
第3项,项目名为：test扩展名为:c路径为:\/ss/sss/test.c修改时间为：2019年12月15日19时31分11秒
输入想要写入的项号
3
请输入要写入字节数
1024
请输入读写指针的位置
2048
要随机填充吗？
y
\/ss/sss/test.c>*read*

\/ss/sss/test.c>*read*
当前打开文件/文件夹表为
第0项,项目名为：.扩展名为:dic路径为:\修改时间为：2019年12月15日17时6分3秒
第1项,项目名为：ss扩展名为:dic路径为:\/ss修改时间为：2019年12月15日17时8分25秒
第2项,项目名为：sss扩展名为:dic路径为:\/ss/sss修改时间为：2019年12月15日17时10分45秒
第3项,项目名为：test扩展名为:c路径为:\/ss/sss/test.c修改时间为：2019年12月15日19时31分11秒
输入想要读入的项号
3
请输入文件读取开始位置
0
请输入读取长度
3072
0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567

```
234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
234567890123456789012345678901234567890123456789012345678901234567890123
共读取 3072 B
\/ss/sss/test.c>
```

- printfat测试

```
\/ss/sss/test.c>printfat
 ffff 0002 ffff 0004 ffff 0006 ffff ffff ffff ffff 000b 000c ffff 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

- close，rmdir，rm测试

```
\/ss/sss/test.c>close
 当前打开文件/文件夹表为
第0项,项目名为：.扩展名为:dic路径为:\修改时间为：2019年12月15日17时6分3秒
第3项,项目名为：test扩展名为:c路径为:\/ss/sss/test.c修改时间为：2019年12月15日19时31分11秒
输入想要关闭的项号
3
\>rm \/ss/sss/test.c
 \>rmdir ss
 \>文件夹\/ss中还存在文件

 命令错误!
\>rmdir \/ss/sss
 \>rmdir ss
 \>rmdir cc
 \>
```

源码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdint.h>
#define END             0xffff  //fat表项占用
#define FREE            0x0000  //fat表项未占用
#define MAX_OPENFILE    10  //最多打开文件表项数
#define PATH_MAX_LENGTH 100  //路径最大长度
#define NAME_MAX_LENGTH 15  //文件名最大长度
#define PATH            "./sys"  //文件系统写入文件.sys中
#define MODIFIED '1'     //打开文件表项被修改
#define UNMODEFIED '0'  //打开文件表项未修改
#define ASSIGNED '1'   //打开文件表项被占用
#define UNASSIGNED '0' //打开文件表项被释放
#define DICTORY '0'  //fcb为文件夹
#define DATA '1'   //fcb为文件
#define EMPTY '0'  //文件、文件夹为空
#define UNEMPTY '1'//文件、文件夹不为空
typedef struct FCB {
    char filename[8];//文件名
    char exname[4];//扩展名
    unsigned char attribute;//标识位 DATA('1')为数据文件 DICTORY('0')为文件夹
    struct tm time_info;//创建时间
    unsigned short first;//开始块
    unsigned long length;//占用大小(以BLOCK_SIZE递增)
```

```
27      char free;//标识位 EMPTY('0')表示文件为空 UNEMPTY('1')表示文件不为空
28      int len;//文件实际长度
29  } fcb;
30
31  typedef struct FAT {
32      unsigned short id;
33  } fat;
34
35  typedef struct USEROPEN {
36      fcb open_fcb;//打开的文件项目的信息
37      char dir[80];//绝对路径
38      int count;//读写指针
39      char fcb_state;//标识是否被修改 MODIFIED('1')表示被修改 UNMODIFIED('0')表示未被
    修改
40      char topenfile;//标识是否被占用 ASSIGNED('1')表示被占用 UNASSIGNED('0')表示未被
    占用
41  } useropen;
42
43  typedef struct BLOCK0 {
44      int BLOCK_SIZE;//单个磁盘块大小
45      int BLOCK_NUM;//磁盘块个数
46      int SIZE;//文件系统总大小
47      int FAT_NUM;//储存fat表需要的磁盘块数
48      unsigned short root_start;//根目录开始块数
49      unsigned char *start_block;//数据区开始指针
50  } block0;
51
52  unsigned char *myvhard;//文件系统开始位置指针
53  useropen openfile_list[MAX_OPENFILE];//打开文件项目表
54  int curdir;//当前打开文件项目号
55  char current_dir[80];//当前路径
56  unsigned char *startp;//数据区开始指针
57  int BLOCK_SIZE;//单个磁盘块大小
58  int BLOCK_NUM;//磁盘块个数
59  int SIZE;//文件系统总大小
60  int FAT_NUM;//储存一张fat表需要的磁盘块数
61  int ROOT_START;//根目录开始块数
62
63  void openfile_set(int setdir,fcb *open_fcb, char dir[],int count, char fcbstate,
    char topenfile);
64  void fcb_set(fcb *f, const char *filename, const char *exname, unsigned char
    attribute, unsigned short first,
65              unsigned long length, char ffree, int len);
66  void fcb_copy(fcb *dest, fcb *src);
67  int do_read(int des_fd,int len,char *text);
68  void format();
69  void fat_format();
70  int get_free();
71  int fat_allocate(unsigned short first, unsigned short length);
72  //取得实际地址
73  void *get_fact_path(char *fact_path, const char *path);
74  fcb *find_fcb_r(char *token, int start,int *cnt);
75  fcb *find_fcb(const char *path, int *cnt);
76  void init_folder(int first, int second);
77  void reclaim_space(int first);
78  void reclaim_space_fat(int cnt);
79  void do_write(int des_fd,char *str, int len,int mode);
80  void startsys() {
81      FILE *fp;
82      int i;
83      if ((fp = fopen(PATH, "r")) != NULL) {
84          //首先读取引导块，确定磁盘块大小
85          block0 *init_block = (block0 *) malloc(sizeof(block0));
86          memset(init_block, 0, sizeof(block0));
87          fread(init_block, sizeof(block0), 1, fp);
```

```
88          BLOCK_SIZE = init_block->BLOCK_SIZE;
89          SIZE = init_block->SIZE;
90          BLOCK_NUM = init_block->BLOCK_NUM;
91          FAT_NUM = init_block->FAT_NUM;
92          ROOT_START = init_block->root_start;
93          fclose(fp);
94          fp = fopen(PATH, "r");
95          myvhard = (unsigned char *) malloc(SIZE);
96          memset(myvhard, 0, SIZE);
97          fread(myvhard, SIZE, 1, fp);
98          fclose(fp);
99      } else {
100         printf("文件系统未安装将调用format函数安装文件系统\n");
101         format();
102     }
103     //初始化用户打开表
104     //第一项初始化为根目录
105     openfile_set(0,((fcb *) (myvhard + (1+FAT_NUM*2) *
    BLOCK_SIZE)),"\\",0,UNMODEFIED,ASSIGNED);
106     curdir = 0;//当前目录
107     //其他项置为空
108     fcb *empty = (fcb *) malloc(sizeof(fcb));//生成一个空的FCB
109     fcb_set(empty, "\0", "\0", DICTORY, 0, 0, EMPTY,0);
110     for (i = 1; i < MAX_OPENFILE; i++) {
111         openfile_set(i,empty,"\0",0,UNMODEFIED,UNASSIGNED);
112     }
113     //设置当前目录
114     strcpy(current_dir, openfile_list[curdir].dir);
115     startp = ((block0 *) myvhard)->start_block;
116     free(empty);
117 }
118 void format() {
119     int i;
120     int first;
121     FILE *fp;
122     block0 *init_block = (block0 *) malloc(sizeof(block0));
123     unsigned char *ptr;
124     printf("请输入每块磁盘块大小\n");
125     scanf("%d",&init_block->BLOCK_SIZE);
126     printf("请输入磁盘块个数\n");
127     scanf("%d",&init_block->BLOCK_NUM);
128     getchar();
129     init_block->SIZE = init_block->BLOCK_NUM * init_block->BLOCK_SIZE;
130     SIZE = init_block->SIZE;
131     BLOCK_SIZE = init_block->BLOCK_SIZE;
132     BLOCK_NUM = init_block->BLOCK_NUM;
133     init_block->FAT_NUM = (int) (((init_block->BLOCK_NUM)*
    sizeof(fat))/init_block->BLOCK_SIZE)+1;
134     FAT_NUM = init_block->FAT_NUM;
135     init_block->root_start = 1+2*FAT_NUM;
136     ROOT_START = init_block->root_start;
137     init_block = realloc(init_block,init_block->SIZE);
138     ptr = (unsigned char *) init_block;
139     myvhard = (unsigned char *) init_block;
140     init_block->start_block = (unsigned char *) (init_block + BLOCK_SIZE *
    (FAT_NUM * 2 + 1 +2));
141     ptr = ptr + BLOCK_SIZE;
142     //初始化FAT
143     fat_format();
144     //分配空间给引导块和FAT1 FAT2块
145     //第一次get_free未检查
146     fat_allocate(get_free(),1);
147     fat_allocate(get_free(), FAT_NUM);
148     fat_allocate(get_free(), FAT_NUM);
149     ptr += BLOCK_SIZE * 2*FAT_NUM;
```

```
150     //分配空间给根目录
151     fcb *root = (fcb *) ptr;
152     first = get_free();
153     if (first == -1){
154         fprintf(stderr, "系统已无空闲磁盘块\nformat失败\n");
155         exit(-1);
156     }
157     if(fat_allocate(first, 2) == -1){
158         fprintf(stderr, "系统已无空闲磁盘块\nformat失败\n");
159         exit(-1);
160     }
161     fcb_set(root, ".", "dic", DICTORY, first, BLOCK_SIZE , UNEMPTY,0);
162     root++;
163     fcb_set(root, "..", "dic", DICTORY, first, BLOCK_SIZE, UNEMPTY,0);
164     root++;
165     // 初始化根目录区剩余的表项
166     for (i = 2; i < BLOCK_SIZE / sizeof(fcb); i++, root++) {
167         root->free = EMPTY;
168     }
169     root = (fcb *)(myvhard + BLOCK_SIZE *(2+2*FAT_NUM));
170     for (i = 0; i < BLOCK_SIZE / sizeof(fcb); i++, root++) {
171         root->free = EMPTY;
172     }
173     //写入文件
174     fp = fopen(PATH, "w");
175     fwrite(myvhard, SIZE, 1, fp);
176     fclose(fp);
177 }
178 
179 //fcb设置
180 void fcb_set(fcb *f, const char *filename, const char *exname, unsigned char
    attribute, unsigned short first,
181              unsigned long length, char ffree, int len) {
182     time_t *now = (time_t *) malloc(sizeof(time_t));
183     struct tm *time_info;
184     time(now);
185     time_info = localtime(now);
186     memset(f->filename, 0, 8);
187     memset(f->exname, 0, 4);
188     strncpy(f->filename, filename, 7);
189     strncpy(f->exname, exname, 3);
190     f->attribute = attribute;
191     memcpy(&f->time_info,time_info, sizeof(struct tm));
192     f->first = first;
193     f->length = length;
194     f->free = ffree;
195     f->len = len;
196     free(now);
197 }
198 //打开项设置
199 void openfile_set(int setdir,fcb *open_fcb, char dir[],int count, char fcbstate,
    char topenfile){
200     fcb_copy(&openfile_list[setdir].open_fcb, open_fcb);
201     strcpy(openfile_list[setdir].dir, dir);
202     openfile_list[setdir].count = count;//读写指针为0
203     openfile_list[setdir].fcb_state = fcbstate;
204     openfile_list[setdir].topenfile = topenfile;
205 }
206 //fcb复制
207 void fcb_copy(fcb *dest, fcb *src) {
208     memset(dest->filename, '\0', 8);
209     memset(dest->exname, '\0', 3);
210     strcpy(dest->filename, src->filename);
211     strcpy(dest->exname, src->exname);
212     memcpy(&dest->time_info, &src->time_info, sizeof(struct tm));
```

```
    dest->attribute = src->attribute;
    dest->first = src->first;
    dest->length = src->length;
    dest->free = src->free;
    dest->len = src->len;
}
//FAT初始化
void fat_format(){
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * (FAT_NUM + 1));
    int i;
    for (i = 0; i < BLOCK_NUM; i++, fat0++, fat1++) {
        fat0->id = FREE;
        fat1->id = FREE;
    }
}
//获取空闲块 否则返回-1
int get_free() {
    unsigned char *ptr = myvhard;
    fat *fat0 = (fat *) (ptr + BLOCK_SIZE);
    fat *fat1 = (fat *) (ptr + BLOCK_SIZE * (1+FAT_NUM));
    int i ;
    for (i = 0; i < BLOCK_NUM; i++, fat0++,fat1++) {
        if (fat0->id == FREE){
            fat0->id = END;
            return i;
        }
    }
    return -1;
}
//fat表分配空间
int fat_allocate(unsigned short first, unsigned short length){
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
    int i,j;
    int allocated;
    //到达起始位置
    fat0 = fat0 + first;
    for (i = 0; i < length-1; i++) {
        allocated = get_free();
        if(allocated != -1){
            fat0->id = allocated;
        } else{
            fprintf(stderr, "系统已无空闲磁盘块\n");
            //回退操作
            fat0 = (fat *) (myvhard + BLOCK_SIZE);
            fat1 = (fat *) (myvhard + BLOCK_SIZE * (FAT_NUM + 1));
            for (j = 0; j < BLOCK_NUM; j++, fat0++, fat1++) {
                fat0->id = fat1->id;
            }
            return -1;
        }
        fat0 = (fat *) (myvhard + BLOCK_SIZE);
        fat0 = fat0 + allocated;
    }
    fat0->id = END;
    fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat1 = (fat *) (myvhard + BLOCK_SIZE * (FAT_NUM + 1));
    for (j = 0; j < BLOCK_NUM; j++, fat0++, fat1++) {
        fat1->id = fat0->id;
    }
    return 0;
}
void get_fullname(char *fullname, fcb *fcb) {
    memset(fullname, '\0', NAME_MAX_LENGTH);
```

```
278         strcat(fullname, fcb->filename);
279         if (fcb->attribute == DATA) {
280             strncat(fullname, ".", 2);
281             strncat(fullname, fcb->exname, 4);
282         }
283     }
284     int get_useropenfd(){
285         int i;
286         for (i = 0; i < MAX_OPENFILE; i++) {
287             if (openfile_list[i].topenfile == UNASSIGNED) {
288                 return i;
289             }
290         }
291         return -1;
292     }
293     //取得实际地址
294     void *get_fact_path(char *fact_path, const char *path) {
295         //如果开头为\即是从根目录开始,就是其实际地址
296         if (path[0] == '\\') {
297             strcpy(fact_path, path);
298             return 0;
299         }
300         char str[PATH_MAX_LENGTH];
301         char *token,*final;
302         //到当前目录
303         memset(fact_path, '\0', PATH_MAX_LENGTH);
304         strcpy(fact_path, current_dir);
305         //划分
306         strcpy(str, path);
307         token = strtok(str, "/");
308         do {
309             //当前目录
310             if (!strcmp(token, ".")) {
311                 continue;
312             }
313             if (!strcmp(token, "~")){
314                 continue;
315             }
316             //上层目录
317             if (!strcmp(token, "..")) {
318                 //根目录的上层目录仍为根目录
319                 if (!strcmp(fact_path, "\\")) {
320                     continue;
321                 } else {
322                     //找到最后一个分割符出现位置
323                     final = strrchr(fact_path,'/');
324                     //最后的位置置成\0,到上层目录
325                     memset(final, '\0', 1);
326                     continue;
327                 }
328             }
329             strcat(fact_path, "/");
330             strcat(fact_path, token);
331         } while ((token = strtok(NULL, "/")) != NULL);
332         return fact_path;
333     }
334     //寻找文件或目录在块中位置  从根目录开始翻
335     fcb *find_fcb(const char *path, int *cnt) {
336         char fact_path[PATH_MAX_LENGTH];
337         get_fact_path(fact_path, path);
338         if (strcmp(fact_path,"\\")==0) {
339             *cnt = ROOT_START;
340             return (fcb *) (myvhard + BLOCK_SIZE * ROOT_START);
341         }
342         char *token = strtok(fact_path, "/");
```

```
    token = strtok(NULL, "/");
    return find_fcb_r(token, ROOT_START,cnt);
}
//寻找文件或目录在块中位置  开始递归
fcb *find_fcb_r(char *token, int start,int *cnt) {
    int i, length = BLOCK_SIZE;
    char fullname[NAME_MAX_LENGTH] = "\0";
    fcb *root = (fcb *) (BLOCK_SIZE * start + myvhard);
    fcb *dir;
    block0 *init_block = (block0 *) myvhard;
    for (i = 0, dir = root; i < length / sizeof(fcb); i++, dir++) {
        if (dir->free == EMPTY) {
            continue;
        }
        //取得完整文件名
        get_fullname(fullname, dir);
        if (!strcmp(token, fullname)) {
            token = strtok(NULL, "/");
            if (token == NULL) {
                *cnt = start;
                return dir;
            }
            return find_fcb_r(token, dir->first,cnt);
        }
    }
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat0 = fat0 + start;
    if(fat0->id == END){
        *cnt = 0;
        return NULL;
    } else{
        return find_fcb_r(token, fat0->id,cnt);
    }
}
void reclaim_space_fat(int cnt){
    fcb * dir = (fcb *)(myvhard + BLOCK_SIZE * cnt);
    int i;
    int length = BLOCK_SIZE;
    int flag = 0;
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
    int flag2 = 0;
    for (i = 0 ; i < length / sizeof(fcb); i++, dir++) {
        if (dir->free == UNEMPTY) {
            flag = 1;
            break;
        }
    }
    if (flag == 0){
        return;
    }
    for (int j = 0; j < BLOCK_NUM; ++j,fat0++,fat1++) {
        if(fat0->id == cnt){
            flag2 =1;
            break;
        }
    }
    if(flag2 == 0){
        return;
    }
    fat* fat0_copy = (fat *) (myvhard + BLOCK_SIZE);
    fat* fat1_copy = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
    fat0_copy = fat0_copy + cnt;
    fat1_copy = fat1_copy + cnt;
    fat0->id = fat0_copy->id;
```

```
    fat1->id = fat1_copy->id;
    fat0_copy->id = fat1_copy->id = FREE;

}
void cd_open(char *args,int mode) {
    int fd;
    char fact_path[PATH_MAX_LENGTH];
    fcb *dir;
    int i;
    int cnt;
    memset(fact_path, '\0', PATH_MAX_LENGTH);
    get_fact_path(fact_path, args);
    dir = find_fcb(fact_path,&cnt);
    if ( dir == NULL) {
        fprintf(stderr, "找不到该项目\n");
        return;
    }
    if (mode == DICTORY && dir->attribute == DATA){
        fprintf(stderr, "请不要输入文件名\n");
        return;
    }
    if (mode == DATA && dir->attribute == DICTORY){
        fprintf(stderr, "请不要输入文件夹名\n");
        return;
    }
    //检查cd是否已经在openfile_list内
    for (i = 0; i < MAX_OPENFILE; i++) {
        if (openfile_list[i].topenfile == UNASSIGNED) {
            continue;
        }
        if (!strcmp(dir->filename, openfile_list[i].open_fcb.filename) &&
            dir->first == openfile_list[i].open_fcb.first) {
            //设置打开文件位置
            curdir = i;
            memset(current_dir, '\0', sizeof(current_dir));
            strcpy(current_dir, openfile_list[i].dir);
            printf("项目名为：%s扩展名为:%s路径为:%s修改时间为：%d年%d月%d日%d时%d
分%d秒\n",
 openfile_list[curdir].open_fcb.filename,openfile_list[curdir].open_fcb.exname,o
penfile_list[curdir].dir

,openfile_list[curdir].open_fcb.time_info.tm_year+1900,openfile_list[curdir].ope
n_fcb.time_info.tm_mon+1,openfile_list[curdir].open_fcb.time_info.tm_mday,openfi
le_list[curdir].open_fcb.time_info.tm_hour,openfile_list[curdir].open_fcb.time_i
nfo.tm_min,openfile_list[curdir].open_fcb.time_info.tm_sec);
            return;
        }
    }
    //文件未打开
    fd = get_useropenfd();
    if (fd == -1) {
        fprintf(stderr, "没有多余的打开窗口,请先关闭某个窗口\n");
        return;
    }
    openfile_set(fd,dir,fact_path,0,UNMODEFIED,ASSIGNED);
    curdir = fd;
    memset(current_dir, '\0', sizeof(current_dir));
    strcpy(current_dir, openfile_list[fd].dir);
    printf("项目名为：%8s扩展名为:%-6s路径为:%s修改时间为：%d年%d月%d日%d时%d分%d秒
\n",
 openfile_list[curdir].open_fcb.filename,openfile_list[curdir].open_fcb.exname,o
penfile_list[curdir].dir
```

```
460
    ,openfile_list[curdir].open_fcb.time_info.tm_year+1900,openfile_list[curdir].ope
    n_fcb.time_info.tm_mon+1,openfile_list[curdir].open_fcb.time_info.tm_mday,openfi
    le_list[curdir].open_fcb.time_info.tm_hour,openfile_list[curdir].open_fcb.time_i
    nfo.tm_min,openfile_list[curdir].open_fcb.time_info.tm_sec);
461 }
462 fcb *find_free_space(int first){
463     int i;
464     fcb *dir = (fcb *) (myvhard + BLOCK_SIZE * first);
465     for (i = 0; i < BLOCK_SIZE / sizeof(fcb); i++, dir++) {
466         if (dir->free == EMPTY) {
467             return dir;
468         }
469     }
470     fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
471     fat0 = fat0 + first;
472     if(fat0->id == END){
473         return NULL;
474     } else{
475         return find_free_space(fat0->id);
476     }
477 }
478 void init_folder(int first, int second) {
479     int i;
480     fcb *par = (fcb *) (myvhard + BLOCK_SIZE * first);
481     fcb *cur = (fcb *) (myvhard + BLOCK_SIZE * second);
482
483     fcb_set(cur, ".", "dic", DICTORY, second, BLOCK_SIZE, UNEMPTY,0);
484     cur++;
485     fcb_set(cur, "..", "dic", DICTORY, first, par->length, UNEMPTY,0);
486     cur++;
487     for (i = 2; i < BLOCK_SIZE / sizeof(fcb); i++, cur++) {
488         cur->free = EMPTY;
489     }
490 }
491 void mkdir_create(char *args,int mode) {
492     int first,second,thrid;
493     char path[PATH_MAX_LENGTH];
494     char parpath[PATH_MAX_LENGTH], dirname[PATH_MAX_LENGTH];
495     char *end;
496     fcb *dir = NULL;
497     fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
498     int i;
499     int cnt;
500     get_fact_path(path, args);
501     end = strrchr(path, '/');
502     memset(parpath,'\0', PATH_MAX_LENGTH);
503     if (end == NULL) {
504         fprintf(stderr, "不要输入特殊字符/\n");
505         return;
506     } else {
507         strncpy(parpath, path, end - path);
508         strcpy(dirname, end + 1);
509     }
510     if (find_fcb(parpath,&cnt) == NULL) {
511         fprintf(stderr, "无法找到路径 %s\n", parpath);
512         return;
513     }
514     if (find_fcb(path,&cnt) != NULL) {
515         fprintf(stderr, "'%s已存在\n", args);
516         return;
517     }
518     first = find_fcb(parpath,&cnt)->first;
519     fat0 = fat0 + first;
520     second = fat0->id;
```

```
521     while (second != END){
522         fat0 = (fat *) (myvhard + BLOCK_SIZE);
523         fat0 = fat0 + second;
524         second = fat0->id;
525     }
526     dir = find_free_space(first);
527     if(dir == NULL){
528         second = get_free();
529         if(second !=-1){
530             fat_allocate(second,1);
531             fat0->id = second;
532             fcb *cur = (fcb *) (myvhard + BLOCK_SIZE * second);
533             for (i = 0; i < BLOCK_SIZE / sizeof(fcb); i++, cur++) {
534                 cur->free = 0;
535             }
536             find_fcb(parpath,&cnt)->length += BLOCK_SIZE;
537         } else{
538             fprintf(stderr, "系统全部写满\n");
539             return;
540         }
541         dir = (fcb *) (myvhard + second);
542     }
543     thrid= get_free();
544     if(thrid !=-1){
545         fat_allocate(thrid,1);
546     } else{
547         fprintf(stderr, "系统全部写满\n");
548         return;
549     }
550     if(mode == DICTORY){
551         fcb_set(dir, dirname, "dic", DICTORY, thrid, BLOCK_SIZE, UNEMPTY,0);
552         init_folder(first, thrid);
553     } else{
554         char *token = strtok(dirname, ".");
555         char exname[8];
556         char filename[8];
557         strncpy(filename, token, 8);
558         token = strtok(NULL, ".");
559         if (token != NULL) {
560             strncpy(exname, token, 4);
561         } else {
562             strncpy(exname, "dat", 3);
563         }
564         fcb_set(dir, filename, exname, DATA, thrid, BLOCK_SIZE, UNEMPTY,0);
565     }
566 }
567 void rmdir_rm(char *args, int mode){
568     int i, j;
569     fcb *dir;
570     fcb *dir_start;
571     int first;
572     int cnt;
573     char path[PATH_MAX_LENGTH];
574     get_fact_path(path, args);
575     i = (int )strlen(path);
576     //检查..,.不能被删除
577     if(path[i-1] == '.'){
578         fprintf(stderr, "没有权限删除.和..\n");
579         return ;
580     }
581     if (!strcmp(path, "\\")) {
582         fprintf(stderr, "没有权限删除根目录\n");
583         return ;
584     }
585     dir = find_fcb(path,&cnt);
```

```
    if(mode == DICTORY){
        if (dir == NULL) {
            fprintf(stderr, "找不到文件夹%s\n", path);
            return ;
        }
        if (dir->attribute == DATA) {
            fprintf(stderr, "请使用rm指令删除文件%s\n", path);
            return ;
        }
    } else{
        if (dir == NULL) {
            fprintf(stderr, "找不到文件%s\n", path);
            return ;
        }
        if (dir->attribute == DICTORY) {
            fprintf(stderr, "请使用rmdir指令删除文件夹%s\n", path);
            return ;
        }
    }
    //查看该项目是否被打开
    for (j = 0; j < MAX_OPENFILE; j++) {
        if (openfile_list[j].topenfile == UNASSIGNED) {
            continue;
        }
        if (!strcmp(dir->filename, openfile_list[j].open_fcb.filename) &&
            dir->first == openfile_list[j].open_fcb.first) {
            fprintf(stderr, "请先关闭%s,再删除\n", path);
            return;
        }
    }
    first = dir->first;
    if(mode == DICTORY){
        dir_start = (fcb *) (myvhard + BLOCK_SIZE * first);
        dir_start++;
        dir_start++;
        if(dir_start->free == UNEMPTY){
            fprintf(stderr, "文件夹%s中还存在文件\n", path);
            return;
        }
        dir->free = EMPTY;
        reclaim_space_fat(cnt);
        dir_start = (fcb *) (myvhard + BLOCK_SIZE * first);
        dir_start->free = EMPTY;
        dir_start++;
        dir_start->free = EMPTY;
        reclaim_space(first);
    } else{
        dir->free = EMPTY;
        reclaim_space(first);
    }
}
void reclaim_space(int first){
    fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
    fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
    fat0 = fat0 + first;
    fat1 = fat1 + first;
    int offset;
    while (fat0->id != END) {
        offset = fat0->id - first;
        first = fat0->id;
        fat0->id = FREE;
        fat1->id = FREE;
        fat0 += offset;
        fat1 += offset;
    }
```

```
651     fat0->id = FREE;
652     fat1->id = FREE;
653 
654 }
655 void ls(){
656     int first = openfile_list[curdir].open_fcb.first;
657     fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
658     fcb *dir = (fcb *) (myvhard + BLOCK_SIZE * first);
659     int i;
660     fat0 = fat0 + first;
661     block0 *init_block = (block0 *) myvhard;
662     if (openfile_list[curdir].open_fcb.attribute == DATA){
663         fprintf(stderr, "'%s'不是一个目录\n",
    openfile_list[curdir].open_fcb.filename);
664         return ;
665     }
666     while (fat0->id != END){
667         dir = (fcb *) (myvhard + BLOCK_SIZE * first);
668         for (i = 0; i < BLOCK_SIZE / sizeof(fcb); i++, dir++)  {
669             if (dir->free == EMPTY) {
670                 continue;
671             }
672             printf("项目名为：%s,扩展名为:%s,属性为%c,长度为%ld,修改时间为：%d年%d
    月%d日%d时%d分%d秒\n",  dir->filename,dir->exname,dir->attribute,dir->length,
673                    dir->time_info.tm_year+1900,dir->time_info.tm_mon+1,dir-
    >time_info.tm_mday,dir->time_info.tm_hour,dir->time_info.tm_min,dir-
    >time_info.tm_sec);
674         }
675         first = fat0->id;
676         fat0 = (fat *) (myvhard + BLOCK_SIZE);
677         fat0 = fat0 + first;
678     }
679     dir = (fcb *) (myvhard + BLOCK_SIZE * first);
680     for (i = 0; i < BLOCK_SIZE / sizeof(fcb); i++, dir++)  {
681         if (dir->free == EMPTY) {
682             continue;
683         }
684         printf("项目名为：%s扩展名为:%s属性为%c长度为%ld个磁盘块修改时间为：%d年%d
    月%d日%d时%d分%d秒\n",  dir->filename,dir->exname,dir->attribute,dir->length,
685                dir->time_info.tm_year+1900,dir->time_info.tm_mon+1,dir-
    >time_info.tm_mday,dir->time_info.tm_hour,dir->time_info.tm_min,dir-
    >time_info.tm_sec);
686     }
687 
688 }
689 void close(int fd){
690     int cur;
691     if (openfile_list[fd].topenfile == UNASSIGNED){
692         fprintf(stderr, "这个文件已处于关闭状态\n");
693         return;
694     }
695     if (!strcmp(openfile_list[fd].dir,"\\")){
696         fprintf(stderr, "根目录无法关闭\n");
697         return;
698     }
699     if (fd == curdir){
700         curdir = 0;
701         strcpy(current_dir, openfile_list[curdir].dir);
702     }
703     if (openfile_list[fd].fcb_state == 1) {
704         fcb_copy(find_fcb(openfile_list[fd].dir,&cur),
    &openfile_list[fd].open_fcb);
705     }
706     openfile_list[fd].topenfile = UNASSIGNED;
707 }
```

```
708  void write(int des_fd,int mode){
709      char aa;
710      int len = 0;
711      int str_len = 0;
712      int len_sum;
713      int str_start;
714      printf("请输入要写入字节数\n");
715      scanf("%d",&str_len);
716      char* str = NULL;
717      int cur;
718      if(mode == 2){
719          printf("请输入读写指针的位置\n");
720          scanf("%d",&cur);
721      } else{
722          cur = openfile_list[des_fd].count;
723      }
724      if(mode == 1){
725          str = malloc(str_len);
726          len_sum = str_len;
727      } else if(mode == 2){
728          if((openfile_list[des_fd].open_fcb.length - cur) > str_len){
729              len_sum = openfile_list[des_fd].open_fcb.length - cur + str_len;
730              str_start = len_sum - cur - str_len;
731          } else{
732              len_sum = str_len + cur;
733              str_start = len_sum;
734          }
735          str = malloc(len_sum*2);
736      } else{
737          len_sum = openfile_list[des_fd].open_fcb.length + str_len;
738          str = malloc(len_sum*2);
739      }
740      if (openfile_list[des_fd].open_fcb.attribute == DICTORY){
741          fprintf(stderr, "无法写入文件夹\n");
742          return ;
743      }
744      if (openfile_list[des_fd].topenfile == UNASSIGNED){
745          fprintf(stderr, "文件未打开\n");
746          return ;
747      }
748      if(mode == 1){
749          memset(str, '\0',str_len);
750          getchar();
751          printf("要随机填充吗？\n");
752          if((aa=getchar())=='y'){
753              for (int i = 0; i < str_len; ++i) {
754                  str[i] = (i%10)+'0';
755              }
756              getchar();
757          } else{
758              getchar();
759              while (scanf("%c",&aa)!=EOF){
760                  str[len] = aa;
761                  len++;
762                  if(len>=str_len){
763                      break;
764                  }
765              }
766          }
767      } else if (mode == 2){
768          openfile_list[des_fd].open_fcb.len = len_sum;
769          openfile_list[des_fd].count = 0;
770          memset(str, '\0',len_sum*2);
771          do_read(des_fd,cur,str);
772          char *input = malloc(str_len*2);
```

```
773             memset(input, '\0',str_len*2);
774             getchar();
775             printf("要随机填充吗？\n");
776             if((aa=getchar())=='y'){
777                 int i;
778                 for ( i = 0; i < str_len; ++i) {
779                     input[i] = (i%10)+'0';
780                 }
781                 input[i] = '\0';
782                 getchar();
783             } else{
784                 getchar();
785                 while (scanf("%c",&aa)!=EOF){
786                     input[len] = aa;
787                     len++;
788                     if(len>=str_len){
789                         break;
790                     }
791                 }
792             }
793             char *input_end = malloc(2*(len_sum-str_start));
794             openfile_list[des_fd].count = str_start;
795             do_read(des_fd,len_sum-str_start,input_end);
796             strcat(str,input);
797             free(input);
798             strcat(str,input_end);
799             free(input_end);
800         } else{
801             openfile_list[des_fd].count = 0;
802             memset(str, '\0',len_sum*2);
803             do_read(des_fd,openfile_list[des_fd].open_fcb.len,str);
804             openfile_list[des_fd].open_fcb.len = len_sum;
805             char *input = malloc(str_len*2);
806             memset(input, '\0',str_len*2);
807             getchar();
808             printf("要随机填充吗？\n");
809             if((aa=getchar())=='y'){
810                 int i;
811                 for ( i = 0; i < str_len; ++i) {
812                     input[i] = (i%10)+'0';
813                 }
814                 input[i] = '\0';
815                 getchar();
816             } else{
817                 getchar();
818                 while (scanf("%c",&aa)!=EOF){
819                     input[len] = aa;
820                     len++;
821                     if(len>=str_len){
822                         break;
823                     }
824                 }
825             }
826             strcat(str,input);
827             free(input);
828         }
829         do_write(des_fd,str,len_sum,mode);
830     }
831     void do_write(int des_fd,char *str, int len,int mode){
832         int first = openfile_list[des_fd].open_fcb.first;
833         fat *fat0 = (fat *) (myvhard + BLOCK_SIZE);
834         fat *fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
835         reclaim_space(first);
836         int cur;
837         int block_num = (int)((double)len/BLOCK_SIZE + 0.5);
```

```
838         int write = get_free();
839         first = write;
840         if (write == -1){
841             fprintf(stderr, "系统已无空闲磁盘块\n");
842             fat0 = (fat *) (myvhard + BLOCK_SIZE);
843             fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
844             memcpy(fat0, fat1, FAT_NUM*BLOCK_SIZE);
845         }
846         memcpy(myvhard+write*(BLOCK_SIZE),str,BLOCK_SIZE);
847         fcb *dir = find_fcb(openfile_list[des_fd].dir,&cur);
848         dir->first = write;
849         int old = write;
850         int i = 1;
851         fat0 = fat0 + old;
852         fat0->id = END;
853         block_num--;
854         while (i <= block_num){
855             write = get_free();
856             if (write == -1){
857                 fprintf(stderr, "系统已无空闲磁盘块\n");
858                 fat0 = (fat *) (myvhard + BLOCK_SIZE);
859                 fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
860                 memcpy(fat0, fat1, FAT_NUM*BLOCK_SIZE);
861             }
862             fat0->id = write;
863             fat0 = (fat *) (myvhard + BLOCK_SIZE);
864             fat0 = fat0 + write;
865             fat0->id = END;
866             memcpy(myvhard+write*(BLOCK_SIZE),str+BLOCK_SIZE*i,BLOCK_SIZE);
867             i++;
868         }
869         fat0 = (fat *) (myvhard + BLOCK_SIZE);
870         fat1 = (fat *) (myvhard + BLOCK_SIZE * FAT_NUM + BLOCK_SIZE);
871         memcpy(fat1, fat0, FAT_NUM*BLOCK_SIZE);
872         dir->length = (int)((double)len/BLOCK_SIZE + 0.5)*BLOCK_SIZE;
873         dir->len = len;
874         openfile_set(des_fd,dir,openfile_list[des_fd].dir,0,UNMODEFIED,ASSIGNED);
875     }
876     int do_read(int des_fd,int len,char *text){
877         if((openfile_list[des_fd].count + len) > openfile_list[des_fd].open_fcb.len)
    {
878             fprintf(stderr, "读取超过文件总长\n");
879             return -1;
880         }
881         // 计算逻辑块号和块内偏移量
882         int block_num = openfile_list[des_fd].count / BLOCK_SIZE;
883         int block_offset = openfile_list[des_fd].count % BLOCK_SIZE;
884         char buf[BLOCK_SIZE];
885         // 得到物理块号
886         fat* fat1 = (fat*)(myvhard + BLOCK_SIZE);
887         int cur_fat = openfile_list[des_fd].open_fcb.first;
888         fat* fat_ptr = fat1 + openfile_list[des_fd].open_fcb.first;
889         for(int i = 0; i < block_num; i++){
890             cur_fat = fat_ptr->id;
891             fat_ptr = fat1 + fat_ptr->id;
892         }
893         // 开始读取
894         int cnt = 0;
895         while(cnt < len){
896             unsigned char* pos = (unsigned char*)(myvhard + BLOCK_SIZE*cur_fat);
897             for(int i = 0; i < BLOCK_SIZE; ++i){
898                 buf[i] = pos[i];
899             }
900             for(; block_offset < BLOCK_SIZE; ++block_offset){
901                 text[cnt] = buf[block_offset];
```

```
            ++cnt;
            openfile_list[des_fd].count++;
            if(cnt == len){
                break;
            }
        }
        if(cnt < len){
            cur_fat = fat_ptr->id;
            fat_ptr = fat1 + fat_ptr->id;
            block_offset = 0;
        }
    }
    text[cnt] = '\0';
    return cnt;
}
void read(int des_fd){
    if(des_fd < 0 || des_fd >= MAX_OPENFILE){
        fprintf(stderr, "文件描述符错误\n");
        return;
    }
    int start;
    printf("请输入文件读取开始位置\n");
    scanf("%d",&start);
    int len;
    printf("请输入读取长度\n");
    scanf("%d",&len);
    char *text = malloc(len*2);
    if (openfile_list[des_fd].open_fcb.attribute == DICTORY){
        fprintf(stderr, "无法读入文件夹\n");
        return ;
    }
    if (openfile_list[des_fd].topenfile == UNASSIGNED){
        fprintf(stderr, "文件未打开\n");
        return ;
    }
    if (len > openfile_list[des_fd].open_fcb.len){
        fprintf(stderr, "读入长度大于文件总长\n");
        return ;
    }
    openfile_list[des_fd].count = start;
    int cnt = do_read(des_fd,  len, text);
    if(cnt == -1){
        fprintf(stderr, "读取文件错误\n");
        return ;
    }else{
        printf("%s\n", text);
        printf("共读取 %d B\n", cnt);
        getchar();
    }
}
void my_exitsys(){
    FILE *fp = fopen(PATH, "w");
    fwrite(myvhard, SIZE, 1, fp);
    fclose(fp);
    free(myvhard);
}
void pwd(){
    printf("当前打开文件/文件夹表为\n");
    int i;
    for (i = 0; i < MAX_OPENFILE; i++) {
        if (openfile_list[i].topenfile == UNASSIGNED) {
            continue;
        } else{
```

```
965             printf("第%d项,项目名为：%s扩展名为:%s路径为:%s修改时间为：%d年%d月%d
    日%d时%d分%d秒\n",
     i,openfile_list[i].open_fcb.filename,openfile_list[i].open_fcb.exname,openfile_
    list[i].dir
966
    ,openfile_list[i].open_fcb.time_info.tm_year+1900,openfile_list[i].open_fcb.time
    _info.tm_mon+1,openfile_list[i].open_fcb.time_info.tm_mday,openfile_list[i].open
    _fcb.time_info.tm_hour,openfile_list[i].open_fcb.time_info.tm_min,openfile_list[
    i].open_fcb.time_info.tm_sec);
967         }
968     }
969 }
970 void printfat(){
971     fat* fat1 = (fat*)(myvhard + BLOCK_SIZE);
972     int cnt = 0;
973     for(int i = 0; i < SIZE / BLOCK_SIZE; ++i){
974         printf("%04x ",fat1->id);
975         ++fat1;
976         ++cnt;
977         if(cnt % 16 == 0){
978             printf("\n");
979             cnt = 0;
980         }
981     }
982     printf("\n");
983 }
984 int main(){
985     startsys();
986     printf("文件系统初始化成功\n");
987     while(1){
988         printf("%s> ", current_dir);
989         char yes;
990         char command[100] = "";
991         fgets(command, sizeof(command), stdin);
992         command[strlen(command) - 1] = '\0';
993         if(strncmp(command, "format", 6) == 0){
994            printf("是否更改磁盘大小,将重新初始化，丢失文件数据。\n");
995             if((yes=getchar())=='y'){
996                 format();
997                 startsys();
998             }
999         }
1000        else if(strncmp(command, "cd", 2) == 0){
1001            char *dirname = command + 3;
1002            cd_open(dirname,DICTORY);
1003        }else if(strncmp(command, "mkdir", 5) == 0){
1004            char *dirname = command + 6;
1005            mkdir_create(dirname,DICTORY);
1006        }
1007        else if(strncmp(command, "rmdir", 5) == 0){
1008            char *dirname = command + 6;
1009            rmdir_rm(dirname,DICTORY);
1010        }
1011        else if(strncmp(command, "ls", 2) == 0){
1012            ls();
1013        }
1014        else if(strncmp(command, "create", 6) == 0){
1015            char *dirname = command + 7;
1016            mkdir_create(dirname,DATA);
1017        }
1018        else if(strncmp(command, "rm", 2) == 0){
1019            char *dirname = command + 3;
1020            rmdir_rm(dirname,DATA);
1021        }
1022        else if(strncmp(command, "open", 4) == 0){
```

```
1023                char *dirname = command + 5;
1024                cd_open(dirname,DATA);
1025            }
1026            else if(strncmp(command, "close", 5) == 0){
1027                pwd();
1028                printf("输入想要关闭的项号\n");
1029                int dir;
1030                scanf("%d",&dir);
1031                getchar();
1032                close(dir);
1033            }
1034            else if(strncmp(command, "write", 5) == 0){
1035                int mode;
1036                printf("请选择写入模式(1.截断2.覆盖3.追加)\n");
1037                scanf("%d",&mode);
1038                pwd();
1039                printf("输入想要写入的项号\n");
1040                int dir;
1041                scanf("%d",&dir);
1042                write(dir,mode);
1043            }
1044            else if(strncmp(command, "read", 4) == 0){
1045                pwd();
1046                printf("输入想要读入的项号\n");
1047                int dir;
1048                scanf("%d",&dir);
1049                read(dir);
1050            }
1051            else if(strncmp(command, "printfat", 8) == 0){
1052                printfat();
1053            }
1054            else if(strncmp(command, "exit", 4) == 0){
1055                my_exitsys();
1056                return 0;
1057            }
1058            else{
1059                printf("命令错误！\n");
1060            }
1061        }
1062    }
1063
```