
文件系统实现报告

作者:周泽勇

2011-12-10

引言:我们的操作系统每天都在产生大量的临时文件.而我们并不知情.这些文件用完以后就想垃圾一样拖慢我们系统的速度.而磁盘这种慢速设备也会让我们的系统变得迟钝.我一我们需要一种基于内存的小"磁盘"来保存这些临时文件.我们所做的并非真正的文件系统,因为我们的东西并不真正的存到类似硬盘,U 盘之类的存储介质上去,因为不会写驱动程序,而是申请一固定大小的内存空间,把他当成是硬盘之类的存储介质,将其格式化成文件系统,进行一系列的文件存储读取等操作.

(一)预备知识,知道的可以直接无视

<I> 磁盘块

磁盘中存储信息的最小单位是扇区,一个扇区是 512B(一个自己八位就不用我说了吧,大家都是学计算机的),但是在实际的文件系统中应为觉得这个单位实在是太小了, 所以决定把几个扇区绑在一块使用.通常是把 8 个扇区作为一个基本单位来使用.也就是 8*512=4K 个字节,512 就是 0.5K 嘛.所以在我们的文件系统中也就也用了这种方式,但是我们用不了设置的那么大,所以我们把最小的存储单位设置为 1k(1024B),也就是一个磁盘块或者说叫做簇.我们的文件系统总的空间就是 1000 个这种最小单位,也就是 1000 个磁盘块即 1K*1000 = 1M,因为 1000 略小于 1024,所以大小是不到 4M 的,window 下显示是 0.97M.

<II>FAT16

接下来介绍一下 FAT16 的相关知识

Fat16 的组织形式

引导扇区	FAT1	FAT2(重复的)	根文件夹	其他文件夹及所有文件	剩余扇区
1 扇区	*实际情况取大小*	*同 FAT1*	*32 个扇区*	*开始簇编号(从 2 开始)*	*不足一簇*

引导扇区是硬盘、软盘或类似的数据存储设备的一个扇区，内含负责启动（booting）"存放在碟片（disk）的其他部份的程序（通常，但不必然是操作系统）"的机器码。(来自维基百科)

引导扇区(Boot Sector) 通常指设备的第一个扇区,用于加载并转让处理器控制权给操作系统.(来自百度百科想要继续了解的话可以点击去看看)

我们也设置了引导扇区,但是引导扇区对于我们并没有用,但是我们也不能完全把这段空间浪费掉.我

们把用来存储我们自己的一些信息.具体见 format 函数解析，这里就不展开了.

紧随其后的就是两张 FAT 表,这两张表是一摸一样的,FAT2 完全就是 FAT1 的备份,这样设置是为了防止 FAT1 由于各种情况损害后恢复用的.有关于 FAT 表的作用我就不废话了,不知道的去看下书吧.

(二)代码的布局设计

代码没用使用单个 C 文件的书写方式.这是因为代码量有点大,如果全部卸载一个 C 文件中的话就会变得非常长,代码查找和查看都会变的非常不方便,所以这里采用了多文件设置工程.废话不多,介绍下各个文件的作用吧

<I>头文件

const.h	定义了工程中所有用到的常量定义
fat16.h	定义了工程中所有的变量定义和全局变量,因为都是用来描述 fat16 的,所以名字就 fat16 了
kernel.h	定义了工程中所有用到的函数
stack.h	工程中用到了堆栈,这个是堆栈的定义

(II)源文件

main.c	主调函数所在文件
sys.c	完成文件系统的初始化以及结束的功能
disk.c	只有一个函数,就是格式化磁盘的函数,建议从这里开始看,我也是从这里开始写的,所以没有调用其他的函数
dir.c	完成目录处理的函数都在这个文件里
file.c	完成文件处理的函数全在这里
stack.c	堆栈处理的具体实现

(三)代码解析

const.h

```
#ifndef _CONST_H
#define _CONST_H

#define BLOCKSIZE   1024 /* the size of disk block */
#define VERSION     0.01
#define SIZE    1024000 /* the size of virtual disk */
#define END         65535 /* the flag whitch means end of file in fat*/
#define FREE    0 /* the flag whitch means the disk block is free */
#define ROOTBLOCKNUM    5 /* the number of root block  */
#define MAXOPENFILE 10 /* the maximum number to open file in the same time
*/

#endif //_CONST_H
```

```
#ifndef _CONST_H
#define _CONST_H
#endif //_CONST_H
```

这几个宏定义是为了防止重定义,因为头文件会经常被包含来包含去,如果在同一个文件里直接或间接

的包含了同一个头文件的话,那么该头文件中的内容就被重定义了,为了消除这中错误,我们就需要加这么几句宏定义,代码解释过来就是 if not define _CONST_H then define _CONST_H and end if 翻译成中文就是如果没有定义_CONST_H那么就定义_CONST_H,定义结束(文件尾)关于_CONST_H的命名规范是_+文件名大写+_后缀名大写

下面几句都已经给出了解释,翻译过来就是(本人的不成形的英语你们看起来可能会很蛋疼)

```
#define BLOCKSIZE 1024     一个磁盘块的大小
#define VERSION   0.01     文件系统的版本
#define SIZE      1024000  文件系统所用到的总的虚拟磁盘(即内存)的大小
#define END       65535    一个文件结束的标志,用于标志 fat 的
#define FREE      0        标明该 fat 是没有使用的
#define ROOTBLOCKNUM  5    根目录所在的磁盘块号
#define MAXOPENFILE   10   能够打开的最大的文件或目录数
```

fat16.h

```
#include "const.h"
#ifndef _FAT16_H
#define _FAT16_H

/* file control block*/
#ifndef __FS_FAT16_FCB__
#define __FS_FAT16_FCB__
typedef struct FCB{
 char filename[11];/*file name include extend file name*/
 unsigned char attribute;/*file attribution*/
 unsigned char other[10];
 unsigned short time;/*file create time*/
 unsigned short date;/*file create data*/
 unsigned short first;/*file start disk block*/
 unsigned int length;/*file length*/
}fcb;
#endif//__FS_FAT16_FCB__

/* file allocation table */
#ifndef __FS_FAT16_FAT__
#define __FS_FAT16_FAT__
typedef struct FAT{
 unsigned short id;
}fat;
#endif//__FS_FAT16_FAT__

/* user open table*/
#ifndef __FS_FAT16_USEROPEN__
```

```
#define __FS_FAT16_USEROPEN__
typedef struct USEROPEN{
 char filename[11];/*file name include extend file name*/
 unsigned char attribute;/*file attribution*/
 unsigned short time;/*file create time*/
 unsigned short date;/*file create data*/
 unsigned short first;/*file start disk block*/
 unsigned short length;/*file length*/
 char free;/*mark whether this directory is empty, 0 means empty, 1 means
assigned*/
 /* previous content is fcb, in the next recode the directory of it's parent's
that opened */
 int dirno; /*the disk block id of the directory of the opened file in it's
parent's */
 int diroff; /*the index id of the disk block id of the directory of the opened
file in it's parent's*/
 char dir[MAXOPENFILE][80]; /* the directory of opened file, so as to check
whether the file is opened faster*/
 int count; /* the position of pointer in the file */
 char fcbstate; /* whether modified the content of fcb, if modified set 1,
or set 0*/
 char topenfile; /* wheather opened table item is empty, if the value is 0
empty, or occupied */
}useropen;
#endif//__FS_FAT16_USEROPEN__

#ifndef __FS_FAT16_BLOCK0__
#define __FS_FAT16_BLOCK0__
typedef struct BLOCK0 {
 char name[20];/* name of the file system */
 float version;
 int blocksize;
 int size;
 int maxopenfile;
 unsigned short root;
}block0;
#endif //__FS_FAT16_BLOCK0__

unsigned char *vhard; /* pointer to the start of vertual disk*/
useropen openfilelist[MAXOPENFILE]; /* array of user open file list*/
int fileopenptr;
useropen ptrcurdir; /*pointer to current user open list*/
char currentdir[80]; /*recode cureent dir (include path)*/
unsigned char* startp; /* recode the start position in data area of vertual
```

```
disk*/

#endif //_FAT16_H
```

有关于这个文件中各个变量我就不做说明,注释里有.

kernel.h

```
/* kernel.h contain all the function prototypes */
#ifndef _KERNEL_H
#define _KERNEL_H
/* system function prototypes */
void startsys();
void existsys();

/* disk function prototypes */
void format();

/* directory function prototypes */
void mkdir(char *dirname);
void cd(char *dirname);
void deldir(char *dirname);
void ls();

/* file function prototypes */
int create(char *filename);
void delfile(char *filename);
int open(char filename);
void close(int fid);
int write(int fid);
int dowrite(int fid, char *text, int len, char wstyle);
int read(int fid);
#endif
```

这里包含了所有函数的定义,主要分为四种处理函数:系统相关函数,虚拟磁盘相关函数,目录处理函数,文件处理函数.
系统相关函数主要是两个,一个是系统开始时所做的系统初始化函数,还有一个是就是在系统结束时所做的清理和保存工作.
虚拟磁盘相关的函数就写了一个,那就是磁盘格式化.
目录的操作主要包括创建目录,切换目录, 删除目录, 列出目录.
文件处理主要包括创建文件,删除文件,打开文件, 关闭文件,写文件和读文件.这里 write 和 dowrite 两个函数共同完成了些的任务.

disk.h

```
/*
 *      fs/fs/disk.c
```

```
 *
 *      (C) zhou.zheyong
 */
#include "kernel.h"
#include "fat16.h"

#include <time.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>


void format(){
    FILE * myfsys;
    time_t *now;
    struct tm *current;
    block0 *boot;
    fat *fat1, *fat2;
    fcb *root;
    unsigned char * p = vhard;
    int i=0;
```

p 指针最开始指向虚拟磁盘的首地址

```
    boot = ((block0 *)p);

    strcpy(boot->name, "zzycami file system\0");
    boot->version = VERSION;
    boot->blocksize = BLOCKSIZE;
    boot->size = SIZE;
    boot->maxopenfile = MAXOPENFILE;
    boot->root = 5;

    printf("\nWelcome to %s\nversion:%1.2f\nblock size:%d\ntotal size:%d\nmax
opened files:%d\nroot block location:%u\n\n", boot->name, boot->version,
boot->blocksize, boot->size, boot->maxopenfile, boot->root);
```

在函数中我们实例化了一个 boot 参数,我们随后对其进行相应的赋值,就是刚才所讲的这些信息,然后对这些信息进行显示.你还可以定义你自己的信息(修改 BLOCK0 结构体,增加 boot 的赋值语句就行了)只要中的大小不要超过一个磁盘块就行了.

fat 表的布局很简单,因为是初始化,所以除了标号为 0-5 的这六个表项不能用之外(第一个磁盘块用来保存引导块了,1234 号磁盘块用来保存 fat 表自己了,5 号磁盘块用来保存根目录了,所以这六个表项赋值为 END),后面的表项全部赋值为 FREE

```
    //build two fat
    p += BLOCKSIZE;
```

因为引导块占一个字节所以在引导块结束后p指针向后移动一个磁盘块

```
    fat1 = ((fat *)p);
    fat2 = ((fat *)(p + 2*BLOCKSIZE));
    //first five block is used
    for(i=0;i<5;i++){
        fat1->id = END;
        fat2->id = END;
        fat1 ++;
        fat2 ++;
    }
    fat1->id = 5;
    fat2->id = 5;
    fat1 ++;
    fat2 ++;
    for(i = 6;i<SIZE/BLOCKSIZE;i++){
        fat1->id = FREE;
        fat2->id = FREE;
        fat1 ++;
        fat2 ++;
    }
```

每个磁盘块占两个字节,所以在定义完两张 fat 表以后要向后移动 2*2 个磁盘块

接下来就是根目录的布局了,对于每一个新建的目录都需要默认创建两个名字为.和..的这两个目录(就是这两个表示目录的 fcb),分别表示当前目录和上一级目录.特别说明一下目录的大小问题(fcb->length)我的做法可能比较随便,对于一个 fcb,如果这个 fcb 对应的是一个文件,那么在文件空的时候他的长度就是一个磁盘块的长度,但是如果文件不为空了,那么就是他真是的文件长度,如果这个 fcb 多对应的是一个目录,那么这个 fcb->length 就保存这个目录下 fcb 的个数.但是对于.和..我就没做具体规定了,我的程序中就是随便赋值的.

```
    p += 4*BLOCKSIZE;
    root = (fcb *)p;
    strcpy(root->filename, ".");
    root->attribute = 0x4;

    now = (time_t*)malloc(sizeof(time_t));
    time(now);
    current = localtime(now);
    root->date = ((current->tm_year-80)<<9) + ((current->tm_mon +1)<<5) +
current->tm_mday;
    root->time = (current->tm_hour<<11) + (current->tm_min<<5)
+(current->tm_sec>>1);
    root->first = 5;
    root->length = 2;

    root ++;
```

```
    strcpy(root->filename, "..");
    root->attribute = 0x4;
    time(now);
    current = localtime(now);
    root->date = ((current->tm_year-80)<<9) + ((current->tm_mon +1)<<5) +
current->tm_mday;
    root->time = (current->tm_hour<<11) + (current->tm_min<<5)
+(current->tm_sec>>1);
    root->first = 5;
    root->length = 1;
    root ++;

    for(i=2;i<BLOCKSIZE/sizeof(fcb);i++){
        root->filename[0] = '\0';
        root ++;
    }
```

最后保存刚刚创建的整个文件系统的内存,那么 format 的工作就结束了.

```
    myfsys = fopen("myfsys", "w");
    fwrite(vhard, SIZE, 1, myfsys);
    fclose(myfsys);
}
```

这个函数是做磁盘的格式化的,是非常基础也是非常重要的一个函数,他文成了 fat16 格式在磁盘上的布局工作.
函数开始就开始创建磁盘引导扇区的工作,前面讲过这个引导扇区在我们的文件系统中没有什么用,所以我们在这个扇区写入我们自己所需要的东西.我定义了一个结构体(参见 fat16.h)

```
#ifndef __FS_FAT16_BLOCK0__
#define __FS_FAT16_BLOCK0__
typedef struct BLOCK0 {
    char name[20];/* name of the file system */
    float version;
    int blocksize;
    int size;
    int maxopenfile;
    unsigned short root;
}block0;
#endif //__FS_FAT16_BLOCK0__
```

这个就是我们定义的引导扇区所保存的信息.包括系统的名字,系统的版本,一个磁盘块的大小,总共的容量,最大可以打开的目录或文件以及根目录所在的磁盘序号.
在函数中我们实例化了一个 boot 参数,我们随后对其进行相应的赋值,就是刚才所讲的这些信息,然后对这

些信息进行显示.你还可以定义你自己的信息(修改 BLOCK0 结构体,增加 boot 的赋值语句就行了)只要中的大小不要超过一个磁盘块就行了.

接下来就是对两个 fat 表的定义,首先让我们计算一下为什么一张 fat 表要占两个磁盘块,fat 表中每一项对应一个磁盘块,我们总共有 1000 个磁盘块,也就是说我们的 fat 表的表项中共至少需要 1000 个表项,而我们一个表项是占两个字节(参见 fat16.h 中对 fat 的定义),而一个磁盘块是占 1024 个字节,也就是说一个磁盘块可以保存 1024/2=512 个字节 512<1000,所以我们一个磁盘块保存不了 100 个表项,所以我们至少需要两个磁盘块来保存 fat 表
fat 表的布局很简单,因为是初始化,所以除了标号为 0-5 的这六个表项不能用之外(第一个磁盘块用来保存引导块了,1234 号磁盘块用来保存 fat 表自己了,5 号磁盘块用来保存根目录了,所以这六个表项赋值为 END),后面的表项全部赋值为 FREE
接下来就是根目录的布局了,对于每一个新建的目录都需要默认创建两个名字为.和..的这两个目录(就是这两个表示目录的 fcb),分别表示当前目录和上一级目录.特别说明一下目录的大小问题(fcb->length)我的做法可能比较随便,对于一个 fcb,如果这个 fcb 对应的是一个文件,那么在文件空的时候他的长度就是一个磁盘块的长度,但是如果文件不为空了,那么就是他真是的文件长度,如果这个 fcb 多对应的是一个目录,那么这个 fcb->length 就保存这个目录下 fcb 的个数.但是对于.和..我就没做具体规定了,我的程序中就是随便赋值的.

这里要特别提一下这个系统里时间的设置方法.
首先了解一下 C 语言中有关于时间的用法, unix 时间戳,这是一个通用的标准,不管是 C 语言也好,PHP 也好都是用这个方式来定义时间的,具体介绍参见(百度百科),简而言之就是是从 1970 年 1 月 1 日(UTC/GMT 的午夜) 开始所经过的秒数，不考虑闰秒。C 语言中使用 localtime 函数可以将这样一个时间戳转换为年月日小时分秒等信息,C 中保存 Unix 时间戳的是 time_t 这样一个类型的变量

```
#ifndef _TIME_T_DEFINED
typedef long time_t; /* time value */
#define _TIME_T_DEFINED /* avoid multiple defines of time_t */
#endif
```

上面就是 time_t 的定义.使用 time 函数可以得到当前的时间戳.那么 C 中用来保存年月日信息的类型是一个叫 struct tm 的结构体,定义如下:

```
#ifndef _TM_DEFINED
    struct tm {
    int tm_sec; /* 秒–取值区间为[0,59] */
    int tm_min; /* 分 - 取值区间为[0,59] */
    int tm_hour; /* 时 - 取值区间为[0,23] */
    int tm_mday; /* 一个月中的日期 - 取值区间为[1,31] */
    int tm_mon; /* 月份（从一月开始，0 代表一月） - 取值区间为[0,11] */
    int tm_year; /* 年份，其值从 1900 开始 */
    int tm_wday; /* 星期–取值区间为[0,6]，其中 0 代表星期天，1 代表星期一，以此类推 */
    int tm_yday; /* 从每年的 1 月 1 日开始的天数–取值区间为[0,365]，其中 0 代表 1 月 1 日，1 代表 1 月 2 日，以此类推 */
    int tm_isdst; /* 夏令时标识符，实行夏令时的时候，tm_isdst 为正。不实行夏令时的进候，tm_isdst 为 0；不了解情况时，tm_isdst()为负。*/
    };
    #define _TM_DEFINED
```

```
    #endif
```

那么我们的程序中 fcb->date 用来保存年月日,fcb->time 用来保存小时分秒(定义参见 fat16.h 中),他们都只有 2 个字节也就是 16 位.我们这里就使用了位运算来保存时间.对于年月日的日范围是[0,31]一个月总没有 32 天吧.所以我们用 5 位就可以保存这个数了(2^5=32>31).我们按照顺序保存,日就放在 16 位的最后 5 位.同理月的话有 12 个月 4 位就够了因为最低 5 位已经访日了所以我们需要向右移动 5 位才能放在日的前面,剩下 7 位([0,127])全部用来放年,应为年是从 1970 年开始的,所以如果是 2011 年的话,因该保存 31.我们为了能够在 7 位里保存更多的数我们减了 80 然后把它向右移动 9 位放在最高位.
接下来就是在 16 位中保存小时分秒了,我们可以计算一下小时[0-24],分[0-60],秒[0-60]分别需要 5 位 6 位,6 位来保存,总共 17 位,我们 16 位保存不下.所以我们就把其中一个数以二了,这里是秒,你也可以选其它的,剩下的设置方式和年月日的设置方法一样.

最后保存刚刚创建的整个文件系统的内存,那么 format 的工作就结束了.

sys.c

```
/*
 *      fs/fs/sys.c
 *
 *      (C) zhou.zheyong
 */
#include "kernel.h"
#include "fat16.h"

#include <malloc.h>
#include <stdio.h>
#include <string.h>


void startsys(){
    FILE *myfsys;
    fcb *root;
    block0 *boot;

    vhard = (unsigned char*)malloc(SIZE);
    memset(vhard,0,SIZE);
    if(myfsys = fopen("myfsys","r")){
        fread(vhard,SIZE, 1, myfsys);
        boot = (block0 *)vhard;
        printf("\nWelcome to %s\nversion:%1.2f\nblock size:%d\ntotal size:%d\nmax
opened files:%d\nroot block location:%u\n\n", boot->name, boot->version,
boot->blocksize, boot->size, boot->maxopenfile, boot->root);
        fclose(myfsys);
    }else {
        printf("myfsys is not exist, now create it\n");
```

```
        format();
        boot = (block0 *)vhard;
    }

    root = (fcb *)(vhard + boot->root*BLOCKSIZE);
    strcpy(openfilelist[0].filename, "root");
    openfilelist[0].attribute = 0x4;
    openfilelist[0].time = ((fcb *)(vhard + 5*BLOCKSIZE))->time;
    openfilelist[0].date = ((fcb *)(vhard + 5*BLOCKSIZE))->date;
    openfilelist[0].first = ((fcb *)(vhard + 5*BLOCKSIZE))->first;
    /* the root is a directory, so the length means how many fcb this directory
contained */
    openfilelist[0].length = ((fcb *)(vhard + 5*BLOCKSIZE))->length;
    /*while(strcmp(root->filename, "")){
        openfilelist[0].length ++;
        root ++;
    }*/
    openfilelist[0].count = 1;
    openfilelist[0].dirno = 5;
    openfilelist[0].diroff = 0;
    openfilelist[0].fcbstate = 0;
    strcpy(currentdir, "/root");
    strcpy(openfilelist[0].dir[0], currentdir);
    openfilelist[0].free = 1;
    openfilelist[0].topenfile = 0;
    ptrcurdir = openfilelist[0];
    fileopenptr = 0;
}

void existsys(){
    FILE *myfsys;
    myfsys = fopen("myfsys", "w");
    fwrite(vhard,SIZE,1,myfsys);
    free(vhard);
    fclose(myfsys);
    printf("exit sucessed! bye!\n");
}
```

这个文件里的两个函数是做系统初始化和系统退出时所用到的函数

系统初始化时先对 vhard 这个全局变量申请 1000*1024 长度的空间用来表示我们所管理的这个虚拟磁盘的存储空间,如果之前保存过这段虚拟磁盘的文件映像的话我们就直接读入.如果没有的话就调用 format 格式化磁盘重新建立一个虚拟磁盘.之后就是打开根目录,打开方法很简单,填写以下 openfilelist 这个用户打开文件表的结构体实例就行了.

退出函数的话就跟简单了,吧我们的操作玩的 vhard 所指的长度为 1000*1024 的这段虚拟空间保存成文件

就行了.

dir.c

这个文件的代码就比较多了,我就不贴代码了,不然这个文档就太大了.你可以参照文档来理解.
ls函数里面就涉及到前面所讲的时间的恢复.具体不难,就是做一个逆运算而已.

mkdir创建目录的函数需要注意想要对输入的目录名进行严格的检测,比如这个文件名不能是/|这样的字符,而且不能在当前目录下重名等等,一旦发现错误就给出错误提示并退出程序创建失败,如果都通过的话就在当前目录下新建一个目录的fcb,填写这个fcb,设置相应的fat表,分配新的磁盘块,然后在新分配的磁盘块内新建两个.和..的目录,之前说过如何建立,不会的会看看前面的文档.

cd函数的话不难,但是挺麻烦的.首先先把输入的cd目录分解成一个字符串数组,每一项表示一级目录,然后再解析当前所在目录,当前所在目录记录在currentdirc这个全局变量里.解析当前目录主要是为了找到所要切换的目的目录的父级目录.然后更具所解析得到的数据去尝试查找这个目录,如果成功的话修改相应参数就行了.

deldir这个函数有两个重点,一个还是对所删除对象的名字要进过审核,通过之后才能删除.在删除是就需要遍历树形的目录结构.这里有两种方法可以做这个树形遍历,一个是写一个遍历的递归函数,另一个方法就是用栈来做树形遍历,这个遍历在数据结构中学过,不会的话可以去翻旧书.

file.c

文件的操作和目录插补多.区别这个fcb是文件还是目录的方法就是在fcb->attribute的值不一样,在真正的fat16系统中属性是可以叠加的,比如一个文件既可以是系统文件也可以同时具有隐藏的属性,实现这个的方法就在与属性的赋值是采用位表示法,顾名思义就是哪一个位表示什么属性比如attribute是一个字节的有八位00000010就表示它是系统的00001000就表示他是隐藏的,那么00000010 | 00001000=00001010就表示他是系统隐藏属性的,就是说什么位代表设么属性,因为我们的系统没有那么多要求我们就单纯的0x5表示文件 0x4表示目录.
文件处理还有一个和目录不同的就是文件是要存数据的,当然数据是存储在磁盘块里的,但是一个磁盘块只有1024个字节,但是文件有可能大于1024个字节呀,所以我们就需要用到fat表了比如一个文件有2050个字节就需要3块磁盘块了,第一块1024B,第二块1024B, 第三块2B,那么fcb表中记录了保存这个文件的第一个磁盘块号,比如说是561块,那么第二块记录的地方就写在fat表中底516这个表项中,比如说这个表项中记录的是342块,那么第二块记录这个文件的磁盘块就在342块中,最后一块的话一样查第342个fat表项,比如找到是456块,那么在456块磁盘块中就记录了两个字节,剩下的1022个字节就是快内碎片,没办法利用了,到这里这个文件就保存完了.所以在这个fat表项中就填写END表示文件结束.在我们的程序中END是65535(2^{16}-1).

main.c

最后的最后就是main函数了,我想大家都看的懂.我就不多说了.