

计算机硬件基础

磁盘模拟报告

章敏捷, 计科, 3130102283

September 18, 2014

目 录

1	交互界面与应用操作	2
1.1	基本功能	2
1.2	附加功能	4
2	代码与算法	6
2.1	程序结构简介	6
2.2	底层驱动	6
2.3	文件系统简介	8
2.3.1	头文件	8
2.3.2	缓存机制	10
2.3.3	FAT表操作	11
2.3.4	目录操作	14
2.4	交互界面简介	16
3	总结	19
3.1	感谢	19
3.2	版权	19

Chapter 1

交互界面与应用操作

交互启动时自动打开磁盘管理便于挂载查看，被磁盘管理挂载时请不要进行读写操作。本程序交互模仿命令行，部分功能需调用系统程序。

1.1 基本功能

q 退出程序
? 显示帮助信息
fi0 加载模拟磁盘文件从扇区读取系统信息
ll 读取当前目录
cd <path> 改变目录同时输出子目录
dd 0 <扇区> 查看扇区(支持二进制0b 和16进制0x)
ds 查看当前系统簇与扇区偏移量
fc 创建空文件
fr 删除文件¹

¹*注意以上两个文件操作只针对根目录

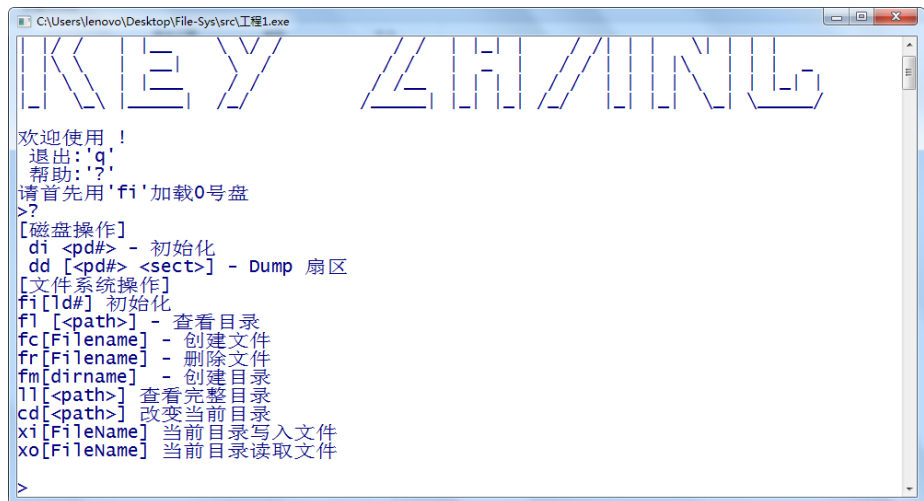


图 1.1: 帮助界面



图 1.2: 读取目录

```
>dd 0 248
Drive:0 Sector:248
00000000: 46 46 20 20 20 20 20 20 20 20 10 00 00 00 00 FF .....
00000010: 00 00 00 00 00 00 00 C2 A4 31 45 1B 00 00 00 00 .....1E.....
00000020: 4D 41 49 4E 20 20 20 20 43 20 20 20 00 00 00 00 MAIN C ....
00000030: 00 00 00 00 00 00 00 37 9D 31 45 02 00 01 31 00 00 .....7.1E...1..
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

图 1.3: Dump

```

>fczmj
writed s:248 c:1
>ll
文件名      目录扇区      大小      日期
FF\         248         0         2014 9 17
MAIN.C      248         12545      2014 9 17
ZMJ         248         0         2014 9 18
>

```

图 1.4: 创建文件

```

>ll
文件名      目录扇区      大小      日期
FF\         248         0         2014 9 17
ZMJ         248         0         2014 9 18
>dd 0 248
Drive:0 Sector:248
00000000: 46 46 20 20 20 20 20 20 20 20 20 20 10 00 00 00 00 FF
00000010: 00 00 00 00 00 00 00 C2 A4 31 45 1B 00 00 00 00 00 .....1E.....
00000020: E5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000040: 5A 4D 4A 20 20 20 20 20 20 20 20 20 20 00 00 3B A4 ZMJ
00000050: 32 45 00 00 00 00 00 3B A4 32 45 00 00 00 00 00 00 2E....;.2E.....
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

图 1.5: 执行 frmain.c 后

1.2 附加功能

xi <Filename> 从外部写入磁盘

xo <Filename> 从磁盘读取到data文件夹下

xp <Filename> 用画图打开

xt <Filename> 用记事本打开²

²此功能调用系统仅供试用

```

文件名      目录扇区      大小      日期
TEST.C      305         15361      2014 9 17
>dd 0 305
Drive:0 Sector:305
00000000: 2E 20 20 20 20 20 20 20 20 20 20 20 20 10 00 00 00 00 .
00000010: 00 00 00 00 00 00 00 C2 A4 31 45 1B 00 00 00 00 00 .....1E.....
00000020: 2E 2E 20 20 20 20 20 20 20 20 20 20 20 10 00 00 00 00 ..
00000030: 00 00 00 00 00 00 00 C2 A4 31 45 00 00 00 00 00 00 .....1E.....
00000040: 54 45 53 54 20 20 20 20 43 20 20 20 20 00 00 00 00 TEST C
00000050: 00 00 00 00 00 00 00 D4 B4 31 45 5A 00 01 3C 00 00 .....1EZ.<..
00000060: E5 46 20 20 20 20 20 20 20 48 20 20 20 00 00 00 00 .F H
00000070: 00 00 00 00 00 00 00 99 A5 32 45 02 00 9F 32 00 00 .....2E...2..
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

图 1.6: 之前

```

>dd 0 305
Drive:0 Sector:305
00000000: 2E 20 20 20 20 20 20 20 20 20 20 10 00 00 00 00 .
00000010: 00 00 00 00 00 00 C2 A4 31 45 1B 00 00 00 00 00 .....1E.....
00000020: 2E 2E 20 20 20 20 20 20 20 20 20 10 00 00 00 00 ..
00000030: 00 00 00 00 00 00 C2 A4 31 45 00 00 00 00 00 00 .....1E.....
00000040: 54 45 53 54 20 20 20 20 43 20 20 20 00 00 00 00 TEST C ....
00000050: 00 00 00 00 00 00 D4 B4 31 45 5A 00 01 3C 00 00 .....1EZ..<..
00000060: E5 46 20 20 20 20 20 20 48 20 20 20 00 00 00 00 .F H ....
00000070: 00 00 00 00 00 00 99 A5 32 45 02 00 9F 32 00 00 .....2E...2..
00000080: 46 46 20 20 20 20 20 20 48 20 20 20 00 00 00 00 FF H ....
00000090: 00 00 00 00 00 00 D2 A5 32 45 7A 00 9F 32 00 00 .....2Ez..2..
000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

图 1.7: xiff.h之后

Chapter 2

代码与算法

注意：本程序主要代码额外存放于fun.c中，只用于展示，并未参与编译。

2.1 程序结构简介

本程序主要分为三层，分别为底层驱动位于diskio.c，文件系统层ff.c,上层应用main.c，并分别对应头文件。

2.2 底层驱动

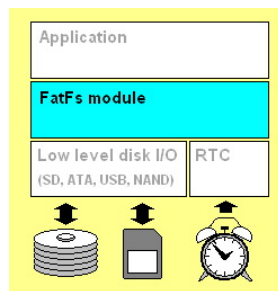


图 2.1: 程序结构

```

1  DRESULT disk_read (
2      BYTE pdrv,
3      BYTE *buff,    缓存
4      DWORD sector, 扇区编号
5      UINT count     读写数)
6  {
7      DRESULT res;
8      char a[255];
9      sprintf(a, "%d.img", pdrv);
10     fp=fopen(a, "rb+");
11     fseek(fp, sector*512, 0);
12     fread(buff, count, 512, fp);
13     fclose(fp);
14     return 0;
15 }
16
17 DRESULT disk_write (
18     BYTE pdrv,
19     const BYTE *buff,
20     DWORD sector,
21     UINT count
22 )
23 {
24     DRESULT res;
25     char a[255];
26     sprintf(a, "%d.img", pdrv);
27     fp=fopen(a, "rb+");
28     fseek(fp, sector*512, 0);
29     fwrite(buff, 512, count, fp);
30     fflush(fp);
31     fclose(fp);
32     printf("writed_s:%d_c:%d\n", sector, count);
33     return 0;
34 }
35 获取时间
36 DWORD get_fattime(){
37     int year, month, date, hour, minute, second;
38     struct tm *local;
39     time_t t;
40     t=time(NULL);
41     local=localtime(&t);
42
43     year = local->tm_year + 1900;
44     month = local->tm_mon+1;
45     date = local->tm_mday;

```



```

46     hour = local->tm_hour;
47     minute = local->tm_min;
48     second = local->tm_sec;
49
50     return ((DWORD)(year - 1980) << 25)
51         | ((DWORD)month << 21)
52         | ((DWORD)date << 16)
53         | (WORD)(hour << 11)
54         | (WORD)(minute << 5)
55         | (WORD)(second >> 1);
56 }

```

2.3 文件系统简介

2.3.1 头文件

文件系统各个函数之间的数据传递依靠以下结构和常量：

```

1
2  /* 文件系统结构 */
3  typedef struct {
4      BYTE fs_type; /* 分区类型 */
5      BYTE drv; /* 驱动器编号 (012文件名) */
6      BYTE csize; /* 单簇扇区数 1 */
7      BYTE n_fats; /* fat数 (1 or 2) */
8      BYTE wflag; /* 是否有缓存等待写入 */
9      BYTE fsi_flag; /* 是否分区表 */
10     WORD id; /* 读写标识 */
11     WORD n_rootdir; /* 根目录文件数 */
12     DWORD last_clust; /* 最后分配簇 */
13     DWORD free_clust; /* 空闲簇 */
14     DWORD n_fatent; /* fat记录类型数 (簇数+空闲+损坏) */
15     DWORD fsize; /* fat记录的扇区数 */
16     DWORD volbase; /* 卷开始扇区 */
17     DWORD fatbase; /* FAT记录开始 */
18     DWORD dirbase; /* 根目录起始扇区 */
19     DWORD database; /* 数据起始 */
20     DWORD winsect; /* 当前缓存的扇区位置 */
21     BYTE win[512]; /* 缓存 */
22 } FATFS;
23
24
25
26 /* 文件存储结构 */
27

```

```

28 typedef struct {
29     FATFS* fs;           /* 文件系统指针（指向上面的结构）*/
30     DWORD fsize;         /* 大小*/
31     DWORD sclust;        /* 起始簇*/
32     //DWORD clust;        /* 当前指针指向的簇 */
33     DWORD dsect;         /* 缓存中扇区*/
34     DWORD dir_sect;      /* 目录记录所在扇区*/
35     BYTE* dir_ptr;       /* 指向目录记录的指针*/
36     BYTE buf[512];       /* 文件读写缓存*/
37 } FIL;
38
39
40
41 /* 目录结构 */
42
43 typedef struct {
44     FATFS* fs;           /* */
45     WORD index;          /* 当前读写文件索引号*/
46     DWORD sclust;        /* 起始簇(根目录标识为0)*/
47     DWORD clust;         /* 当前簇*/
48     DWORD sect;          /* */
49     BYTE* dir;           /* 缓存中的目录记录指针*/
50     BYTE* fn;            /* 缓存中的本目录文件名指针*/
51 } DIR;
52
53
54
55 /* 文件信息结构（从目录记录读取）*/
56
57 typedef struct {
58     DWORD fsize;         /* 大小*/
59     WORD fdate;          /* 编辑日期*/
60     WORD ftime;          /* 时间*/
61     BYTE fattrib;        /* 属性*/
62     TCHAR fname[13];     /* 文件名*/
63 } FILINFO;
64 //小头转换
65 #define LD_WORD(ptr)
66     (WORD)(((WORD)*((BYTE*)(ptr)+1)<<8)|
67     (WORD)*((BYTE*)(ptr)))
68 #define LD_DWORD(ptr)
69     (DWORD)(((DWORD)*((BYTE*)(ptr)+3)<<24)|
70     ((DWORD)*((BYTE*)(ptr)+2)<<16)|
71     ((WORD)*((BYTE*)(ptr)+1)<<8)*((BYTE*)(ptr)))
72 #define ST_WORD(ptr, val)
73     *((BYTE*)(ptr))=(BYTE)(val);

```

```

74     *((BYTE*)(ptr)+1)=(BYTE)((WORD)(val)>>8)
75 #define STDWORD(ptr, val)
76     *((BYTE*)(ptr))=(BYTE)(val);
77     *((BYTE*)(ptr)+1)=(BYTE)((WORD)(val)>>8);
78     *((BYTE*)(ptr)+2)=(BYTE)((DWORD)(val)>>16);
79     *((BYTE*)(ptr)+3)=(BYTE)((DWORD)(val)>>24)
80
81
82 /* 文件属性*/
83
84 #define AMRDO 0x01 /* 只读 */
85 #define AM_HID 0x02 /* 隐藏 */
86 #define AM_SYS 0x04 /* System */
87 #define AM_VOL 0x08 /* Volume label */
88 #define AM_LFN 0x0F /* LFN entry */
89 #define AM_DIR 0x10 /* Directory */
90 #define AM_ARC 0x20 /* Archive */
91 #define AM_MASK 0x3F /* Mask of defined bits */

```

2.3.2 缓存机制

通过缓存1个扇区做到各个函数之间能够直接调用缓存而不是重新读盘：

```

1  /*缓存扇区 至内存*/
2  static
3  FRESULT move_window (
4      FATFS* fs, /* */
5      DWORD sector /* 缓存扇区编号*/
6  )
7  {
8      if (sector != fs->winsect) { /* 改变当前读写窗口*/
9          if (sync_window(fs) != FR_OK)
10             return FR_DISK_ERR;
11             if (disk_read(fs->drv, fs->win, sector, 1))
12                 return FR_DISK_ERR;
13             fs->winsect = sector;
14         }
15
16         return FR_OK;
17     }
18     /*确认缓存写入完成（同步读写窗口）*/
19     static
20     FRESULT sync_window (
21         FATFS* fs /* File system object */
22     )

```

```

23 {
24     DWORD wsect;
25     UINT nf;
26
27
28     if (fs->wflag) { /* 检查是否缓存*/
29         wsect = fs->winsect; /* 当前缓存的扇区地址*/
30         if (disk_write(fs->drv, fs->win, wsect, 1))
31             return FR_DISK_ERR;
32         fs->wflag = 0;
33         if (wsect - fs->fatbase < fs->fsize) {
34             /*判断是否在fat区*/
35             for (nf = fs->n_fats; nf >= 2; nf--) {
36                 /* 写入到其他fat拷贝*/
37                 wsect += fs->fsize;
38                 disk_write(fs->drv, fs->win, wsect, 1);
39             }
40         }
41     }
42     return FR_OK;
43 }

```

2.3.3 FAT表操作

在操作FAT表的时候经常需要计算簇和扇区对应关系以及FAT记录所在位置，本块函数主要解决FAT表读写和链的操作¹

```

1  /*从簇得扇区*/
2  DWORD clust2sect (
3      FATFS* fs ,
4      DWORD clst
5  )
6  {
7      clst -= 2;
8      if (clst >= (fs->n_fatent - 2)) return 0;
9      return clst * fs->csize + fs->database;
10 }
11 /*簇读取fat中入口并返回默认为fat16*/
12 DWORD get_fat (
13     FATFS* fs ,
14     DWORD clst
15 )
16 {

```

¹包括新建，删除以及延长

```

17     UINT wc, bc;
18     BYTE *p;
19
20
21     if (clst < 2 || clst >= fs->n_fatent)
22         return 1;
23     move_window(fs, fs->fatbase+(clst/(SS(fs)/2)));
24     /*16位的fat所以除2*/
25     p = &fs->win[clst * 2 % SS(fs)];
26     return ID_WORD(p);
27     return 0xFFFFFFFF; /* 错误 */
28 }
29
30
31 /*簇写入fat入口*/
32
33
34 FRESULT put_fat (
35     FATFS* fs,
36     DWORD clst,
37     DWORD val
38 )
39 {
40     UINT bc;
41     BYTE *p;
42     FRESULT res;
43
44
45     if (clst < 2 || clst >= fs->n_fatent) {
46         res = FR_INT_ERR;
47     } else {
48         res = move_window(fs, fs->fatbase+(clst/(SS(fs)/2)));
49         res != FR_OK;
50         p = &fs->win[clst * 2 % SS(fs)];
51         ST_WORD(p, (WORD)val); /*16位小头存储*/
52         fs->wflag = 1; /*标记缓存未写入状态*/
53         sync_window(fs);
54     }
55     return res;
56 }
57
58 /*创建链表*/
59 static
60 DWORD create_chain (
61     FATFS* fs,
62     DWORD clst

```

```

63 )
64 {
65     DWORD cs, ncl, scl;
66     FRESULT res;
67
68     if (clst == 0) { /* 创建新链 */
69         scl = fs->last_clust;
70         /* 从最前的从未分配空簇开始 */
71         if (!scl || scl >= fs->n_fatent) scl = 1;
72         /* 所有簇都已经分配过 */
73     }
74     else { /* 连接 */
75         cs = get_fat(fs, clst);
76         /* 检查开始簇是否空 */
77         if (cs < fs->n_fatent) return cs;
78         /* 已经占用 */
79         scl = clst;
80     }
81
82     ncl = scl;
83     /* 循环直到找到下一个空簇 */
84     for (;;) {
85         ncl++;
86         if (ncl >= fs->n_fatent) { /* 回到开头 */
87             ncl = 2;
88             if (ncl > scl) return 0;
89         }
90         cs = get_fat(fs, ncl);
91         if (cs == 0) break; /* 找到空簇 */
92         if (ncl == scl) return 0; /* 没了 */
93     }
94
95     res = put_fat(fs, ncl, 0x0FFFFFFF); /* 标记为文件结
96 束 */
97     if (res == FR_OK && clst != 0) {
98         res = put_fat(fs, clst, ncl); /* 与上一个连接 */
99     }
100     if (res == FR_OK) {
101         fs->last_clust = ncl; /* 更新文件系统 */
102         if (fs->free_clust != 0xFFFFFFFF) {
103             fs->free_clust--;
104             fs->fsi_flag |= 1;
105         }
106     } else {
107         ncl = (res == FR_DISK_ERR) ? 0xFFFFFFFF : 1;

```

```

108     }
109
110     return ncl;    /* 返回新簇*/
111 }
112 /*清除链表*/
113 static
114 FRESULT remove_chain (
115     FATFS* fs,
116     DWORD clst
117 )
118 {
119     FRESULT res;
120     DWORD nxt;
121
122
123     if (clst < 2 || clst >= fs->n_fatent) {
124         res = FR_INT_ERR;
125
126     } else {
127         res = FR_OK;
128         while (clst < fs->n_fatent) {
129             /* 判断是否超出范围*/
130             nxt = get_fat(fs, clst);
131             res = put_fat(fs, clst, 0);
132             /* 标记当前簇为空 */
133             if (res != FR_OK) break;
134             if (fs->free_clust != 0xFFFFFFFF) {
135                 /* 更新文件系统 增加空扇区计数*/
136                 fs->free_clust++;
137                 fs->fsi_flag |= 1;
138                 /*标记等待写入*/
139             }
140
141             clst = nxt;
142         }
143     }
144
145     return res;
146 }

```

2.3.4 目录操作

主要是目录记录的读取写入，以及我自己写的使用方案

1

```

2 //获取目录信息
3 FRESULT f_opendir (
4     DIR* dp,
5     const TCHAR* path
6 )
7 {
8     FRESULT res;
9     FATFS* fs;
10    DEFNAMEBUF;
11
12
13    if (!dp) return FR_INVALID_OBJECT;
14    if (res == FR_OK) {
15        dp->fs = fs;
16        if (res == FR_OK) {
17            if (dp->dir) {
18                if (dp->dir[DIR_Attr] & AMDIR)
19                    dp->sclust = ld_clust(fs, dp->dir);
20                else /* 目标是文件 */
21                    res = FR_NO_PATH;
22            }
23            if (res == FR_OK) {
24                dp->id = fs->id;
25                res = dir_sdi(dp, 0);
26            }
27        }
28    }
29
30 }
31
32 //获取文件信息
33 static
34 void get_fileinfo (
35     DIR* dp,
36     FILINFO* fno
37 )
38 {
39     {
40         UINT i;
41         TCHAR *p, c;
42
43
44         p = fno->fname;
45         if (dp->sect) { /* 获取名称*/
46             BYTE *dir = dp->dir; /*目录索引目标名称*/
47             i = 0;

```



```

48 while (i < 11) { /* 文件名和扩展名 */
49     c = (TCHAR) dir[i++];
50     if (c == ' ') continue; /* 跳过空格 */
51     if (i == 9) *p++ = '.'; /* 扩展名 */
52     *p++ = c;
53 }
54 fno->fattrib = dir[DIR_Attr];
55 /* 读取属性 */
56 fno->fsize = IDDWORD( dir+DIR_FileSize );
57 /* 大小 */
58 fno->fdate = IDWORD( dir+DIR_WrtDate );
59 /* Date */
60 fno->ftime = IDWORD( dir+DIR_WrtTime );
61 /* Time */
62 }
63 *p = 0; /* Terminate SFN string by a \0 文件名结
束 */
64 }
65
66 使用方案：针对区分目录的文件读写策略
67 if(dp->sclust)
68     disk_read(0, buff, clust2sect(dp->fs, dp->sclust), 1);
69 else
70     disk_read(0, buff, dp->fs->dirbase, 1);
71 先判断是否是根目录，设定缓存读取扇区
72 if(dp->sclust)
73     disk_write(0, buff, clust2sect(dp->fs, dp->sclust), 1);
74 else
75     disk_write(0, buff, dp->fs->dirbase, 1);
76 缓存写入对应的数据之后，写入对应扇区，从而避免对其它读写
缓存的影响

```

2.4 交互界面简介

交互界面采用控制台模式，通过switch-case得到命令之后继续读取得到参数从而既避免了二次读取参数又保证了后续的可升级性。下面贴出参数处理的重要函数：

```

1 //用于分离参数中的数值，并且可以识别不同进制制
2 int xatoi (
3     TCHAR **str,
4     long *res
5 )
6 {
7     unsigned long val;

```

```

8      unsigned char r, s = 0;
9      TCHAR c;

10
11
12      *res = 0;
13      while ((c = **str) == ) (*str)++;
14      //跳过空格
15
16      if (c == -) {          /* 负数 */
17          s = 1;
18          c = *(++(*str));
19      }
20
21      if (c == 0) {
22          c = *(++(*str));
23          switch (c) {
24              case x:
25                  r = 16; c = *(++(*str));
26                  break;
27              case b:
28                  r = 2; c = *(++(*str));
29                  break;
30              default:
31                  if (c <= ) return 1;
32                  if (c < 0 || c > 9) return 0;
33                  r = 8;
34              }
35          } else {
36              if (c < 0 || c > 9) return 0;
37              r = 10;
38          }
39
40      val = 0;
41      while (c > ) {
42          if (c >= a) c -= 0x20;
43          c -= 0;
44          if (c >= 17) {
45              c -= 7;
46              if (c <= 9) return 0;
47          }
48          if (c >= r) return 0;
49          val = val * r + c;
50          c = *(++(*str));
51      }
52      if (s) val = 0 - val;
53

```

```

54     *res = val;
55     return 1;
56 }
57 /*-----系统调用-----*\
58     char c[126]="mspaint_.\data\\";
59     char d[126]="mspaint_.\data\\";
60     char t[126]="notepad_.\data\\";
61     char tt[126]="notepad_.\data\\";
62     case p: xcopy(&dir,ptr); strcpy(c,d);
63     strcat(c,ptr);
64     system(c);
65     break;
66     case t: xcopy(&dir,ptr); strcpy(t,tt);
67     strcat(t,ptr);
68     system(t);
69     break;
70     \*最后突发奇想利用系统调用外部命令查看从磁盘里读取的文件并且在应用开始打开磁盘管理*\

```

Chapter 3

总结

3.1 感谢

在这次作业过程中我参考了许多开源的文件系统，在很大程度上帮助了我对于FAT文件系统的理解，并且在作品中也多少用到了他们的成果,在最后要感谢这些无私贡献的码农们。感谢：

- FatFs - FAT file system module R0.10b (C)ChaN, 2014
- cnFat Tony Yang,2007

3.2 版权

在这份代码中包含了部分基于FatFs的代码

FatFs module is a generic FAT file system module for small embedded systems. This is a free software that opened for education, research and commercial developments under license policy of following terms.

Copyright (C) 2014, ChaN, all right reserved.

The FatFs module is a free software and there is NO WARRANTY. No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY. Redistributions of source code must retain the above copyright notice.

本程序源代码公开于 <https://github.com/zmj1316/File-Sys>

更多关于FAT文件系统的文档参阅<http://msdn.microsoft.com/en-us/windows/hardware/gg463080.aspx>