

FATFS 浅谈

刚开始看到 FATFS 时,一头雾水,不知道从何下手,网上也搜了很多资料,要么高深莫测,要么简单地一笔代过.

断断续续地摸索了一段时间,算是对文件系统有了初步的认识,整理一下思路,将自己的学习过程,及学习心得写出来与大家分享,文笔有限,力求简洁易懂,希望对初学者有所帮助,不足之处请指正.

笔者用的是酷学玩 summer V1.3 的开发板,64M 的 microSD 卡(已格式化为 FAT32 格式),底层驱动是 yuanyin 移植的,如果你用的不是酷学玩 stm32 的板子,那也没关系,网上有很多移植的例程,可以参照着尝试移植;或者根据酷学玩的例程修改也可.

本文主要介绍 FATFS 的 API 函数,仅针对初学者入门,高手请拍砖.

例程中用到的全局变量定义如下:

```
FATFS fs;           // Work area (file system object) for logical drive
FIL fsrc, fdst;     // file objects
FRESULT res;        // FatFs function common result code
UINT br, bw;        // File R/W count
```

一. 注册工作区域

```
FRESULT f_mount (  
    BYTE Drive,          /* Logical drive number */  
    FATFS* FileSystemObject /* Pointer to the work area */  
);
```

函数说明:

1. 此函数的作用就是在磁盘里注册一个缓冲区域,用来存储 FAT32 文件系统的一些相关信息.
2. 参数说明:
 - a) *Drive*: 盘符
 - b) **FileSystemObject*: 指向缓冲区域的指针
3. 对磁盘进行操作之前,这个函数是不可少的

例程 : f_mount(0, &fs);

二. 打开文件夹

```
FRESULT f_opendir (  
    DIR* DirObject,      /* Pointer to the blank directory object  
structure */  
    const TCHAR* DirName /* Pointer to the directory name */  
)
```

函数说明:

1. 此函数可以打开一个已存在的文件夹
2. 参数说明:
 - a) **DirObject*: 指向一个空白的结构体, 用来存储要打开的文件夹信息
 - b) **DirName*: 指向该文件夹名称的指针

三. 读取文件夹

```
FRESULT f_readdir (  
    DIR* DirObject,      /* Pointer to the open directory object */  
    FILINFO* FileInfo /* Pointer to the file information structure */  
);
```

函数说明:

1. 此函数按照顺序读取文件夹内文件
2. 参数说明:
 - a) **DirObject*: 指向读取的文件夹的信息结构体的指针
 - b) **FileInfo*: 指向文件信息结构体, 用来存储读取到的文件的信息
3. 重复调用此函数可读取文件夹内所有文件
4. 当所有文件读取结束,函数返回一个空字符串到 *f_name[]* 中
5. 如果一个空指针赋给 **FileInfo*,将返回从第一个文件开始读取.

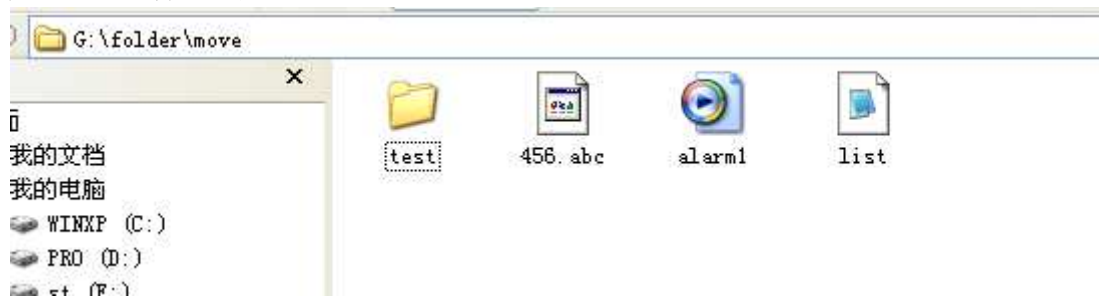
例程：这段程序先打开文件夹 `folde/move` ，然后查找其中的存档文件，并通过串口输出读取的文件名

```
if (f_opendir(&dirs, "folder/move") == FR_OK)           //打开文件夹
{
    while (f_readdir(&dirs, &finfo) == FR_OK)          //按照顺序读文件夹
    {
        if(!finfo.fname[0]) break;                     //如果文件名为 0,结束
        {
            if(finfo.fattrib == AM_ARC)                //判断文件属性
                Debug("文件名:%s\r\n",finfo.fname);
        }
    }
}
```

串口输出如下：

```
文件名:list.lis
文件名:alarm1.wav
文件名:456.abc
```

在 PC 上查看 SD 卡：



可以看到，程序输出了后面三个文件的名称，对文件夹没作处理，因为在程序里对文件属性进行了判断：

`if(finfo.fattrib == AM_ARC)`，意思是只对存档文件进行处理；

至于什么是存档文件，以下是在百度知道搜到的结果：

视窗系统中文件属性有四种类型，我来告诉你这四种类型是什么意思：

只读-表示该文件不能被修改

隐藏 -表示该文件在系统中是隐藏的，在默认情况下用户不能看见这些文件。

系统 - 表示该文件是操作系统的一部分。

存档- 表示该文件在上次备份前已经修改过了，一些备份软件在备份系统后会把这些文件默认的设为存档属性。

存档属性在一般文件管理中意义不大，但是对于频繁的文件批量管理很有帮助。

四. 打开\新建一个文件:

```
FRESULT f_open (  
    FIL* FileObject,          /* Pointer to the blank file object structure  
*/  
    const TCHAR* FileName, /* Pointer to the file name */  
    BYTE ModeFlags          /* Mode flags */  
);
```

函数说明:

1. 此函数可以打开, 或新建一个文件
2. 参数说明
 - a) *FileObject: 指向一个用来存储文件对象的空结构体的指针
 - b) *FileName: 指向文件名的指针
 - c) ModeFlags: 打开方式, 可以是以下一种或几种的组合 (默认方式是 FA_OPEN_EXISTING)

Value	Description
FA_READ	读模式, (读写模式可同时生效)
FA_WRITE	写模式, (读写模式可同时生效)
FA_OPEN_EXISTING	默认打开方式
FA_OPEN_ALWAYS	打开文件, 如果文件不存在, 则创建一个新文件; 用此种方式, 可以用 f_lseek 在文件后追加数据
FA_CREATE_NEW	新建文件, 如果文件已存在, 则新建失败
FA_CREATE_ALWAYS	新建文件, 如果文件已存在, 覆盖旧文件

五. 读取文件:

```
FRESULT f_read (  
    FIL* FileObject,          /* Pointer to the file object structure */  
    void* Buffer,             /* Pointer to the buffer to store read data */  
    UINT ByteToRead,         /* Number of bytes to read */  
    UINT* ByteRead           /* Pointer to the variable to return number of bytes  
read */  
);
```

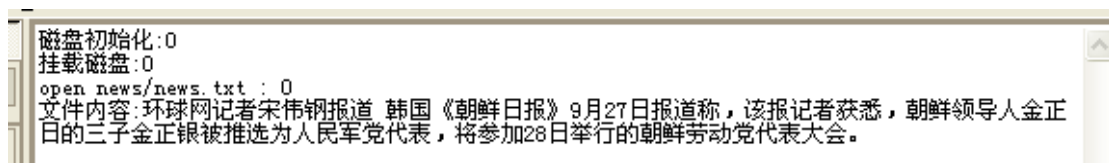
函数说明:

1. 这个函数可以读取文件的内容
2. 参数说明:
 - a) *FileObject: 指向文件对象结构体的指针
 - b) *Buffer: 指向存储读取到的数据的缓冲的指针
 - c) ByteToRead: 准备读取的字节数
 - d) *ByteRead:
 - i. 它的作用就是用来检测文件的末尾, 就是下面例程中的这一句:
if (res || br < sizeof(buffer)) break;
 - ii. 每次 f_read 执行完后, *ByteRead 值等于本次读取到的字节数, 若*ByteRead < ByteToRead, 即本次读取到的字节小于准备读取的字节, 说明读指针已到达文件末尾.

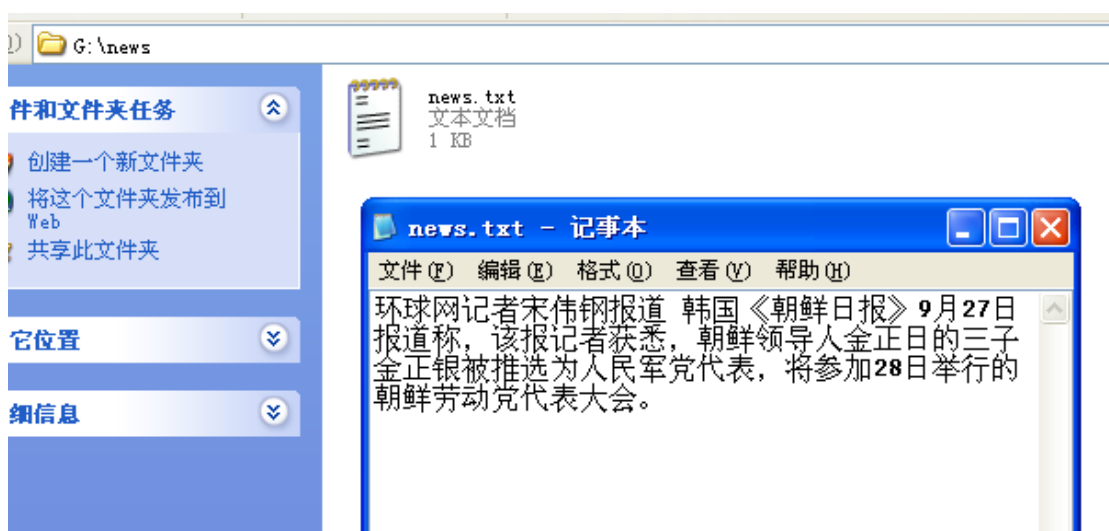
例程：此处参考酷学玩例程，以读取的方式打开文件，然后将文件内容通过串口输出。

```
res = f_open(&fsrc, "news/news.txt", FA_READ);
if(!res)
{
    Debug("open news/news.txt : %d\r\n",res);
    br=1;
    a=0;
    Debug("文件内容:");
    for (;;)
    {
        for(a=0; a<512; a++) buffer[a]=0;
        res = f_read(&fsrc, buffer, sizeof(buffer), &br);
        Debug("%s\r\n",buffer);
        if (res || br < sizeof(buffer)) break;    // error or eof
    }
}
f_close(&fsrc); //不论是打开，还是新建文件，一定记得关闭
```

运行后串口输出结果：



下面是在 P C 中查看：



六. 写文件:

```
FRESULT f_write (  
    FIL* FileObject,      /* Pointer to the file object structure */  
    const void* Buffer,   /* Pointer to the data to be written */  
    UINT ByteToWrite,    /* Number of bytes to write */  
    UINT* ByteWritten     /* Pointer to the variable to return number of  
bytes written */  
);
```

函数说明:

1. 此函数用来向文件中写入数据, 前提是以写文件的方式打开文件
2. 参数说明:
 - a) *FileObject: 指向文件对象结构体的指针
 - b) *Buffer: 指向数据缓冲的指针
 - c) ByteToWrite: 准备写入的字节数
 - d) *ByteWritten: 记录已写入的字节数, 用来检测是否写完
3. 后两个参数的长度都是两个字节, 计数值最大为 65536, 所以一次写入字节数最大为 64K。一般情况下一次不会写这么长的数据, 因为就算 RAM 足够用, 也不会在里面开一个几十 K 的数据缓冲区。

例程:

结合前面的 f_open 函数, 在下面例程中以写的方式新建一个 txt 文档, 然后写入 100 个字节。

已定义: unsigned char buffer[100] = "This is a new file, the data is just written in! 这是一个新文件, 数据也是新的!";

```
res = f_open(&fsrc, "new/NewText.txt", FA_WRITE | FA_CREATE_ALWAYS);  
if (res == FR_OK)  
{  
    Debug("create file ok!\r\n");  
    Debug("start write!\r\n");  
    do  
    {  
        res = f_write(&fsrc, buffer, 100, &bw);  
        if (res)  
        {  
            Debug("write error : %d\r\n", res);  
            break;  
        }  
        Debug("write ok!\r\n");  
    }  
    while (bw < 100); // 判断是否写完(bw > 100, 表示写入完成)  
}  
f_close(&fsrc); // 关闭文件, 必须和 f_open 函数成对出现
```

运行后串口输出:

```
磁盘初始化:0  
挂载磁盘:0  
create file ok!  
start write!  
write ok!  
|
```

下面为 P C 中查看结果:



掌握以上几个函数后, 可以利用 FATFS 对 S D 卡进行基本的读写操作了。

下面介绍另外几个常用的函数。

七. 移动文件指针:

```
FRESULT f_lseek (  
    FIL* FileObject, /* Pointer to the file object structure */  
    DWORD Offset /* File offset in unit of byte */  
);
```

函数说明:

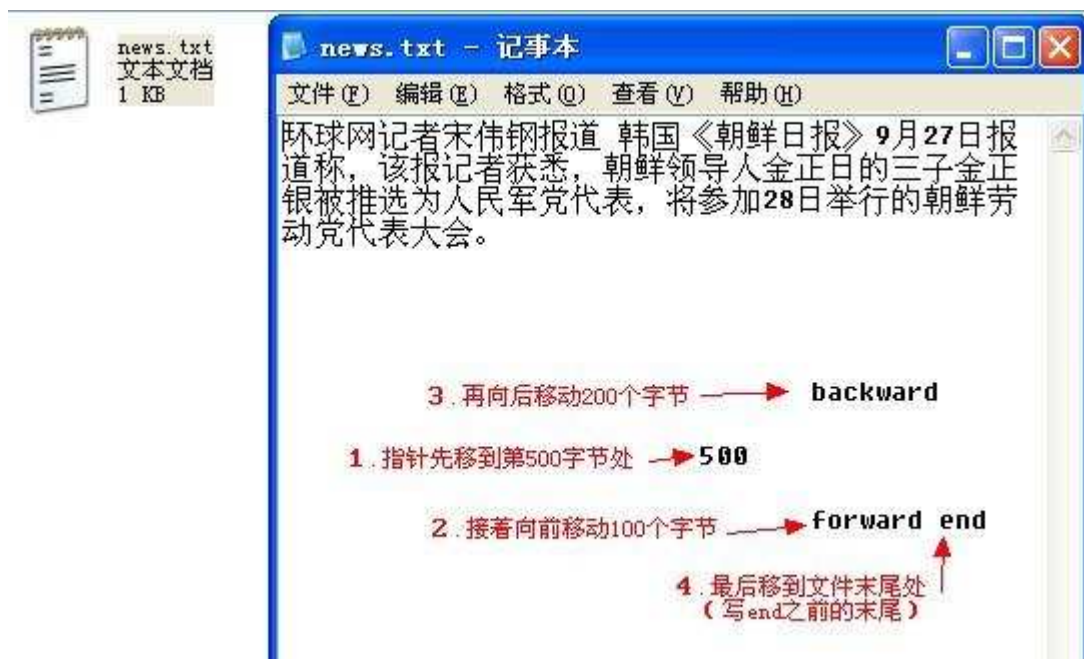
1. 此函数在对已打开的文件进行读或写时, 可以移动当前指针位置
2. 参数说明:
 - a) FileObject: 指向文件对象结构体的指针
 - b) Offset: 指针移动的长度

例程:

参考官网作者公布的例程, 本例以前文中 new/news.txt 文件为基础

```
res = f_open (&fsrc, "news/news.txt", FA_WRITE);  
  
res = f_lseek (&fsrc, 500); //指针指向第 500 个字节  
res = f_write (&fsrc, "500", 3, &bw);  
res = f_lseek (&fsrc, fsrc.fptr + 100); //指针向前移动 100 个字节  
res = f_write (&fsrc, "forward", 8, &bw);  
res = f_lseek (&fsrc, fsrc.fptr - 200); //指针向后移动 200 个字节  
res = f_write (&fsrc, "backward", 9, &bw);  
res = f_lseek (&fsrc, fsrc.fsize); //指针指向文件末尾  
res = f_write (&fsrc, "end", 3, &bw);  
  
res = f_close (&fsrc);
```

运行后在 P C 中查看结果: 红色部分为笔者注.



八. 截断文件:

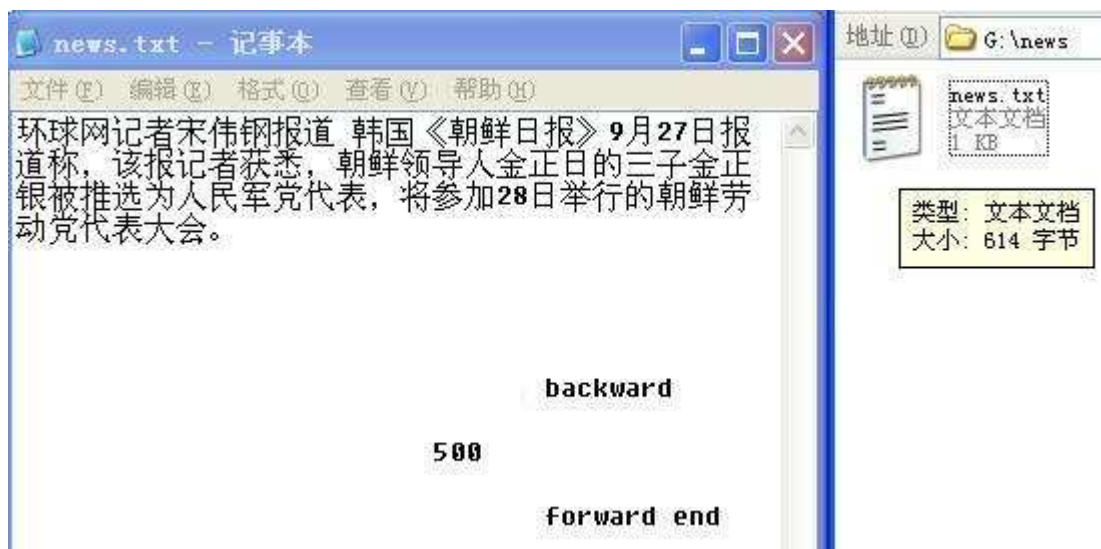
```
FRESULT f_truncate (  
    FIL* FileObject    /* Pointer to the file object */  
);
```

函数说明:

1. 此函数可以在将文件在当前指针处截断
2. 参数说明:
 - a) *FileObject: 指向文件对象结构体的指针
3. 此函数可以截断文件, 也可以延长文件长度

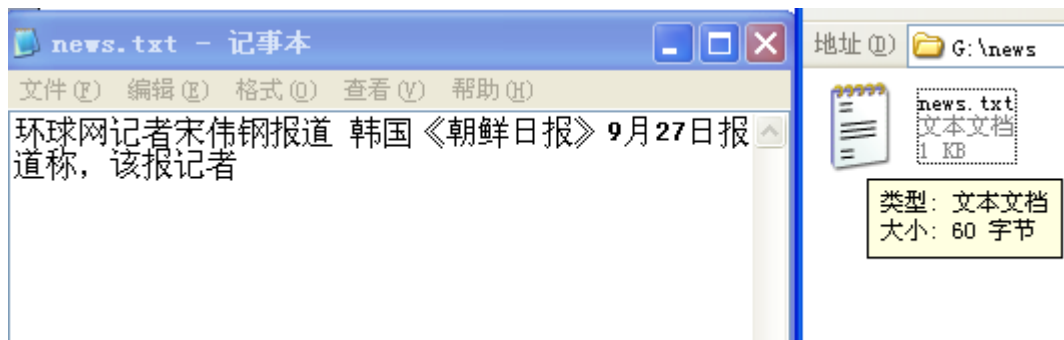
例程: 以上节 news/news.txt 为基础, 本段程序可将文件在指定长度处截断

```
res = f_open (&fsrc, "news/news.txt", FA_WRITE);  
res = f_lseek (&fsrc, 60);           ////指针指向第 60 个字节  
res = f_truncate (&fsrc);           ////将文件在此截断  
res = f_sync (&fsrc);               ////关闭文件
```



上图为运行程序之前在 P C 上查看

下图为运行程序之后, 从右面文件夹中可以看出, 文件大小变成了 60 个字节。



九. 刷新缓存信息:

```
FRESULT f_sync (  
    FIL* FileObject /* Pointer to the file object */  
);
```

函数说明:

1. 此函数功能兼容 `f_close`, 它于 `f_close` 的区别就是执行后, 当前文件是否仍然有效.
2. 参数说明:
 - a) `*FileObject`: 指向文件对象结构体的指针
3. 调用此函数后, 当前文件仍然可读可写可查询.
4. 当文件处于长时间的写模式, 如数据记录时, 定期调用此函数, 或在写入数据后立即调用此函数, 可以减少因断电等意外情况带来的损失. 有点 **WORD** 中后台定期保存的意思.

十. 新建文件夹:

```
FRESULT f_mkdir (  
    const TCHAR* DirName /* Pointer to the directory name */  
);
```

函数说明:

1. 新建一个文件夹
2. 参数说明:
 - a) `*DirName`: 指向将要创建的文件夹名的指针
3. 文件名应符合 `fatfs` 标准, 不能包含非法字符,
4. 若不支持长文件名, 文件名长度不能大于 8, 否则新建不成功
5. 例程:
 - a) `f_mkdir("new");`
 - b) `f_mkdir("folder/new");`

十一. 删除文件或文件夹:

```
FRESULT f_unlink (  
    const TCHAR* FileName /* Pointer to the object name */  
);
```

函数说明:

1. 此函数可以删除一个文件或文件夹
2. 参数说明:
 - a) `*FileName`: 指向文件或文件夹的名称的指针
3. 删除文件夹时:
 - a) 不能为当前文件夹
 - b) 不能为非空文件夹
4. 删除文件时:
 - a) 不能为已打开文件
 - b) 不能为只读文件

十二. 重命名 \ 移动文件 或文件夹

```
FRESULT f_rename (  
    const TCHAR* OldName, /* Pointer to old object name */  
    const TCHAR* NewName /* Pointer to new object name */  
);
```

函数说明:

1. 此函数可以移动或重命名一个文件或文件夹
2. 参数说明:
 - a) **OldName*: 指向旧文件名的指针
 - b) **NewName*: 指向新文件名的指针
3. 此函数可重命名 文件 或 文件夹 ,而不论文件夹是否为空
4. 此函数可移动 文件 或 文件夹 ,而不论文件夹是否为空

例程:

```
res = f_rename("folder/old.txt","folder/newname.txt");//重命名 文件,  
res = f_rename("folder/123.txt","new/456.txt");//将文件夹 folder 中的 123.txt 文件,移动到文件  
夹 new 中并重命名为 456.txt
```

十三. 获取文件信息

```
FRESULT f_stat (  
const TCHAR* FileName, /* Pointer to the file or directory name */  
FILINFO* FileInfo /* Pointer to the FILINFO structure */  
);
```

函数说明:

1. 此函数可以获取文件的最近修改时间,属性等信息,获取的信息存在 `fileinfo` 结构体中
2. 参数说明:
 - a) *FileName: 指向文件名的指针
 - b) *FileInfo: 指向保存文件信息的结构体的指针
3. 如果目标是文件夹,获取的大小为 0.
4. 此函数对根目录无效
5. 时间和日期均为两个字节,存储格式如下
 - a) 日期:
 - i. bit15...bit9: 年 减去 1980
 - ii. bit8 ... bit5: 月
 - iii. bit4 ... bit0: 日
 - b) 时间:
 - i. bit15... bit11 : 时
 - ii. bit10... bit5 : 分
 - iii. bit4 ... bit0 : 秒 除以 2
 - c) 如
 - i. 日期:00000010 00100001,表示 1981 年 1 月 1 日
 - ii. 时间:00001000 00100001,表示 1 点 1 分 2 秒

例:

```
res = f_stat("folder/newname.txt", &finfo); //读取 folder 目录下 newname.txt 文件的信息  
if( res )  
    Debug("newname.txt err : %d\r\n", res);  
else  
{  
    Debug("newname.txt size : %lu\r\n",finfo.fsize);  
    Debug("fdate : %d\r\n",finfo.fdate);  
    Debug("ftime : %d\r\n",finfo.ftime);  
    Debug("fattrib : %d\r\n",finfo.fattrib);  
}
```

串口输出结果: (红色部分为笔者注)

```
newname.txt size : 500      文件长度
fdate : 15685             最近修改日期
ftime : 36278             最近修改时间
fattrib : 36              文件属性
```

结果分析:

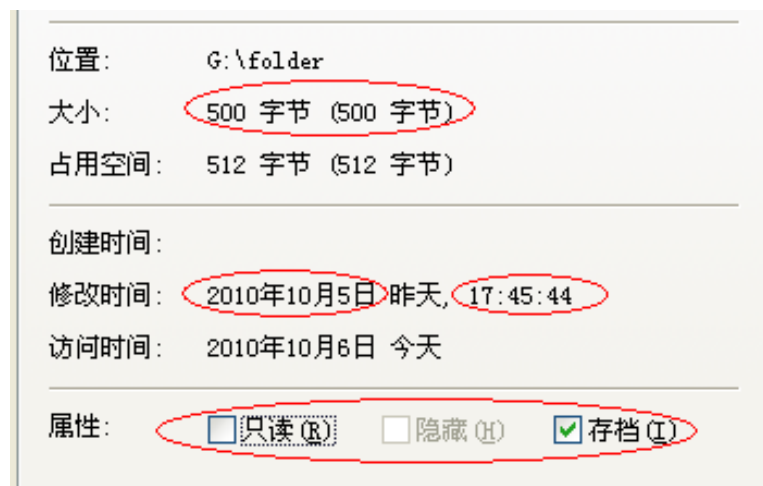
newname.txt size : 500 : 表示该文件的长度为 500 个字节
fdata : 15685 : 修改日期,转化成 2 进制: 0011110 1010 00101
按照日期储存格式,应该是 2010 年 10 月 5 日
Ftime : 36278 : 修改时间,转化成 2 进制: 10001 101101 10110
按照时间储存格式,应该是 17 点 45 分 44 秒
Fattrib : 36 : 文件属性, 按照 FATFS 中定义,如下图:
36 应该是 AM_ARC | AM_SYS, 即存档文件和系统文件

```
/* File attribute bits for directory entry */

#define AM_RDO 0x01 /* Read only */
#define AM_HID 0x02 /* Hidden */
#define AM_SYS 0x04 /* System */
#define AM_VOL 0x08 /* Volume label */
#define AM_LFN 0x0F /* LFN entry */
#define AM_DIR 0x10 /* Directory */
#define AM_ARC 0x20 /* Archive */
#define AM_MASK 0x3F /* Mask of defined bits */
```

(原文见 Ff.h)

以下为 PC 中右击属性查看的结果:



与串读出结果一致.

十四. 改变文件属性:

```
FRESULT f_chmod (  
const TCHAR* FileName, /* Pointer to the file or directory */  
BYTE Attribute,      /* Attribute flags */  
BYTE AttributeMask   /* Attribute masks */  
);
```

函数说明:

1. 此函数可以修改文件或文件夹的属性
2. 可修改的属性只能是以下一种或几种的组合, 对其它属性无效

Attribute	Description
AM_RDO	Read only
AM_ARC	Archive
AM_SYS	System
AM_HID	Hidden

3. 参数说明:

- a) *Filename: 指向文件或文件夹的名称的指针
- b) Attribute: 要置位的属性
- c) AttributeMask: 需要改变的属性 (包括要置位的和要清除的属性)

4. 使用方法:

- a) Attribute 须为 AttributeMask 的子集
- b) 函数对 AttributeMask 中的属性集合进行处理, 若属性包含在 Attribute 中, 则置位, 否则清除

例程:

1. 对文件 newname.txt, 置位 ARC 和 SYS 属性, 取消 HID 和 RDO 属性

```
res = f_chmod("folder/newname.txt", AM_ARC | AM_SYS,  AM_ARC | AM_RDO | AM_HID |  
AM_SYS);
```

```
if( res )
```

```
    Debug("err :%d\r\n", res);
```

```
else
```

```
{
```

```
    res = f_stat("folder/newname.txt", &finfo);
```

```
    Debug("%d\r\n", finfo.fattrib);
```

```
}
```

2. 对文件夹 new, 置位 SYS 和 ARC 属性, 取消 HID 和 RDO 属性

```
res = f_chmod("new", AM_SYS | AM_ARC,  AM_ARC | AM_RDO | AM_HID | AM_SYS);
```

```
if( res )
```

```
    Debug("err :%d\r\n", res);
```

```
else
```

```
{
```

```
    res = f_stat("new", &finfo);
```

```
    Debug("%d\r\n", finfo.fattrib);
```

```
}
```

十五. 改变文件时间戳

```
FRESULT f_ftime (  
    const TCHAR* FileName, /* Pointer to the file or directory path */  
    const FILINFO* TimeDate /* Time and data to be set */  
);
```

函数说明:

1. 此函数可以更改文件的最近修改时间
2. 参数说明:

- a) `Filename`:指向文件的指针
- b) `Timedate`:指向文件信息结构体的指针

3. 使用方法:

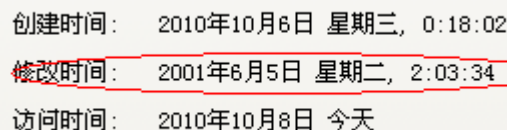
作者在官网中公布了另外一个函数 `set_timestamp`, 在这个函数里, 我们可以写入常规的日期时间, 然后此函数按日期存储格式(见前文)将数据整合后调用 `f_ftime`.

例:

```
FRESULT set_timestamp (    char *obj, /* Pointer to the file name */  
    int year,    int month,    int mday,    int hour,    int min,    int sec  
    )//  
{  
    FILINFO fno;  
  
    fno.fdate = (WORD)(((year - 1980) * 512U) | month * 32U | mday);  
    fno.ftime = (WORD)(hour * 2048U | min * 32U | sec / 2U);  
  
    return f_ftime(obj, &fno);  
}  
  
res = set_timestamp("123.txt", 2001, 06, 05, 02, 03, 34); //修改 123.txt 时间  
Debug("%d\r\n", res);
```

执行函数后, 串口返回 0

在 PC 上右键属性可以看到结果如下:



创建时间: 2010年10月6日 星期三, 0:18:02
修改时间: 2001年6月5日 星期二, 2:03:34
访问时间: 2010年10月8日 今天

先写到这，后面的函数，用到时再研究：

- [f_forward](#) - Forward file data to the stream directly
- [f_chdir](#) - Change current directory
- [f_chdrive](#) - Change current drive
- [f_getcwd](#) - Retrieve the current directory
- [f_gets](#) - Read a string
- [f_putc](#) - Write a character
- [f_puts](#) - Write a string
- [f_printf](#) - Write a formatted string

嵌入式之路还很长,写下此文,与读者共勉. 如有疑问, 请联系笔者, 共同交流, 谢谢.

邮箱: tao3236@126.com