# 实验报告(FAT文件系统)

PB17000289 于佳睿

---

## 实验目的

- 借助FUSE实现一个只读的FAT文件系统

## 实验环境

- OS：Ubuntu 14.04 i386（32位）
- Linux内核版本：Kernel 2.6.26

## 实验步骤

### 一、基础函数

#### 1. char **path_split(char *pathInput, int *pathDepth_ret)

- 函数功能：

> 这个函数的功能是将路径按'/'分割为char **型的数组，注意过长字符串的截断，空白字符的填充和小写字母的转换

- 函数实现

> 首先遍历字符串，获取'/'的个数作为深度赋值给pathDepth_ret，将第i个'/'的位置标记为start，将第i+1个'/'或者'\0'的位置标记为end，i从0循环到pathDepth_ret-1，每次申请从start到end的空间，将字符串数据转化后填入空间之中，同时观察是否有'.'，来确定存储格式.

- 代码结构

```
char **path_split(char *pathInput, int *pathDepth_ret)
{
    int pathDepth = 0;
    int i = 0, j = 0, k = 0, start = 0, end = 0, dot = 0;
    int havedot;
    while (pathInput[i] != '\0') {
        if (pathInput[i] == '/')
            pathDepth++;
        i++;
    }
```

```
        char **paths = (char **)malloc(pathDepth * sizeof(char *));
        for(j = start; j != '\0' && j != '/'; j ++){
        }
        start = j;
        for(i = 0; i != pathDepth; i ++){
            paths[i] = (char *)malloc((13-1) * sizeof(char));
            memset((void *)paths[i], 0, 13-1);
            for(j = start + 1; pathInput[j] != '/' && pathInput[j] != '\0'; j++){}
            end = j;
            havedot = 0;
            for(j = start + 1; j != end; j++){
                if(pathInput[j] == '.'){
                    havedot = 1;
                    dot = j;
                }
            }
            if(havedot == 1){  //是文件不是目录
                if((dot-start-1) > 8){  //文件名超过长度范围
                    截断
                }
                else{//文件名没有超过长度范围
                    存储文件名
                }
                if((end - dot - 1) > 3){ //扩展名超过范围
                    截断
                }
                else{
                    存储扩展名
                }
            }
            else{   //是目录不是文件
                if((end - start -1)>8){ //目录名字超过范围
                    截断
                }
                else{  //目录名字没有超过范围
                    按目录存储
                }
            }
            start = end;  //循环迭代
        }
        *pathDepth_ret = pathDepth;
        return paths;
    }
```

**2. BYTE *path_decode(BYTE *path)**

- 函数功能

  将按照指定存储格式存储的路径解码为用户态看见的路径

- 代码思路

首先观察其path[8]是否为' ', 若是, 则代表是文件则不用加点, 否则, 则代表是目录则正常读取即可

- 代码结构

```
BYTE *path_decode(BYTE *path)  //BYTE char
{
    BYTE *pathDecoded = (BYTE *)malloc(MAX_SHORT_NAME_LEN * sizeof(BYTE));
    memset((void *)pathDecoded, 0, MAX_SHORT_NAME_LEN);
    int i, end, dot;
    if(path[8] != 32){ //不是文件是目录
        读文件名
        加'.'
        读扩展名
    }
    else{  //是目录不是文件
        读文件名
    }
    return pathDecoded;
}
```

### 3. FAT16 *pre_init_fat16(void)

- 函数功能

  按照DBR扇区初始化FAT的信息

- 代码思路

  首先打开指针然后指针遍历第一个扇区, 按照DBR扇区的划分读取每一个值

```
typedef struct {
    BYTE BS_jmpBoot[3];   //跳转命令
    BYTE BS_OEMName[8];   //OEM NAME
    WORD BPB_BytsPerSec;  //每扇区512k字节
    BYTE BPB_SecPerClus;  //每簇32扇区 32*512k = 65536(2^16)
    WORD BPB_RsvdSecCnt;  //保留扇区数
    BYTE BPB_NumFATS;     //FAT数
    WORD BPB_RootEntCnt;  //Root Entery count  11
    WORD BPB_TotSec16;    //FAT占的扇区数       13
    BYTE BPB_Media;       //介质描述             15
    WORD BPB_FATSz16;     //每个FAT的扇区数     16
    WORD BPB_SecPerTrk;   //每个磁道的扇区数    18
    WORD BPB_NumHeads;    //磁头数               1a
    DWORD BPB_HiddSec;    //隐藏扇区数           1c
    DWORD BPB_TotSec32;   //NTFS是否使用         20
    BYTE BS_DrvNum;       //中断驱动器号            24
    BYTE BS_Reserved1;    //未使用               26
    BYTE BS_BootSig;      //扩展引导标记               28
    DWORD BS_VollID;      //卷序列号             28
    BYTE BS_VollLab[11];  //卷标签               2a
```

```
        BYTE BS_FilSysType[8];//FATname              36
        BYTE Reserved2[448];  //引导程序执行代码       3E
        WORD Signature_word;  //扇区结束标志          1FE
    } __attribute__ ((packed)) BPB_BS;
```

- 代码结构

```
    FAT16 *pre_init_fat16(void){
        /* Opening the FAT16 image file */
        FILE *fd;
        FAT16 *fat16_ins;

        fd = fopen(FAT_FILE_NAME, "rb");

        if (fd == NULL)
        {
            fprintf(stderr, "Missing FAT16 image file!\n");
            exit(EXIT_FAILURE);
        }
        指针读取fd指向的文件赋值给FAT16中的变量
        return fat16_ins;
    }
```

#### 4. WORD fat_entry_by_cluster(FAT16 *fat16_ins, WORD ClusterN)

- 函数功能 到FAT表中读取某簇号的下一个簇号
- 代码思路

  首先将簇号换算到扇区, 确定要找的信息在那一个扇区, 将那一个扇区读入buffer, 然后按照簇号乘2来换算索引

- 代码结构

```
    WORD fat_entry_by_cluster(FAT16 *fat16_ins, WORD ClusterN){
        BYTE sector_buffer[BYTES_PER_SECTOR];
        DWORD fat2_sector = (fat16_ins->Bpb.BPB_RsvdSecCnt + fat16_ins-
>Bpb.BPB_FATSz16);  //fat2开始扇区
        if(ClusterN < fat16_ins->Bpb.BPB_FATSz16 * fat16_ins->Bpb.BPB_BytsPerSec){
            计算簇号*2的偏移量在哪一个扇区
            读入扇区到buffer
            将簇号模每个扇区的簇号定位buffer取出FAT条目
        }
        else
            return 0xffff;
    }
```

#### 5. int find_root(FAT16 *fat16_ins, DIR_ENTRY *Dir, const char *path)

- 函数功能

  查找Path路径的文件或者目录地址

- 代码思路

  首先确定根目录级的深度，是文件还是目录, 如果是文件深度还为1，则找到，信息写入Dir, 如果深度不为1，那么返回错误;如果是目录，深度为1就找到，不为1去找子目录

- 代码结构

```
for (i = 0; i != fat16_ins->Bpb.BPB_RootEntCnt/16; i++) //遍历根目录扇区
    {
        sector_read(fat16_ins->fd, fat16_ins->FirstRootDirSecNum + i, buffer); //
读一个扇区进来
        for (j = 0; j != 16; j ++){     //遍历根目录条目
            if(strncmp((char *)(buffer + j*32), paths[0], 11) == 0){    //如果相等
找到了
                if((BYTE)buffer[j * 32 + 11] == 0x10){ //目录
                    if(pathDepth != 1){       //深度不为1寻找子目录
                        信息写入Dir
                        return find_subdir(fat16_ins, Dir, paths, pathDepth, 1);
                    }
                    else{                 //深度为1停止寻找
                        信息写入Dir
                        return 0;
                    }
                }
                else{
                    if(pathDepth!=1)   //如果深度不为1而且是文件
                        return 1;
                    信息写入Dir
                    return 0;
                }
            }
        }
    }
    return 1;
}
```

**6. find_subdir(FAT16 *fat16_ins, DIR_ENTRY *Dir, char **paths, int pathDepth, int curDepth)**

- 函数功能

  查找Path路径当前深度的目录项

- 代码思路

  遍历当前路径深度代表的子目录的所有簇, 递归寻找下一个目录

- 代码结构

```
ClusterN = Dir->DIR_FstClusLO;  //从Dir中读取索引簇号

    while(ClusterN != 0xffff){    //知道簇号为0xffff
        first_sector_by_cluster(fat16_ins, ClusterN, &FatClusEntryVal,
&FirstSectorofCluster, buffer); //读第一个扇区
        ClusterN = FatClusEntryVal;  //获取下一个簇号
        for(j = 0; j != 16; j ++){ //处理第一个扇区的16个条目
            if(buffer[j*32] == 0x00)    //如果开始为0，没有记录就结束
                return 1;
            else if(strncmp((char *)(buffer + j*32), paths[curDepth], 11) == 0){
//找到了
                if((BYTE)buffer[j * 32 + 11] == 0x10){ //目录
                    if(pathDepth != (curDepth + 1)){   //没有到底
                        信息读入Dir
                        return find_subdir(fat16_ins, Dir, paths, pathDepth,
curDepth + 1);
                    }
                    else{
                        信息读入Dir
                        return 0;
                    }
                }
                else{ //是文件
                    if(pathDepth != (curDepth + 1))
                        return 1;
                    else{
                        信息读入Dir
                        return 0;
                    }
                }
            }
        }
        for(i = 1; i != fat16_ins->Bpb.BPB_SecPerClus; i ++){     //处理簇中的其他
扇区
            sector_read(fd, FirstSectorofCluster + i, buffer); //将第i个扇区读入
buffer
            for(j = 0; j != 16; j ++){ //处理第i个扇区的16个条目
                if(buffer[j*32] == 0x00)
                    return 1;
                else if(strncpy((char *)(buffer + j*32), paths[curDepth], 11) ==
0){ //找到了
                    if((BYTE)buffer[j * 32 + 11] == 0x10){ //目录
                        if(pathDepth != (curDepth + 1)){   //没有到底
                            信息读入Dir
                            return find_subdir(fat16_ins, Dir, paths, pathDepth,
curDepth + 1);
                        }
                        else{
                            信息读入Dir
                            return 0;
                        }
                    }
                    else{ //是文件
```

```
                        if(pathDepth != (curDepth + 1))  //没有到底
                            return 1;
                        else{
                            信息读入Dir
                            return 0;
                        }
                    }
                }
            }
        }
    }
    return 1;
}
```

## 二、FUSE相关函数

### 1. int fat16_readdir(const char *path, void *buffer, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi)

- 函数功能

  这一函数用于获取指定目录下的所有条目名称

- 代码思路

  - 如果是根目录(path == "/")

    遍历根目录所有条目读取所有目录项，注意属性位是否合理，删除位是否是0Xe5

  - 如果是子目录

    根据首簇号遍历目录所在的簇，读取所有文件名，注意属性位是否合理，删除位是否是0Xe5

- 代码实现

```
if (strcmp(path, "/") == 0)    //需要显示的是根目录
  {
    DIR_ENTRY Root;
    for (i = 0; i != fat16_ins->Bpb.BPB_RootEntCnt/16; i++)           //遍历32扇区
    {
        sector_read(fat16_ins->fd, fat16_ins->FirstRootDirSecNum + i,
sector_buffer);
        for(j = 0; j != fat16_ins->Bpb.BPB_BytsPerSec/32; j ++){    //遍历一个扇区
的条目
            if(sector_buffer[32*j] != 0x00 && sector_buffer[32*j + 11] != 0x0f &&
sector_buffer[32*j + 8] != 0x00 && sector_buffer[32*j] != 0xe5){
                strncpy(Root.DIR_Name, (char *)(sector_buffer + 32*j), 11);
                const char *filename = (const char *)path_decode(Root.DIR_Name);
                filler(buffer, filename, NULL, 0);
            }
        }
    }
```

```
  }
  else                    //需要显示的是子目录
  {
    DIR_ENTRY Dir;
    find_root(fat16_ins, &Dir, path);
    WORD ClusterN, FatClusEntryVal, FirstSectorofCluster;
    ClusterN = Dir.DIR_FstClusLO; //子目录开始的首簇号
    while(ClusterN != 0xffff){    //知道簇号为0xffff
        first_sector_by_cluster(fat16_ins, ClusterN, &FatClusEntryVal,
&FirstSectorofCluster, sector_buffer); //读第一个扇区
        ClusterN = FatClusEntryVal;
        for(j = 0; j != fat16_ins->Bpb.BPB_BytsPerSec/32; j ++){   //处理该簇第一个
扇区
            if(sector_buffer[j*32] == 0x00)
                return 0;
                    if(sector_buffer[32*j + 11] != 0x0f && sector_buffer[32*j
+ 8] != 0x00 && sector_buffer[32*j] != 0xe5){
                strncpy(temp, (char *)(sector_buffer + 32*j), 11);
                const char *filename = (const char *)path_decode(temp);
                filler(buffer, filename, NULL, 0);
                    }
        }
        for(i = 1; i != fat16_ins->Bpb.BPB_SecPerClus; i ++){
            sector_read(fat16_ins->fd, FirstSectorofCluster + i ,sector_buffer);
            for(j = 0; j != fat16_ins->Bpb.BPB_BytsPerSec/32; j ++){   //处理该簇的
其他扇区
                if(sector_buffer[j*32] == 0x00)
                    return 0;
                                if(sector_buffer[32*j + 11] != 0x0f &&
sector_buffer[32*j + 8] != 0x00 && sector_buffer[32*j] != 0xe5){
                    strncpy(temp, (char *)(sector_buffer + 32*j), 11);
                    const char *filename = (const char *)path_decode(temp);
                    filler(buffer, filename, NULL, 0);
                            }
            }
        }
    }
  }
```

**2. int fat16_read(const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi)**

- 代码功能

  按path读取文件数据到buffer中

- 实现思路

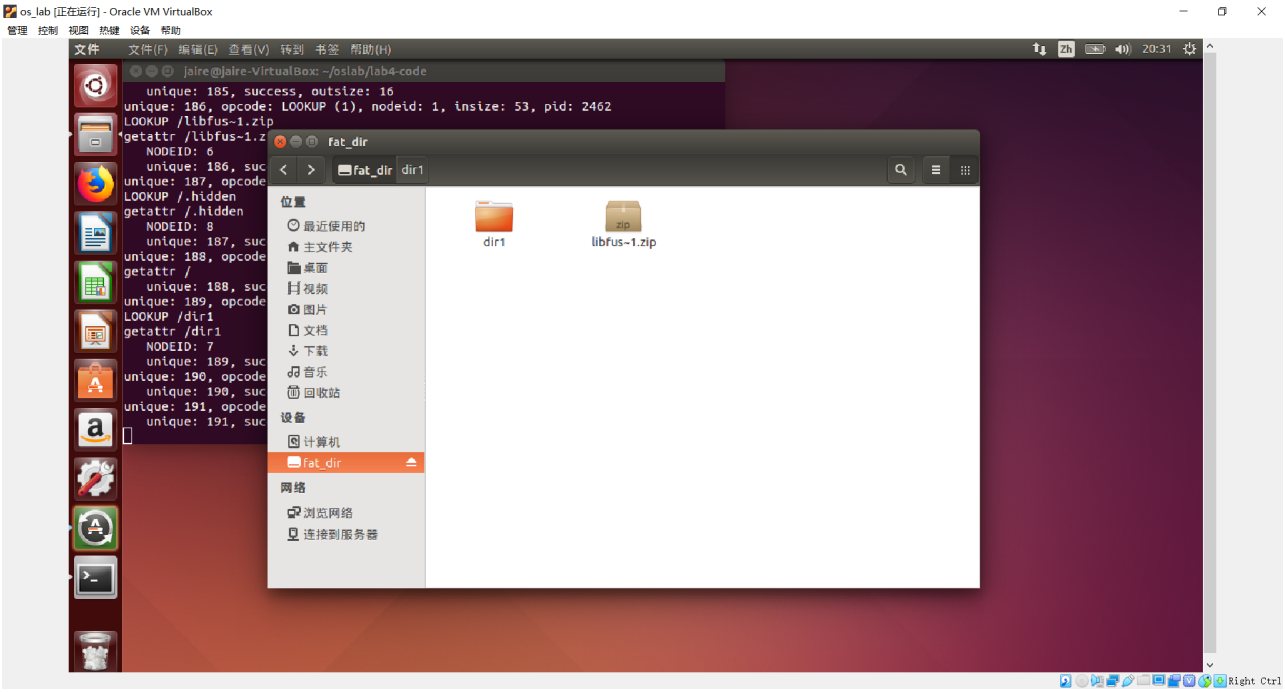
  首先find_root得到Dir信息，然后根据文件大小和偏移量计算开始簇(簇的绝对数目)，开始偏移量和结束簇，结束偏移量，用一个簇计数器计数，从首簇开始遍历簇，当计数器处于开始和结束之间时，将簇上信息读到buffer里面

- 代码实现

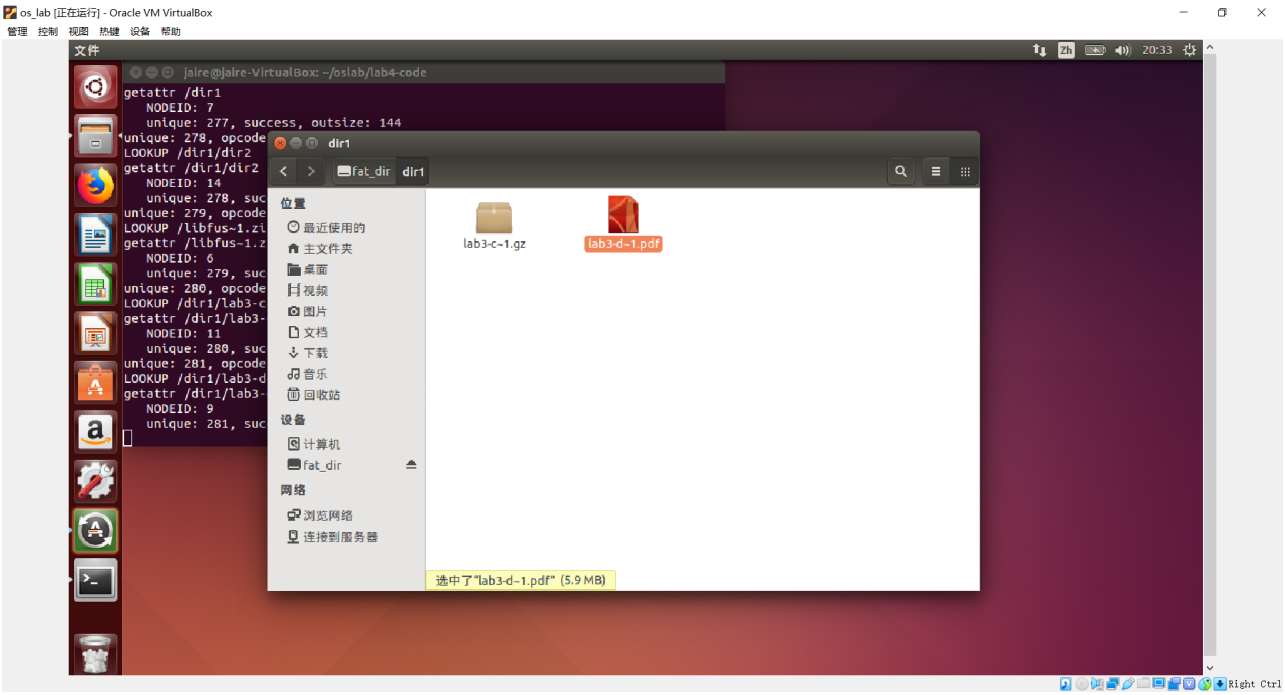
```
  if((find_root(fat16_ins, &Dir, path) == 1) || offset > Dir.DIR_FileSize){    //
没找到或者offset超过FileSize
    return 0;
  }else{
    if((offset + size) > Dir.DIR_FileSize)
        size_real = Dir.DIR_FileSize - offset;
    else
        size_real = size;
    end = size_real + offset;
    off_t BytePerClus = (fat16_ins->Bpb.BPB_BytsPerSec * fat16_ins-
>Bpb.BPB_SecPerClus);
    off_t startCluster = offset / BytePerClus;    //开始簇数
    off_t startOffset = offset % BytePerClus;  //起始簇偏移量
    off_t endCluster = (off_t)end / BytePerClus; //结束簇
    off_t endOffset = (off_t)end % BytePerClus;  //结束簇偏移量
    off_t ClusterNum = 0;           //读簇数目计数器
    off_t ByteCount = 0;            //buffer偏移计数器
    WORD ClusterN = Dir.DIR_FstClusLO;
    while(ClusterN != 0xffff){
        if(startCluster <= ClusterNum && ClusterNum <= endCluster){
            if(startCluster == ClusterNum){             //在开始簇
                fseek(fd, (long)((ClusterN-2) * BytePerClus + fat16_ins-
>FirstDataSector * fat16_ins->Bpb.BPB_BytsPerSec + startOffset), SEEK_SET);
                fread(buffer + ByteCount, (size_t)(BytePerClus - startOffset), 1
,fd);
                ByteCount = ByteCount + BytePerClus - startOffset;
            }
            else if(endCluster == ClusterNum){          //在结束簇
                fseek(fd, (long)((ClusterN-2) * BytePerClus + fat16_ins-
>FirstDataSector * fat16_ins->Bpb.BPB_BytsPerSec), SEEK_SET);
                fread(buffer + ByteCount, (size_t)(endOffset), 1 ,fd);
                ByteCount = ByteCount + endOffset;
            }
            else{                                       //在正常簇
                fseek(fd, (long)((ClusterN-2) * BytePerClus + fat16_ins-
>FirstDataSector * fat16_ins->Bpb.BPB_BytsPerSec), SEEK_SET);
                fread(buffer + ByteCount, (size_t)(BytePerClus), 1 ,fd);
                ByteCount = ByteCount + BytePerClus;
            }
        }
        else if(ClusterNum > endCluster){
            break;
        }
        ClusterN = fat_entry_by_cluster(fat16_ins, ClusterN);  //迭代信息
        ClusterNum = ClusterNum + 1;
    }
    return size_real;
  }
  return 0;
```

# 实验结果

1. 根目录




2. 子目录Dir



3. 打开PDF


# 实验总结

- 如何读多字

> 本次实验遇到了许多读多字(DWORD, WORD)的需求, 我制作了两个专用于读DWORD和WORD的函数(将 i 转化为指针然后将指针当成一个指针数组完成赋值(注意是大尾储存还是小尾储存，如果是大尾储存还要反过来)

```
DWORD get_DWORD(FILE *fp){
    BYTE *s;
    DWORD i;
    s = (BYTE *)&i;
    s[0]=getc(fp);
    s[1]=getc(fp);
    s[2]=getc(fp);
    s[3]=getc(fp);
    return i;
}
WORD get_WORD(FILE *fp){
    BYTE *s;
    WORD i;
    s = (BYTE *)&i;
    s[0] =getc(fp);
    s[1]=getc(fp);
    return i;
}
```

- DBR扇区与目录项的数据组织
    - DBR扇区

| 数据名 | 作用 | 起始位置 |
| --- | --- | --- |
| BYTE BS_jmpBoot[3] | 跳转命令 | 00 |
| BYTE BS_OEMName[8] | OEM NAME | 03 |
| WORD BPB_BytsPerSec | 每扇区512k字节 | 0b |
| BYTE BPB_SecPerClus | 每簇32扇区 32*512k = 65536(2^16) | 0d |
| WORD BPB_RsvdSecCnt | 保留扇区数 | 0e |
| BYTE BPB_NumFATS | FAT数 | 0f |
| WORD BPB_RootEntCnt | Root Entery count | 11 |
| WORD BPB_TotSec16 | FAT占的扇区数 | 13 |
| BYTE BPB_Media | 介质描述 | 15 |
| WORD BPB_FATSz16 | 每个FAT的扇区数 | 16 |
| WORD BPB_SecPerTrk | 每个磁道的扇区数 | 18 |
| WORD BPB_NumHeads | 磁头数 | 1a |
| DWORD BPB_HiddSec | 隐藏扇区数 | 1c |
| DWORD BPB_TotSec32 | NTFS是否使用 | 20 |
| BYTE BS_DrvNum | 中断驱动器号 | 24 |
| BYTE BS_Reserved1 | 未使用 | 25 |
| BYTE BS_BootSig | 扩展引导标记 | 27 |
| DWORD BS_VollID | 卷序列号 | 28 |
| BYTE BS_VollLab[11] | 卷标签 | 2a |
| BYTE BS_FilSysType[8] | FATname | 36 |
| BYTE Reserved2[448] | 引导程序执行代码 | 3E |
| WORD Signature_word | 扇区结束标志 | 1FE |

- 目录项

| 字节位置 | 长度 | 定义 | |
|---|---|---|---|
| 0x00～0x07 | 8 | 文件名 | |
| 0x08～0x0A | 3 | 扩展名 | |
| 0x0B | 1 | 属性（不可同时具有读写、只读、隐藏、系统这 4 个属性，即不能取值 0x0F） | 0x00 读写 |
| | | | 0x01 只读 |
| | | | 0x02 隐藏 |
| | | | 0x04 系统 |
| | | | 0x08 卷标 |
| | | | 0x10 目录 |
| | | | 0x20 归档 |
| 0x0C～0x15 | 10 | 系统保留<br>其中 0x0C 偏移处字节的取值请请参考偏移 0x0C 保留字节取值规则 | |

- GDB调试时的参数传递

  set arg ...

- 实验收获

通过本次实验，更加理解了通过FAT表组织的文件存储格式，了解了一些FUSE的接口，对于扇区，簇等存储大小有了直观的感受.