

操作系统实验 4

题目：FAT 文件系统的实现

PB17111623 范睿

一、 主要步骤

1. path_split()

思路：设置 head 和 tail 使 head 和 tail 把要读的字符夹住。每次读字符只读 head 和 tail 之间的内容。

1) 获取路径深度

```
for(i=0;i<length;i++) if(*(pathInput + i)=='/')pathDepth++;
```

遍历一遍路径，路径深度为“/”的个数

2) 获取各层的文件/文件夹名

```
int head = 0;//head永远指向“/”后的第一个字符  
int tail = 1;//tail永远指向head后的第一个“/”或字符串尾的后一个位置  
int num = 0;//代表当前在读路径的第几层  
int dot=tail;//记录“.”的位置
```

各变量意义如上图所示。

```
while(head!=length){ ...  
}
```

一个 while 循环来完成这件事。While 循环的退出条件为当 head 和 length 相等时（length 为路径字符串的长度）。

```
while(head!=length){  
    if(*(pathInput+head)=='/') head++;  
    else{ ...  
    }  
}
```

每一次循环开始时，先判断 head 指向的是否为“/”，若是，把 head++，使 head 指向“/”的后一位。

```
while(head!=length){  
    if(*(pathInput+head)=='/') head++;  
    else{  
        if(*(pathInput+tail)!='/' && tail!=length) tail++;  
        else{ ...  
        }  
    }  
}
```

当 head 满足指向“/”的后一位时，判断一下 tail 是不是指向“/”或者字符串尾的后一位。如果不是，就使 tail++。

```
while(head!=length){  
    if(*(pathInput+head)=='/') head++;  
    else{  
        if(*(pathInput+tail)!='/' && tail!=length) tail++;  
        else{  
            int len = tail-head;  
            int j;  
            if(num==pathDepth-1){ ...  
            }  
            else{ ...  
            }  
            head=tail;  
            tail++;  
        }  
    }  
}
```

当 head 和 tail 都满足我所规定的条件时，可以开始读取 head 和 tail 之间的数据。分为两种情况：head 和 tail 夹住的是最后一层和不是最后一

层。（因为如果是最后一层，可能要判断“.”的位置）。另外每次在将 head 和 tail 之间的内容读完之后，都令 head 等于 tail，并令 tail++，开始新一轮循环。

```
if(num==pathDepth-1){
    //遍历head和tail之间的字符，找到.的位置，由dot标记，如果没有“.”，dot就等于tail
    for(dot=head;dot<=tail;dot++){ if(*(pathInput+dot)=='.') break;
    *(paths+num)=malloc(11 * sizeof(char));
    if(dot<=tail){//有“.”
        int namelen=dot-head;//文件名长度
        int extendlen=tail-dot-1;//扩展名长度
        if(extendlen>3) extendlen=3;
        for(j=0;j<namelen;j++) *((*paths+num)+j)=TURNUP(*(pathInput+head+j));//变大写
        if(namelen!=8) for(;j<8;j++) *((*paths+num)+j)= ' ';//不足的用空格填充
        for(j=0;j<extendlen;j++) *((*paths+num)+8+j)=TURNUP(*(pathInput+dot+1+j));//变大写
        if(extendlen!=3) for(;j<3;j++) *((*paths+num)+8+j)= ' ';//不足的用空格填充
    }
    else{//没有“.”
        int namelen=tail-head;
        for(j=0;j<11;j++) {
            if(j<namelen) *((*paths+num)+j)=TURNUP(*(pathInput+head+j));//变大写
            else *((*paths+num)+j)= ' ';//剩下的全部用空格填充
        }
    }
}
```

如果是最后一层 ↑

```
else{
    *(paths+num)=malloc(11 * sizeof(char));
    for(j=0;j<11;j++){
        if(j<len)*((*paths+num)+j)=TURNUP(*(pathInput+head+j));//变大写
        else *((*paths+num)+j)= ' ';//剩下的全部用空格填充
    }
    num++;
}
```

如果不是最后一层 ↑。写完了将深度+1。

注：TURNUP() 是我自己写的函数，将小写变大写。

2. path_decode()

```
BYTE *path_decode(BYTE *path)
{
    BYTE *pathDecoded = (BYTE*)malloc(13 * sizeof(BYTE));
    if(path[0]=='.' && path[1]==' '){//判断“.”的情况
        *pathDecoded='.';
        *(pathDecoded+1]='\0';
    }
    else if(path[0]=='.' && path[1]==' '){//判断“..”的情况
        *pathDecoded='.';
        *(pathDecoded+1)= '.';
        *(pathDecoded+2]='\0';
    }
    else{
        int i=0;
        int num=0;
        while(*(path+i)!=' ' && i<8) *(pathDecoded+(i++))=TURNUP(*(path+(num++)));//转化文件名，变大写
        i=8;
        if(path[i]!=' '){//有扩展名
            *(pathDecoded+(num++))='.'; //先加“.”
            while(*(path+i)!=' ' && i<11)
                *(pathDecoded+(num++))=TURNUP(*(path+(i++)));//转化扩展名，变大写
            if(num!=13) *(pathDecoded+num]='\0';
        }
        else *(pathDecoded+(num++))='\0';//无扩展名，直接结束
    }
    return pathDecoded;
}
```

先判断“.”和“..”的情况，如果不是“.”和“..”，就先把前八位文件名转化成小写字母，直到遇到空格或满八位。然后转化扩展名。如果有扩展名，先加“.”，在把扩展名变小写；如果没有，就直接结束。

3. pre_init_fat16()

```
fread(&(fat16_ins->Bpb),sizeof(BPB_BS),1,fd);
fat16_ins->FirstRootDirSecNum=fat16_ins->Bpb.BPB_RsvdSecCnt+fat16_ins->Bpb.BPB_NumFATS*fat16_ins->Bpb.BPB_FATsZ16;
fat16_ins->FirstDataSector=fat16_ins->FirstRootDirSecNum + BYTES_PER_DIR*fat16_ins->Bpb.BPB_RootEntCnt/fat16_ins->Bpb.BPB_BytsPerSec;
return fat16_ins;
```

因为文件扇区对齐，所以直接把 Bpb 按照其大小读进来就行。

根目录第一个扇区号=剩余扇区数+FAT 数量*每个 FAT 扇区数。

数据区第一个扇区号=根目录第一个扇区号+每项字节数*根目录中项总数/每扇区字节数

4. fat_entry_by_cluster()

```
WORD fat_entry_by_cluster(FAT16 *fat16_ins, WORD ClusterN)
{
    BYTE sector_buffer[BYTES_PER_SECTOR];
    //secnum为ClusterN对应FAT内表项所在的扇区号
    int secnum=ClusterN/(BYTES_PER_SECTOR/2)+fat16_ins->Bpb.BPB_RsvdSecCnt;
    sector_read(fat16_ins->fd, secnum,sector_buffer);
    //两byte组成一个FAT表项，一个扇区可以装BYTES_PER_SECTOR/2个表项
    //addr表示ClusterN是当前扇区内第几个表项
    int addr=ClusterN%(BYTES_PER_SECTOR/2);
    addr = addr*2;//addr转化成buffer中的地址
    int returnvalue=sector_buffer[addr+1]*0x0100+sector_buffer[addr];
    return returnvalue;
}
```

先计算出 ClusterN 在 FAT 中对应表项所在的扇区号，将此扇区读出来。再计算 ClusterN 是此扇区内的第几个表项，将其读出来。

5. find_root()

```
for (i = 0; i < fat16_ins->Bpb.BPB_RootEntCnt; i++)
{
    //addr表示现在要读的是本扇区内第几个表项
    int addr=i%(BYTES_PER_SECTOR/BYTES_PER_DIR);//32byte组成一项，一个扇区中有16个项
    if(addr==0) //addr等于0说明要读一个新的扇区
        sector_read(fat16_ins->fd, fat16_ins->FirstRootDirSecNum+i/(BYTES_PER_SECTOR/BYTES_PER_DIR), buffer);
    *Dir=((DIR_ENTRY*)buffer)[addr];
    if(Dir->DIR_Name[0]==0)break;//停止
    if(!strcmp(paths[0],Dir->DIR_Name,11))//找到了根目录，调用find_subdir
        return find_subdir(fat16_ins, Dir, paths, pathDepth, 1);
}
return 1;
```

addr 表示当前扇区中第几个表项，如果 addr 为 0，表示要读新的扇区。

如果找到了根目录，调用 find_subdir。

6. find_subdir

```
int find_subdir(FAT16 *fat16_ins, DIR_ENTRY *Dir, char **paths, int pathDepth, int curDepth)
{
    if(curDepth==pathDepth) return 0;//停止递归的条件
    int i, j;
    BYTE buffer[BYTES_PER_SECTOR];

    WORD ClusterN;
    int sectornum;
    int Flag=0;
    BYTE* p;
    ClusterN=Dir->DIR_FstClusL0;//此子目录的第一个扇区号
    while(ClusterN>=0x0002 && ClusterN<=0xFFFF){...
    }
    return 1;
}
```

find_subdir() 是一个递归函数。递归的停止条件为：当前深度等于路径深度。

ClusterN 记录着当前读到了第几个簇，它最开始记录着第一个簇号。While 循环的条件是当前簇是已分配的簇。

```
while(ClusterN>=0x0002 && ClusterN<=0xFFEF){
    for(i=0;i<fat16_ins->Bpb.BPB_SecPerClus;i++){...
    }
    ClusterN=fat_entry_by_cluster(fat16_ins,ClusterN);//更新簇号
}
```

While 循环中嵌套着另一个循环 for，它遍历一个簇中的所有扇区。遍历之后，更新 ClusterN，将 ClusterN 更新为下一个簇号。

```
while(ClusterN>=0x0002 && ClusterN<=0xFFEF){
    for(i=0;i<fat16_ins->Bpb.BPB_SecPerClus;i++){
        //将当前扇区读出来
        sectornum=((ClusterN - 2) * fat16_ins->Bpb.BPB_SecPerClus) + fat16_ins->FirstDataSector+i;
        sector_read(fat16_ins->fd,sectornum,buffer);

        for(j=0;j<BYTES_PER_SECTOR/BYTES_PER_DIR;j++){//遍历此扇区中的每个表项
            p=&buffer[j*BYTES_PER_DIR];
            *Dir=*(DIR_ENTRY*)p;
            if(Dir->DIR_Name[0]==0x00){//若当前表项是末尾了，退出
                Flag=1;
                break;
            }
            else if(!strcmp(Dir->DIR_Name,paths[curDepth],11)){//如果找到该子目录，继续调用find_subdir
                return find_subdir(fat16_ins,Dir,paths,pathDepth,curDepth+1);
            }
        }
        if(Flag){
            Flag=0;
            break;
        }
    }
    ClusterN=fat_entry_by_cluster(fat16_ins,ClusterN);//更新簇号
}
```

for 中，先将此扇区读出来，然后遍历此扇区中的所有表项。若找到了此目录，继续调用 find_subdir。

7. fat16_readdir()—1

```
for (i = 0; i < fat16_ins->Bpb.BPB_RootEntCnt; i++){//遍历根目录中所有表项
{
    //将对应扇区读出来
    addr=i%(BYTES_PER_SECTOR/BYTES_PER_DIR);
    secnum=fat16_ins->FirstRootDirSecNum+i/(BYTES_PER_SECTOR/BYTES_PER_DIR);
    if(addr) sector_read(fat16_ins->fd, secnum, sector_buffer);

    for(j=0;j<BYTES_PER_SECTOR/BYTES_PER_DIR;j++){//遍历扇区内所有表项
        p=&sector_buffer[j*BYTES_PER_DIR];
        Root=*(DIR_ENTRY*)p;
        if(Root.DIR_Name[0]==0x00){//末尾
            Flag=1;
            break;
        }
        else{
            const char *filename = (const char *)path_decode(Root.DIR_Name);
            //若此目录/文件有效且未被删除
            if((Root.DIR_Attr==0x20 || Root.DIR_Attr==0x10) && Root.DIR_Name[0]!=0xE5)
                filler(buffer, filename, NULL, 0);
        }
    }
    if(Flag){
        Flag=0;
        break;
    }
}
```

先将此扇区读出来，在遍历扇区内所有表项。若此表项有效且未被删除，再将其填入 buffer。遇到末尾就跳出循环。

8. fat16_readdir()—2

```

while((ClusterN==0x0002 && ClusterN<0xFFEF)){//遍历此目录的所有簇
for(i=0;i<fat16_ins->Bpb.BPB_SecPerClus;i++){//遍历本簇中所有扇区
//读扇区
int secnum=fat16_ins->firstDataSector+(ClusterN-2)*fat16_ins->Bpb.BPB_SecPerClus+i;
sector_read(fat16_ins->fd,secnum,sector_buffer);

for(j=0;j<BYTES_PER_SECTOR/BYTES_PER_DIR;j++){//遍历本扇区中所有表项
p=&sector_buffer[j*BYTES_PER_DIR];
Dir=(DIR_ENTRY*)p;
if(Dir->DIR_Name[0]==0x00){
Flag=1;
break;
}
else{
const char *filename = (const char *)path_decode(Dir->DIR_Name);
if((Dir->DIR_Attr==0x20 || Dir->DIR_Attr==0x10) && Dir->DIR_Name[0]!=0xE5) filler(buffer, filename, NULL, 0);
}
}
if(Flag){
Flag=0;
break;
}
}
ClusterN=fat_entry_by_cluster(fat16_ins,ClusterN);//将ClusterN更新至下一个簇号
}
}

```

先遍历本目录的所有簇，在每个簇中遍历所有扇区，每个扇区中遍历所有表项。若此表项有效且未被删除，则填入 buffer，若遇到末尾，退出循环。

9. fat16_read

```

int fat16_read(const char *path, char *buffer, size_t size, off_t offset,
               struct fuse_file_info *fi)
{
    FAT16 *fat16_ins;
    struct fuse_context *context;
    context = fuse_get_context();
    fat16_ins = (FAT16 *)context->private_data;

    int i,j;
    BYTE sector_buffer[BYTES_PER_SECTOR];
    DIR_ENTRY* Dir;
    Dir = (DIR_ENTRY*)malloc(sizeof(DIR_ENTRY));
    find_root(fat16_ins,Dir,path);//Dir里为该文件的表项
    if(Dir->DIR_FileSize<offset) return 0;//offset大于文件大小，直接返回
}

```

Dir 中存放所读文件的表项，若 offset 大于文件大小，返回。

```

WORD ClusterN;//簇号
WORD FirstSectorNum;//本簇中第一个扇区号
WORD SectorNumInClus;//簇内读到了第几个扇区，取值0,1, ..., fat16_ins->Bpb.BPB_SecPerClus
WORD ReadSectorNum;//要读的扇区号

off_t ReadTail=offset;//记录读取末尾，取值在offset和offset+size之间
BYTE* ReadP;//ReadP永远指向sector_buffer第首地址，读取的数据地址以ReadP为基地址，加偏移量的方式寻找
WORD secnum;
WORD ClusterNum;
ReadP=&sector_buffer[0];

```

定义一些变量，方便使用。

```

for(i=0;i<size;i++,ReadTail++){
    if(ReadTail==offset){//第一次...
    }
    else if(ReadTail%BYTES_PER_SECTOR==0){//换扇区...
    }
    *(buffer+i)=*(ReadP+ReadTail%BYTES_PER_SECTOR);
}

```

遍历 size 的大小。

每一次遍历，先判断是不是第一次进来，再判断需不需要换扇区。如果都不要，直接给 buffer 赋值，进入下一个循环。

```

if(ReadTail==offset){//第一次
//找到offset所在的那个簇
ClusterNum=offset/(fat16_ins->Bpb.BPB_SecPerClus*BYTES_PER_SECTOR);
if(ClusterNum==0) ClusterN=Dir->DIR_FstClus0;
else{
ClusterN=Dir->DIR_FstClus0;
for(j=0;j<ClusterNum;j++) ClusterN=fat_entry_by_cluster(fat16_ins,ClusterN);
}
//把offset扇区读进来
SectorNumInClus=(offset-ClusterNum*fat16_ins->Bpb.BPB_SecPerClus*BYTES_PER_SECTOR)/BYTES_PER_SECTOR;
FirstSectorNum=fat16_ins->FirstDataSector+(ClusterN-2)*fat16_ins->Bpb.BPB_SecPerClus;
ReadSectorNum=FirstSectorNum+SectorNumInClus;
sector_read(fat16_ins->fd,ReadSectorNum,sector_buffer);
}

```

如果是第一次进来，先找到 offset 所在的那个簇（offset 可能不在第一个簇），再把 offset 所在的扇区读进来。

```

else if(ReadTail%BYTES_PER_SECTOR==0){//换扇区
if(SectorNumInClus==fat16_ins->Bpb.BPB_SecPerClus-1){//换cluster
ClusterN=fat_entry_by_cluster(fat16 (int)65519 N);
if(!(ClusterN==0x0002 && ClusterN<=0xFFEF)) return (int)(ReadTail-offset-1);
WORD* NextCluster;
first_sector_by_cluster(fat16_ins,ClusterN,NextCluster,&ReadSectorNum,sector_buffer);
}
else{//本cluster内换sector
sector_read(fat16_ins->fd,++ReadSectorNum,sector_buffer);
}
}
}

```

若需要换扇区，分成两种情况来考虑。第一种是需要换簇，那么找到下一个簇号，若下一个簇号是未分配的，则返回 ReadTail-offset，即所读字符长度，否则一定需要将下一个簇的首扇区读进来，则调用 first_sector_by_cluster；若不需要换簇，则直接将下一个扇区读进来。

二、运行结果截图

```

^Cfr@ubuntu:~/fr/lab4-code$ ./simple_fat16 --test
running test
-----
#1 running test_path_split
test case 1: OK
test case 2: OK
test case 3: OK
success in test_path_split

#2 running test_path_decode
test case 1: OK
test case 2: OK
test case 3: OK
success in test_path_decode

#3 running test_pre_init_fat16
success in test_pre_init_fat16

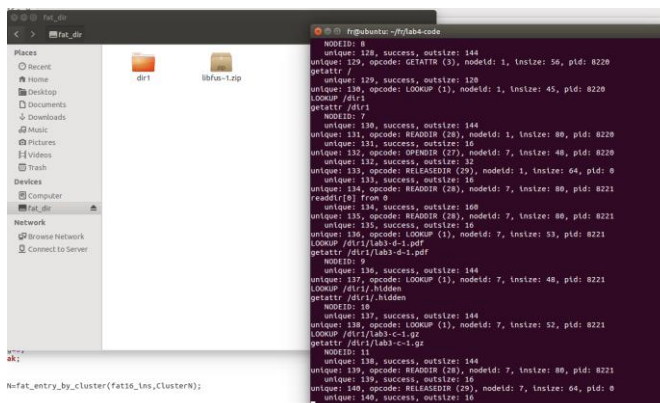
#4 running test_fat_entry_by_cluster
test case 1: OK
test case 2: OK
test case 3: OK
success in test_fat_entry_by_cluster

#5 running test_find_root
test case 1: OK
test case 2: OK
test case 3: OK
success in test_find_root

#6 running test_find_subdir
test case 1: OK
test case 2: OK
test case 3: OK
success in test_find_subdir

fr@ubuntu:~/fr/lab4-code$

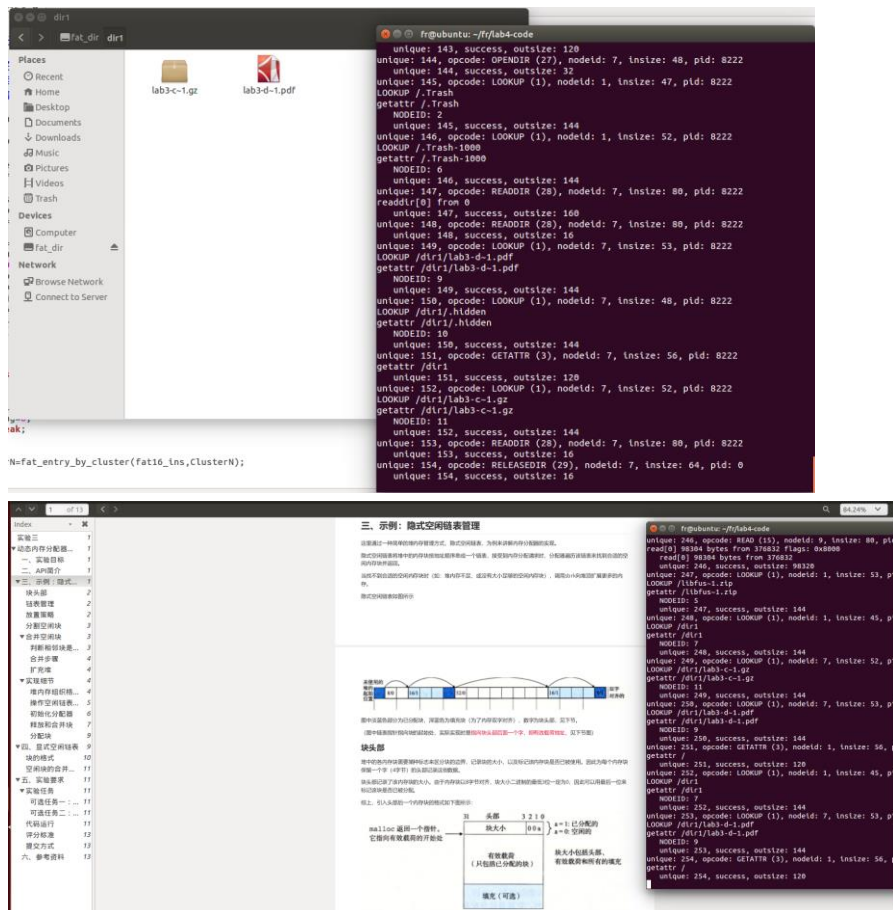
```



```

fr@ubuntu:~/fr/lab4-code$

```



三、技术问题

1. path_split 函数卡了我好久。主要是“.”没判断对，没有考虑缺失文件名和缺失扩展名的情况。本来觉得用一个循环来判断“.”的位置好麻烦，但是后来感觉不这样不行。
2. 本来没想到用 fread 可以把 Dir 或者 Bpb 一起读出来，还想着一项一项把它们读出来，后来想到它们作为一个结构体，是被连续存放的，所以直接读就 ok。
3. 最大最大、卡了我一天的 bug，就是，我定义了指针但是，没！有！malloc！这 bug 看了我一天，一直 segmentation fault，找到后如梦初醒，感觉这一天被浪费了。

四、实验总结

在这一次实验中，我实现了 FAT16 文件系统的读文件功能。从头到尾我没有遇到太大的困难，因为我做了充足的实验准备工作。我在开始写代码前，把 FAT16 文件系统的结构重新学习了一遍，把每一个区域的定义、组成精细到了每个字节地画了示意图出来，并在图上详细的标注了每一个变量的意义。所以写起 fat16_read 和 fat16_readdir 这两个函数的时候思路清晰，几乎改了两三遍就过了的程度。我深刻地体会到了在写代码之前先清楚怎么写是有多么关键。我看到身边一些同学在写代码时遇到 bug，不知道为什么会出，不知道怎么 debug，甚至写完代码之后对 fat 还不是很清楚。我认为这都是先前工作没有做好的表现。总之这次实验加深了我对文件系统的理解，写完后有一种“对上暗号”了的感觉。