操作系统实验 4

题目：FAT 文件系统的实现

廖洲洲，PB17081504

一、实验目标

- 熟悉 FAT16 的存储结构，利用 FUSE 实现一个只读的 FAT 文件系统

二、实验步骤

1. 将输入路径按“/”分割成多个字符串，并按照 FAT 文件名格式转换字符串

a) 根据文件中的“/”的数量求出 pathDepth

```
for(i=0;i<len;i++){
    if(pathInput[i]=='/')
        (*pathDepth_ret)=(*pathDepth_ret)+1;
    }
```

b) 将输入的文件名按“/”分割存储在 paths 中，同时将文件名转换为大写字符

```
for(i=0;i<(*pathDepth_ret);i++){
    for(j=0;j<32;j++){
    if(r<len&&pathInput[r]!='/'){
            if(pathInput[r] != '.')
                paths[i][j]=toupper(pathInput[r]);
            else
                paths[i][j]=pathInput[r];
            r++;
            }
    else
        paths[i][j]=' ';
        }
        r++;
    }
```

c) 将 paths 中的字符串分为文件名和文件类型分别保存在两个数组中，然后按前 8 位文件名后 3 位文件类型将文件名和文件类型重新存入 paths

```
    for(i=0;i<(*pathDepth_ret);i++){
        for(j=0;j<32&&paths[i][j]!='.';j++){
            if(j<8){
```

```
                name[j]=paths[i][j];
                }
            }
        for(j++,r=0;j<32&&r<3;j++,r++){
                type[r]=paths[i][j];
        }
        for(j=0;j<11;j++){
            if(j<8)
                paths[i][j]=name[j];
            else
                paths[i][j]=type[j-8];
            }
        paths[i][11]='\0';
        char type[]="   ";
        char name[]="        ";

    }
```

2. 将 FAT 文件名格式解码成原始的文件名

a) 对于“.”和“..”的文件名特殊处理，直接转换为“.”和“..”

```
    if(path[0]=='.'){
        pathDecoded[0]='.';
        if(path[1] == '.'){
            pathDecoded[1] = '.';
            pathDecoded[2] = '\0';
                }
        else{
            pathDecoded[1] = '\0';
            }
//      printf("@@##!!%s\n",pathDecoded);
        return pathDecoded;
    } //处理.. and .
```

b) 对输入的文件名进行解码，根据前 8 位为文件名，后 3 位为文件类型，对于前 8 位的字符，如果不是空格，则将它们转换位大写字符，然后再其后加“.”，然后将后 3 位不是空格的字符转换为小写字符，然后保存在其后。

```
    for(i=0,j=0;i<8;i++){
```

```
            if(path[i] !=' '){
                pathDecoded[j]=tolower(path[i]);//前 8 位
                j++;
                }
            }
        pathDecoded[j]='.';
        for(j++;i<11;i++){ //后 3 位
            if(path[i] != ' '){
                pathDecoded[j]=tolower(path[i]);
                j++;
                }
            }
    pathDecoded[j] = '\0';
```

3. 初始化 fat16_ins 的其余成员变量
   a) 先将 BPB 扇区读出,BPB 为第一个扇区，直接读出即可

      ```
      sector_read(fd,0,&(fat16_ins->Bpb));
      ```

   b) 根目录扇区号=保留扇区数+FAT 扇区数*FAT 数量

      ```
      fat16_ins->FirstRootDirSecNum = fat16_ins->Bpb.BPB_RsvdSecCnt+fat16_ins->Bpb.BPB_NumFATS * fat16_ins->Bpb.BPB_FATSz16;
      ```

   c) 数据扇区号=根目录扇区号+32*根目录项数/每扇区字节数

      ```
      fat16_ins->FirstDataSector=fat16_ins->FirstRootDirSecNum + 32 * fat16_ins->Bpb.BPB_RootEntCnt/fat16_ins->Bpb.BPB_BytsPerSec;
      ```

4. 返回簇号为 ClusterN 对应的 FAT 表项
   a) 缓存一个扇区的内容，这里可以直接缓存 FAT1, 因为一个族号对应的表项为 2 字节，使用 WORD 更为方便进行读取

      ```
      WORD sector_buffer[BYTES_PER_SECTOR/2];
      ```

   b) 每个 FAT 占了 60 个扇区，相对应的扇区=保留扇区+族号/256 因为一扇区有 512 字节，一个族号对应 2 个字节的表项

      ```
      sector_read(fat16_ins->fd,fat16_ins->Bpb.BPB_RsvdSecCnt+
      ClusterN/256,sector_buffer);
      ```

   c) 余数为对应表项在缓存区的位置

      ```
      return sector_buffer[ClusterN%256];
      ```

5. 从 root directory 开始，查找 path 对应的文件或目录，找到返回 0，没找

到返回 1，并将 Dir 填充为查找到的对应目录项

a) 首先查找名字为 paths[0]的目录项，即在根目录中的文件或文件夹

b) 计算出根目录所占的扇区数，由于一个扇区 512 字节，一个目录项 32 字节，即能保存 16 个目录项，则对每个扇区的 16 个目录项依次查找

c) 根据 pathDepth 判断是否调用 find_subdir 继续查找

```
int                         RootDirSecCnt=32                         *
fat16_ins->Bpb.BPB_RootEntCnt/fat16_ins->Bpb.BPB_BytsPerSec;
    for(i=0;i<RootDirSecCnt;i++){
         sector_read(fat16_ins->fd,
fat16_ins->FirstRootDirSecNum+i, buffer);
            for(j=0;j<16;j++){
                if(strncmp(paths[0],                (DIR_ENTRY*)
&buffer[32*j],11) == 0){
                     *Dir = *( (DIR_ENTRY*) &buffer[32*j]);
                   if(pathDepth==1){
                     return 0;
                        }
                   else
                   {
    //                    printf("\n###start find_subdir \n");
                     return   find_subdir(fat16_ins,   Dir,
paths, pathDepth, 1);
                   }
                }
            }
    }
```

6. 从子目录开始查找 path 对应的文件或目录，找到返回 0，没找到返回 1，并将 Dir 填充为查找到的对应目录项

a) 在 find_subdir 入口处，Dir 应该是要查找的这一级目录的表项，需要根据其中的簇号，读取这级目录对应的扇区数据

```
WORD ClusterN, FatClusEntryVal, FirstSectorofCluster;
ClusterN = Dir->DIR_FstClusLO;//得到文件夹或文件的首簇号
```

b) 根据簇号 ClusterN，获取其对应的第一个扇区的扇区号和数据，以及对应的 FAT 表项，根据簇找到对应的文件信息，按簇->扇区->32 字节文件信息的分割顺序进行查找，即对 32 字节的文件信息依次查找，如

果一个扇区查找完毕，则查找下一扇区，如果一个簇查找完毕，再进行跨簇

```
                do{ //这里要传入的是指针
      first_sector_by_cluster(fat16_ins,          ClusterN,&FatClusEntryVal,
&FirstSectorofCluster, buffer);
      for(i=0;i<fat16_ins->Bpb.BPB_SecPerClus;i++){
            sector_read(fat16_ins->fd, FirstSectorofCluster+i, buffer);
              for(j=0;j<16;j++){
                  if(strncmp(paths[curDepth],                  (DIR_ENTRY*)
&buffer[32*j],11) == 0){
                     *Dir = *( (DIR_ENTRY*) &buffer[32*j]);
                    if(pathDepth==curDepth+1){
                        return 0;
                            }
                    else {
                         return   find_subdir(fat16_ins,   Dir,   paths,
pathDepth, curDepth+1);
                    }
                 }
              }
           }
    ClusterN=FatClusEntryVal;
}while(FatClusEntryVal <0xFFF8);
```

7. 将 root directory 下的文件或目录通过 filler 填充到 buffer 中，不需要遍历子目录

   a) 将根目录的多个扇区进行依次查找，对于每个扇区则按每 32 字节进行查找，如果该 32 字节文件信息表示的是一个有效的文件或则目录，则将其文件名或目录名通过 filler 填充到 buffer 中

```
      int                    RootDirSecCnt=32                    *
fat16_ins->Bpb.BPB_RootEntCnt/fat16_ins->Bpb.BPB_BytsPerSec;
       for(i=0;i<RootDirSecCnt;i++){
             sector_read(fat16_ins->fd,
fat16_ins->FirstRootDirSecNum+i,Root_buffer);
                for(j=0;j<16;j++){
                        Root      =      *(      (DIR_ENTRY*)
&Root_buffer[32*j]);
                     if(Root.DIR_Name[0]     !=     0xE5     &&
```

```
(Root.DIR_Attr == 0x10 || Root.DIR_Attr == 0x20)){//目录项不为空也未被删除
                        const char *filename = (const char *)path_decode(Root.DIR_Name);
                        filler(buffer, filename, NULL, 0);
                        }
                    }
                }
```

8. 通过 find_root 获取 path 对应的目录的目录项，然后访问该目录，将其下的文件或目录通过 filler 填充到 buffer 中，不需要遍历子目录

   a) 得到对应目录的目录项，获取首簇号

```
find_root(fat16_ins, &Dir, path);
WORD ClusterN, FatClusEntryVal, FirstSectorofCluster;
ClusterN = Dir.DIR_FstClusLO;
```

   b) 根据簇号 ClusterN 获取其对应的第一个扇区的扇区号和数据，以及对应的 FAT 表项，根据族找到对应的文件信息，按簇->扇区->32 字节文件信息遍历，即将一个簇中的所有扇区遍历后，若不止一个簇，则进行跨簇。如果对应的文件或目录有效,则将其名字用 filler 填充到 buffer 中

```
    do{ //这里要传入的是指针
                first_sector_by_cluster(fat16_ins, ClusterN,&FatClusEntryVal, &FirstSectorofCluster, sector_buffer);
                for(i=0;i<fat16_ins->Bpb.BPB_SecPerClus;i++){
                sector_read(fat16_ins->fd, FirstSectorofCluster+i, sector_buffer);
                for(j=0;j<16;j++){

                        Dir = *(DIR_ENTRY*)&sector_buffer[32*j]);
                    if(Dir.DIR_Name[0] != 0xE5 && (Dir.DIR_Attr == 0x10 || Dir.DIR_Attr == 0x20)){ //不为空，未删除
                        const char *filename = (const char *)path_decode(Dir.DIR_Name);
                        filler(buffer, filename, NULL, 0);
                        }
                        }
```

```
}

    ClusterN=FatClusEntryVal;
}while(0x0002<= FatClusEntryVal <=0xFFEF);
```

9. 从 path 对应的文件的 offset 字节处开始读取 size 字节的数据到 buffer 中，并返回实际读取的字节数

   a) 当 offset 超过文件大小时，应该返回 0

   ```
   if(offset>= Dir.DIR_FileSize)
       return 0;
   ```

   b) 得到真正要读取的数据大小

   ```
   realsize = (offset + size > Dir.DIR_FileSize) ? Dir.DIR_FileSize - offset : size;
   ```

   c) 根据偏移量找到要读的数据在文件的第几簇，然后得到相应的簇号，再根据要读的数据大小考虑是否要跨簇进行读取

   ```
   offset -= fat16_ins->Bpb.BPB_BytsPerSec * fat16_ins->Bpb.BPB_SecPerClus * clusters;//在要读的族中的地址
       while (clusters > 0)
       {//找到真正读的开始族
         fseek(fat16_ins->fd, FatClusEntryVal, SEEK_SET);
         fread(&clusterN, 2, 1, fat16_ins->fd);
         clusters--;
         FatClusEntryVal = fat16_ins->Bpb.BPB_RsvdSecCnt * fat16_ins->Bpb.BPB_BytsPerSec + clusterN * 2;
       }
       FirstSectorofCluster = fat16_ins->FirstDataSector + (clusterN - 2) * fat16_ins->Bpb.BPB_SecPerClus;
       sectors = offset / fat16_ins->Bpb.BPB_BytsPerSec; //第几个扇区开始读
       sector_read(fat16_ins->fd, FirstSectorofCluster + sectors, sector_buffer);
       for (i = 0; i < realsize; i++)
       {
         buffer[i] = sector_buffer[offset - sectors * BYTES_PER_SECTOR];
         offset++; //一个个字节读进去
   ```

```
            if (offset % BYTES_PER_SECTOR == 0)//如果一个扇区读完
了
            {
              if      (offset      ==      BYTES_PER_SECTOR      *
fat16_ins->Bpb.BPB_SecPerClus)//一个族读完了
              { //族也读完了，需要换下一个族
                offset = 0;
                sectors = 0;
                fseek(fat16_ins->fd, FatClusEntryVal, SEEK_SET);
                fread(&clusterN, 2, 1, fat16_ins->fd);//得到下一族
                FatClusEntryVal = fat16_ins->Bpb.BPB_RsvdSecCnt *
fat16_ins->Bpb.BPB_BytsPerSec + clusterN * 2;
                FirstSectorofCluster = fat16_ins->FirstDataSector
+ (clusterN - 2) * fat16_ins->Bpb.BPB_SecPerClus;
                sector_read(fat16_ins->fd, FirstSectorofCluster +
sectors, sector_buffer);
              }
              else
              {//只是扇区读完了
                sectors++;
                sector_read(fat16_ins->fd, FirstSectorofCluster +
sectors, sector_buffer);
              }
            }
          }

    return realsize;
```

三、实验结果截图

- 第一次测试通过

```
--------------
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
```

- 文件系统挂载

dir1

libfuse-fuse_2_9_5.zip

四、实验注意事项

- 实验中函数调用大都传递的是指针参数，如果传递的不是指针会出现错误
- 对于文件是否有效，如果直接这样判断

  `if(Dir.DIR_Name[0] != 0xE5 && Dir.DIR_Name[0] != 0x00)`当挂载文件系统时会出现输入输出错误，需要这样判断

  `if(Dir.DIR_Name[0] != 0xE5 && (Dir.DIR_Attr == 0x10 || Dir.DIR_Attr == 0x20))`

五、实验总结

- 通过这次实验，我对 FAT16 的存储结构有了更加深入的了解，同时也初步认识了 FUSE 的功能和实现原理。
- 这次实验让我对字符串的处理操作也更加熟悉了。