

# 《操作系统原理与设计》 实验报告

---



实验题目: FAT文件系统的实现

学生姓名: 王志强

学生学号: PB18051049

完成日期: 2020.06.18

计算机实验教学中心制

2019年9月

# 一、实验目的

- 熟悉FAT16的存储结构，利用FUSE实现一个FAT文件系统

# 二、实验环境

- OS: Ubuntu 18.04LTS

# 三、实验要求

## 1、任务一：利用FUSE实现一个功能完善的FAT 16文件系统

只实现了支持读操作的FAT16文件系统

- 下载代码

```
1 $ wget https://raw.githubusercontent.com/ZacharyLiu-
  CS/USTC_OS/master/Lab4-
  File-System/lab4-code.tar.gz
2 File-System/lab4-code.tar.gz
3 $ tar xzf lab4-code.tar.gz
```

- 补全代码包中的simple\_fat16.c中的TODO标记（一共5处）的部分，实现一个只读的FAT16文件系统
- 运行与测试

使用如下的命令编译并测试程序：

```
1 #进入源码目录
2 make clean
3 make
4 #测试一
5 ./simple_fat16 --test
```

测试一是为了验证程序的FAT相关部分的代码正确性，通过测试后，运行如下的命令进行FUSE功能的测试：

```
1 #测试二
2 ./simple_fat16 -d fat_dir
```

- 回答相关问题

# 四、实验内容

## 1. 代码补充及描述

代码解释见注释

### 1.1 path\_split函数

```
1 /** TODO:
2  * 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
```

```

3  * 输入: pathInput: char*, 输入的文件路径名, 如/home/user/m.c
4  * 输出: pathDepth_ret, 文件层次深度, 如 3
5  * 返回: 按FAT格式转换后的文件名字符串.
6  *
7  * Hint1:假设pathInput为“/dir1/dir2/file.txt”, 则将其分割
   成“dir1”, “dir2”, “file.txt”,
8  *     每个字符串转换成长度为11的FAT格式的文件名, 如“file.txt”转换成“FILE
   TXT”,
9  *     返回转换后的字符串数组, 并将*pathDepth_ret设置为3, 转换格式在文档中有说明.
10 * Hint2:可能会出现过长的字符串输入, 如“/.Trash-1000”, 需要自行截断字符串
11 * Hint3:需要考虑.和..的情况(. 和 .. 的输入应当为 /.和/..)
12 **/
13 char **path_split(char *pathInput, int *pathDepth_ret)
14 {
15     int i,j,k ;
16     int pathDepth = 0;
17     i=j=k=0;
18     //计算文件层次深度
19     while(pathInput[i] != '\0'){
20         if(pathInput[i++] == '/')
21             pathDepth++;
22     }
23
24     char **paths = malloc(pathDepth * sizeof(char *));
25     //字符串分离
26     char split_flag[2] = "/";
27     char *temp; //记录每次分离出的字符串
28     temp = strtok(pathInput,split_flag);
29     for(i = 0;i < pathDepth;i++){
30         paths[i] = temp; //指针指向分离的字符串
31         temp = strtok(NULL,split_flag);
32     }
33     /*标准格式化后的字符串指针*/
34     char ** path_transform = (char ** ) malloc(pathDepth * sizeof(char
   *));
35     /*为新串分配空间*/
36     for (i = 0; i < pathDepth; i++)
37         path_transform[i] = (char *) malloc(11 * sizeof(char));
38
39     /*字符串转换为FAT16标准格式*/
40
41     int flag = 0; //出现了'.'则为1
42     for (i = 0; i < pathDepth; i++) {
43         int name_len = 0; //文件名长度
44         int ext_len = 0; //拓展名长度
45         for (j = 0, k = 0;; j++, k++) {
46             /*当遇见'.'字符*/
47             if (paths[i][j] == '.') {
48                 /* 判断是否是 "." 文件 */
49                 if (j == 0 && paths[i][j + 1] == '\0') {
50                     path_transform[i][0] = '.';
51                     for (k = 1; k < 11; k++)
52                         path_transform[i][k] = ' ';
53                     break;
54                 }
55
56                 /* 判断是否是 ".."文件 */

```

```

57         if (j == 0 && paths[i][j + 1] == '.' && paths[i][j + 2] ==
'\0') {
58             path_transform[i][0] = '.';
59             path_transform[i][1] = '.';
60             for (k = 2; k < 11; k++)
61                 path_transform[i][k] = ' ';
62             break;
63         }
64
65
66         /*当出现了'.'*/
67         if (!flag) {
68             if (paths[i][j + 1] == '\0') {
69                 printf("%s:This file has no extension name\n",
paths[i]);
70                 exit(1);
71             }
72
73             flag = 1;
74
75             /*补上空白字符*/
76             for (; k < 8; k++)
77                 path_transform[i][k] = ' ';
78             k = 7;
79
80             /* 文件名不规范 */
81         }
82         else {
83             printf("%s: More than one dot character (.) \n",
paths[i]);
84             exit(1);
85         }
86     }
87
88     /*文件名后补空格*/
89     else if (paths[i][j] == '\0') {
90         for (; k < 11; k++)
91             path_transform[i][k] = ' ';
92         break;
93     }
94
95     /* 小写转大写 */
96     else if (paths[i][j] >= 'a' && paths[i][j] <= 'z') {
97         if (name_len==8 && !flag)
98             continue; /*截断*/
99         if(ext_len==3)
100             break;
101         path_transform[i][k] = paths[i][j] - 32;
102         if (flag)
103             ext_len++;
104         else
105             name_len++;
106     }
107     /*其它字符*/
108     else {
109         if (name_len==8 && !flag)
110             continue; /*截断*/
111         if(ext_len==3)

```

```

112         break;
113         path_transform[i][k] = paths[i][j];
114         if (flag)
115             ext_len++;
116         else
117             name_len++;
118     }
119 }
120 }
121
122 *pathDepth_ret = pathDepth;
123 free(paths);
124 return path_transform;
125 }

```

## 1.2 pre\_init\_fat16函数

```

1  FAT16 *pre_init_fat16(void)
2  {
3      /* Opening the FAT16 image file */
4      FILE *fd;
5      FAT16 *fat16_ins;
6
7      fd = fopen(FAT_FILE_NAME, "rb");
8
9      if (fd == NULL)
10     {
11         fprintf(stderr, "Missing FAT16 image file!\n");
12         exit(EXIT_FAILURE);
13     }
14
15     fat16_ins = malloc(sizeof(FAT16));
16     fat16_ins->fd = fd;
17
18     /** TODO
19      * 初始化fat16_ins的其余成员变量，该struct定义在fat16.c的第65行
20      * 其余成员变量如下：
21      *   FirstRootDirSecNum->第一个根目录的扇区偏移。
22      *   FirstDataSector->第一个数据区的扇区偏移
23      *   Bpb->Bpb结构
24      * Hint1: 使用sector_read读出fat16_ins中Bpb项的内容，这个函数定义在本文件的第18
25 行。
26      * Hint2: 可以使用BPB中的参数完成其他变量的初始化，该struct定义在fat16.c的第23行
27      * Hint3: root directory的大小与Bpb.BPB_RootEntCnt有关，并且是扇区对齐的
28      */
29     /*读取Bpb,在DBR,0号扇区中*/
30     sector_read(fat16_ins->fd, 0, &fat16_ins->Bpb);
31     /*根目录扇区偏移，在保留扇区和FAT表后*/
32     fat16_ins->FirstRootDirSecNum = fat16_ins->Bpb.BPB_RsvdSecCnt
33     + (fat16_ins->Bpb.BPB_FATSz16 * fat16_ins->Bpb.BPB_NumFATS);
34     /*计算根目录所用扇区*/
35     DWORD RootDirSectors = ((fat16_ins->Bpb.BPB_RootEntCnt * 32) +
36     (fat16_ins->Bpb.BPB_BytsPerSec - 1)) / fat16_ins->Bpb.BPB_BytsPerSec;
37     /*保留扇区，fat表，根目录后即是数据区*/
38     fat16_ins->FirstDataSector = fat16_ins->Bpb.BPB_RsvdSecCnt
39     + (fat16_ins->Bpb.BPB_NumFATS *
40     fat16_ins->Bpb.BPB_FATSz16) + RootDirSectors;

```

```

40
41
42     return fat16_ins;
43 }

```

### 1.3 fat\_entry\_by\_cluster函数

```

1  /** TODO:
2   * 返回簇号为ClusterN对应的FAT表项
3   */
4  WORD fat_entry_by_cluster(FAT16 *fat16_ins, WORD ClusterN)
5  {
6      BYTE sector_buffer[BYTES_PER_SECTOR];
7      /*首先确定FAT表相对字节偏移*/
8      WORD FAT_offset = ClusterN * 2;
9
10     /*找到FAT中对应的扇区号 */
11     WORD FatSecNum = fat16_ins->Bpb.BPB_RsvdSecCnt
12         + (FAT_offset / fat16_ins->Bpb.BPB_BytsPerSec);
13     /*计算表项的大小*/
14     WORD FatEntSize = FAT_offset % fat16_ins->Bpb.BPB_BytsPerSec;
15
16     /* 读取FAT扇区中的内容, 获得相应表项 */
17     sector_read(fat16_ins->fd, FatSecNum, &sector_buffer);
18     /*返回FAT表项*/
19     return *((WORD *) &sector_buffer[FatEntSize]);
20 }

```

### 1.4 find\_subdir函数

```

1  /** TODO:
2   * 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
3   *
4   * Hint1: 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
5   * Hint2: 目录的大小是未知的, 可能跨越多个扇区或跨越多个簇; 当查找到某表项以0x00开头就可
   * 以停止查找
6   * Hint3: 需要查找名字为paths[curDepth]的文件或目录, 同样需要根据pathDepth判断是否继
   * 续调用find_subdir函数
7   */
8
9  int find_subdir(FAT16 *fat16_ins, DIR_ENTRY *Dir, char **paths, int
   pathDepth, int curDepth)
10 {
11     int i, j, SameName;
12     int DirSecCnt = 1; /* 用于统计已读取的扇区数 */
13     BYTE buffer[BYTES_PER_SECTOR];
14     /*簇号, 表项, 簇的第一个扇区*/
15     WORD ClusterN, FatClusEntryVal, FirstSectorofCluster;
16     /*获取簇号*/
17     ClusterN = Dir->DIR_FstClusLO;
18     /*获取表项*/
19     FatClusEntryVal = fat_entry_by_cluster(fat16_ins, ClusterN);
20     /*簇的第一个扇区, 注意簇从2开始编号! */
21     FirstSectorofCluster = ((ClusterN - 2) * fat16_ins->Bpb.BPB_SecPerClus)
+ fat16_ins->FirstDataSector;

```

```

22     /*读取第一个扇区的内容*/
23     sector_read(fat16_ins->fd, FirstSectorofCluster, buffer);
24
25     /*查找path对应的文件或目录*/
26     for (i = 1; Dir->DIR_Name[0] != 0x00; i++) {
27         memcpy(Dir, &buffer[((i - 1) * BYTES_PER_DIR) % BYTES_PER_SECTOR],
BYTES_PER_DIR);
28
29         /*比较文件名*/
30         SameName = 1;
31         for (j = 0; j < 11; j++) {
32             if (Dir->DIR_Name[j] != paths[curDepth][j]) {
33                 SameName = 0; /*匹配失败*/
34                 break;
35             }
36         }
37
38         /* 定位成功, 停止检索 */
39         if ((SameName && Dir->DIR_Attr == ATTR_ARCHIVE && curDepth + 1 ==
pathDepth) ||
40             (SameName && Dir->DIR_Attr == ATTR_DIRECTORY && curDepth + 1 ==
pathDepth)) {
41             return 0;
42         }
43
44         /* 检索未完成, 递归搜索 */
45         if (SameName && Dir->DIR_Attr == ATTR_DIRECTORY) {
46             return find_subdir(fat16_ins, Dir, paths, pathDepth, curDepth + 1);
47         }
48
49         /* 扇区读16次后到达尽头 */
50         if (i % 16 == 0) {
51             /* 读取下一个扇区*/
52             if (DirSecCnt < fat16_ins->Bpb.BPB_SecPerClus) {
53                 sector_read(fat16_ins->fd, FirstSectorofCluster + DirSecCnt,
buffer);
54                 DirSecCnt++;
55             } else {
56                 /* 文件在该簇读取完 */
57                 if (FatClusEntryVal == 0xffff) {
58                     return 1;
59                 }
60
61                 /*大文件: 准备下一轮循环, 读取下一个簇, 被记录在当前读出来的表项中*/
62                 /*这段后函数刚开始的功能一样*/
63                 ClusterN = FatClusEntryVal;
64                 FatClusEntryVal = fat_entry_by_cluster(fat16_ins, ClusterN);
65                 FirstSectorofCluster = ((ClusterN - 2) * fat16_ins->
Bpb.BPB_SecPerClus) + fat16_ins->FirstDataSector;
66                 sector_read(fat16_ins->fd, FirstSectorofCluster, buffer);
67                 i = 0;
68                 DirSecCnt = 1;
69             }
70         }
71     }
72
73     /* 未找到文件 */
74     return 1;

```

## 1.5 read\_path函数

```

1  /** TODO:
2   * 从path对应的文件(一定不为根目录"/")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
3   * Hint: 以扇区为单位读入,需要考虑读到当前簇结尾时切换到下一个簇,并更新要读的扇区号等信
   息
4   * Hint: 文件大小属性是Dir.DIR_FileSize; 当offset超过文件大小时,应该返回0
5   * 此函数会作为读文件的函数实现被fuse文件系统使用,见fat16_read函数
6   * 所以实现正确时的效果为实验文档中的测试二中可以正常访问文件
7   **/
8
9  int read_path(FAT16* fat16_ins, const char *path, size_t size,
10               off_t offset, char *buffer)
11  {
12      int i, j;
13      /* 文件对应目录项,簇号,簇对应FAT表项的内容,簇的第一个扇区号 */
14      DIR_ENTRY Dir;
15      WORD ClusterN, FatClusEntryVal, FirstSectorofCluster;
16      BYTE *sector_buffer = malloc((size + offset) * sizeof(BYTE));
17      /* 调用函数,从根目录开始查找文件 */
18      find_root(fat16_ins, &Dir, path);
19
20      /* 拿到目录项后,计算簇号、表项内容、簇的首个扇区号 */
21      ClusterN = Dir.DIR_FstClusLO;
22      FatClusEntryVal = fat_entry_by_cluster(fat16_ins, ClusterN);
23      FirstSectorofCluster = ((ClusterN - 2) * fat16_ins->Bpb.BPB_SecPerClus)
24          + fat16_ins->FirstDataSector;
25
26      /* 找到读取的目标位置,将内容读取到sector_buffer中 */
27      for (i = 0, j = 0; i < size + offset; i += BYTES_PER_SECTOR, j++) {
28          sector_read(fat16_ins->fd, FirstSectorofCluster + j, sector_buffer +
29 i);
30
31          /* 该簇结束,读取下一个簇 */
32          if ((j + 1) % fat16_ins->Bpb.BPB_SecPerClus == 0) {
33              /* 和上个函数的操作一样,进入到下个簇 */
34              ClusterN = FatClusEntryVal;
35              FatClusEntryVal = fat_entry_by_cluster(fat16_ins, ClusterN);
36              FirstSectorofCluster = ((ClusterN - 2) * fat16_ins-
37 >Bpb.BPB_SecPerClus)
38          + fat16_ins->FirstDataSector;
39              j = -1; /*下个循环开始j再次为0*/
40          }
41      }
42
43      /*读取文件内容*/
44      if (offset < Dir.DIR_FileSize) {
45          memcpy(buffer, sector_buffer + offset, size);
46      }
47      /* offset超过文件大小,返回0 */
48      else {
49          size = 0;
50      }
51  }

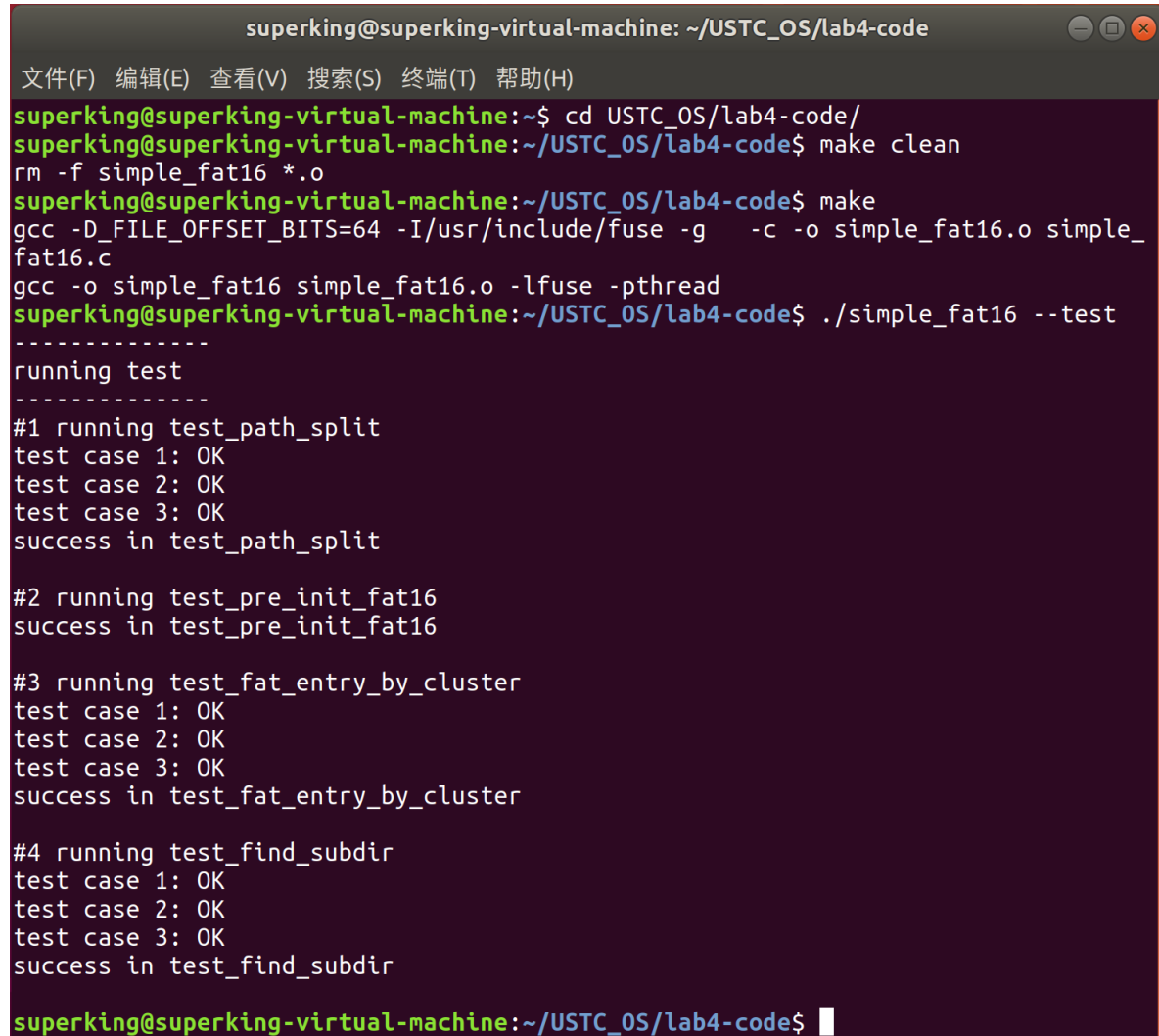
```



```
50
51     free(sector_buffer);
52     return size;
53 }
```

## 2.运行与测试

### 2.1 测试一结果



```
superking@superking-virtual-machine: ~/USTC_OS/lab4-code
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
superking@superking-virtual-machine:~$ cd USTC_OS/lab4-code/
superking@superking-virtual-machine:~/USTC_OS/lab4-code$ make clean
rm -f simple_fat16 *.o
superking@superking-virtual-machine:~/USTC_OS/lab4-code$ make
gcc -D_FILE_OFFSET_BITS=64 -I/usr/include/fuse -g -c -o simple_fat16.o simple_fat16.c
gcc -o simple_fat16 simple_fat16.o -lfuse -pthread
superking@superking-virtual-machine:~/USTC_OS/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_pre_init_fat16
success in test_pre_init_fat16

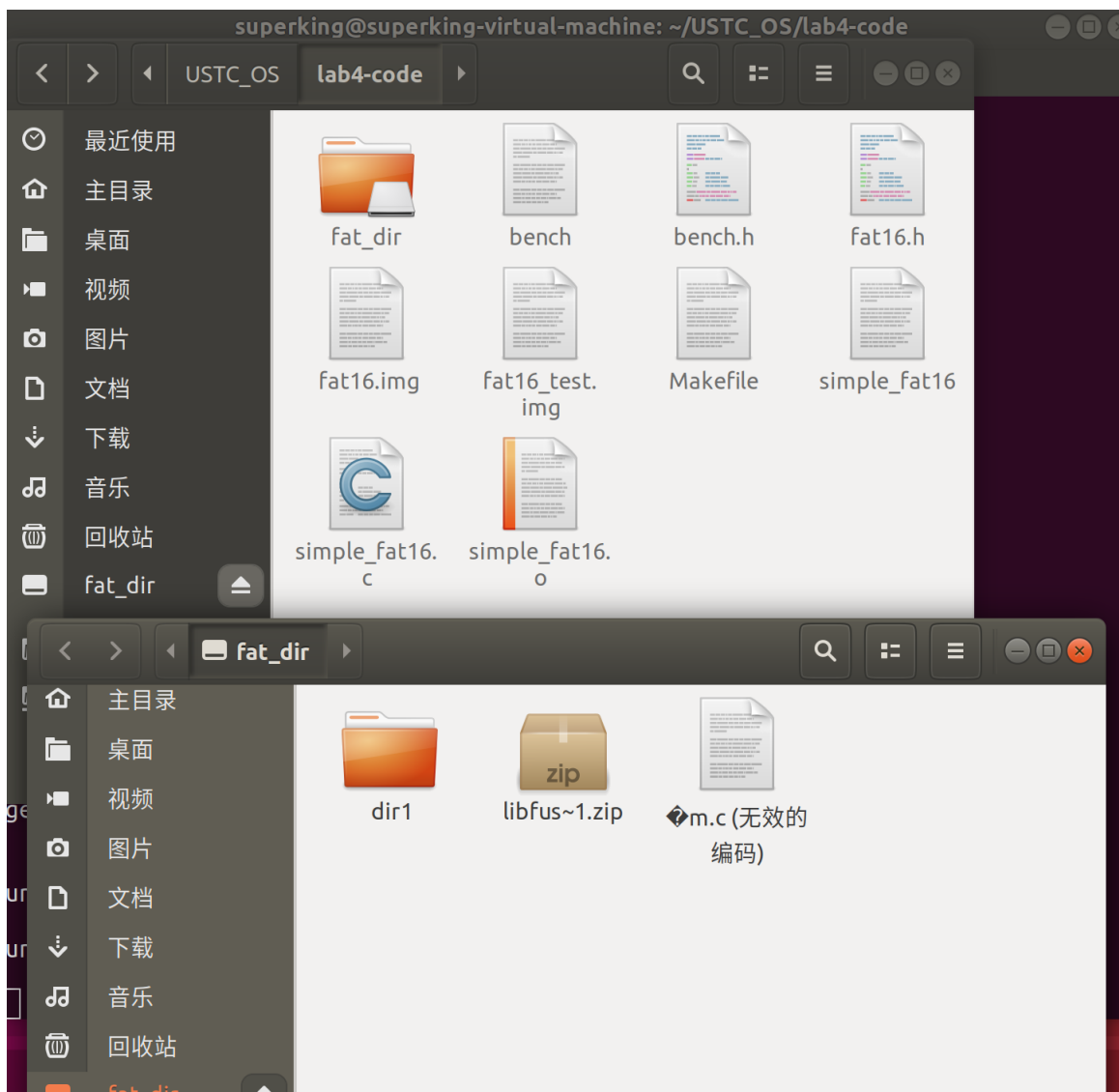
#3 running test_fat_entry_by_cluster
test case 1: OK
test case 2: OK
test case 3: OK
success in test_fat_entry_by_cluster

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

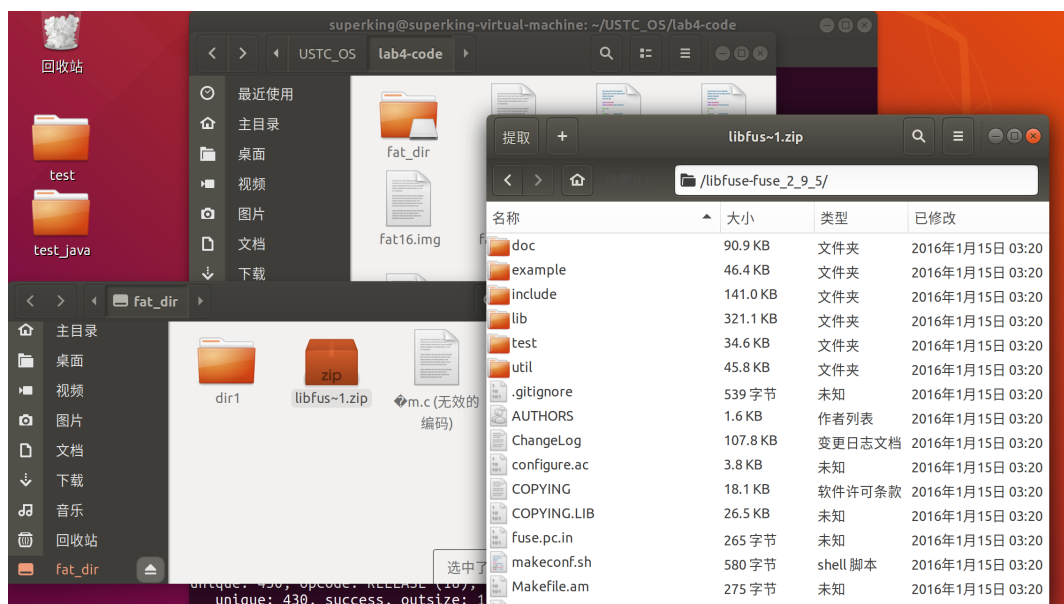
superking@superking-virtual-machine:~/USTC_OS/lab4-code$
```

### 2.2 测试二结果

- 打开fat\_dir 文件夹



- 查看libfus~1.zip 文件内容 (GUI+vim)
  - GUI查看文件内容



- 终端vim查看文件内容

```
superking@superking-virtual-machine: ~/USTC_OS/lab4-code/fat_dir
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
" zip.vim version v28
" Browsing zipfile /home/superking/USTC_OS/lab4-code/fat_dir/libfus~1.zip
" Select a file with cursor and press ENTER

libfuse-fuse_2_9_5/
libfuse-fuse_2_9_5/.gitignore
libfuse-fuse_2_9_5/AUTHORS
libfuse-fuse_2_9_5/COPYING
libfuse-fuse_2_9_5/COPYING.LIB
libfuse-fuse_2_9_5/ChangeLog
libfuse-fuse_2_9_5/Makefile.am
libfuse-fuse_2_9_5/NEWS
libfuse-fuse_2_9_5/README.NFS
libfuse-fuse_2_9_5/README.md
libfuse-fuse_2_9_5/configure.ac
libfuse-fuse_2_9_5/doc/
libfuse-fuse_2_9_5/doc/Doxyfile
libfuse-fuse_2_9_5/doc/Makefile.am
libfuse-fuse_2_9_5/doc/fusermount.1
libfuse-fuse_2_9_5/doc/how-fuse-works
libfuse-fuse_2_9_5/doc/kernel.txt
libfuse-fuse_2_9_5/doc/mount.fuse.8
libfuse-fuse_2_9_5/doc/unlockmgr_server.1
1,1 顶端
unique: 578, opcode: RELEASDIR (29), nodeid: 1, insize: 64, pid: 0
unique: 578, success, outsize: 16
```

## 2.3 录屏测试

```
superking@superking-virtual-machine:~/USTC_OS/lab4-code$ md5sum bench.h -c bench.h
md5sum: bench.h: 找不到格式适用的MD5 校验和
bench.h: 成功
```

## 3.回答问题

- 简要描述代码定义的结构体fat16\_oper中每个元素所对应的操作和操作执行的流程

```
1 struct fuse_operations fat16_oper = {
2     .init = fat16_init,
3     .destroy = fat16_destroy,
4     .getattr = fat16_getattr,
5     .readdir = fat16_readdir,
6     .read = fat16_read
7 };
```

元素	功能	执行
fat16_init	初始化 FAT16 文件	首先打开FAT16文件，然后初始化 fat16_ins 的成员变量
fat16_destroy	销毁 FAT16 文件	直接通过free( )函数实现，释放内存
fat16_getattr	获取 文件 属性	获得文件或目录的属性，不存在的情况下返回-ENOENT
fat16_readdir	读取 FAT16 文件 目录	if (strcmp(path, "/") == 0)，将root directory下的文件或目录填充到buffer中；否则，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填充到buffer中
fat16_read	读 FAT16 文件 内容	通过调用read_path( )实现

- 阅读libfuse源码，试解释本实验中使用到的fuse\_main()函数，要求至少追踪到fuse\_session\_loop()函数的调用并解释出此函数执行的内容
  - 在用户态程序调用 fuse\_main\_common( ) 时，先调用 fuse\_setup\_common( )，该函数先解析用户态程序传递过来的参数，然后调用 fuse\_tearardown\_common( )，该函数是 fuse\_kern\_mount( )函数的封装。
  - fuse\_kern\_mount( )函数中调用fuse\_mount\_fusermount( )，使用 socketpair( )创建一个UNIX域套接字，然后使用创建子进程执行 fusermount 程序，将 FUSE\_COMMFD\_ENV 环境变量中套接字的一端传递给它。
  - fusermount( )确保 fuse 模块已经被加载，然后打开 /dev/fuse 并通过一个UNIX套接字发送文件处理句柄。父进程等待子进程执行完毕回收，然后返回 fuse\_mount\_fusermount( )函数。
  - fuse\_kern\_mount( )通过返回文件句柄给 fuse\_kern\_chan\_new( ) 负责处理内核数据，然后返回到 fuse\_mount\_common( )函数。
  - fuse\_setup\_common( )函数调用 fuse\_new\_common( )，fuse\_new\_common( )函数分配 fuse的数据结构，存储并维护一个文件系统数据镜像缓存cached，返回到 fuse\_main( )。
  - 最后，fuse\_main\_common( )调用 fuse\_loop( )或 fuse\_loop\_mt( )，这两个函数可从设备读取文件系统调用，调用 fuse\_main\_common( )之前调用存储在 fuse\_operations 结构体中的用户态函数。这些调用的结果回写到设备

## 五、实验总结

- 学习了Linux操作系统下FAT-16文件系统的实现
- 实验难度较大，能力有限，只完成了只读文件系统内容
- 对不同文件系统的实现、功能和优缺点有了更清晰的认识