实验四

实现一个fat16文件系统 Part 1

邓龙

2022-06-06

实验目标

- 掌握文件系统的基本功能与工作原理（理论基础）
- 基于FUSE接口实现一个fat16文件系统（必做部分），包含以下四部分功能：
  - 文件目录与文件的读（只读文件系统，基础）
  - 文件/目录的创建与删除（基础）
  - 文件的写（进阶）

- 在实现fat16文件系统的基础上，引入文件系统的重要问题研究（选做部分）
  - 文件系统的性能优化（开放）

实验环境

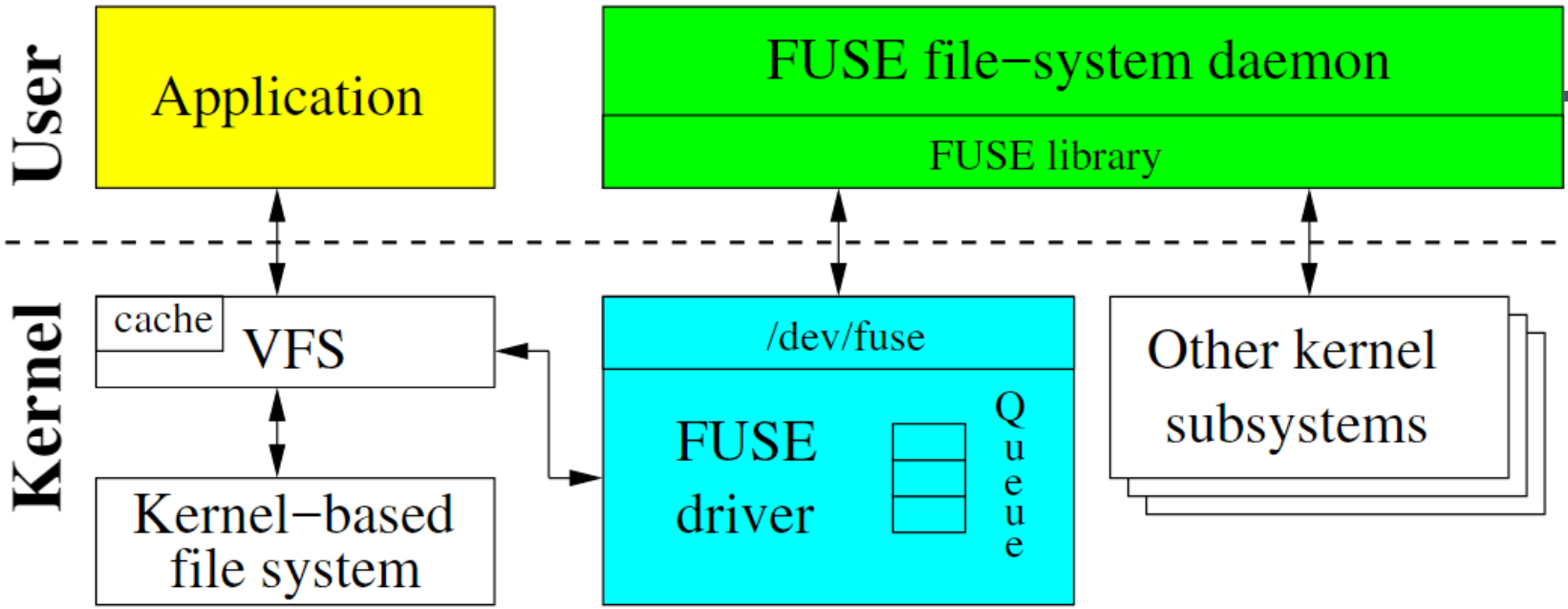
- 硬件环境：
  - 可以运行linux的个人电脑、实验室电脑
  - 注：vlab中没有提供设备挂载权限，无法进行本实验
- 软件环境：
  - 用于运行Linux环境的虚拟机：Vmware/VirtualBox
  - OS：Ubuntu 20.04 LTS
  - Linux内核版本：5.9.0及以上（本实验没有更换内核的要求）
  - 运行库：libfuse 3

实验安排

- 6.6 ——发布实验文档，讲解基础部分：
  - FUSE与fat16基础概念、原理
  - 实验框架代码讲解
  - 只读文件系统的实现思路
  - 文件创建删除的实现思路
  - 现场演示、实验检查方式
- 6.13——讲解进阶部分与选做部分，检查实验
- 6.20——检查实验
- 6.27——检查实验

目录

FUSE 用户态文件系统

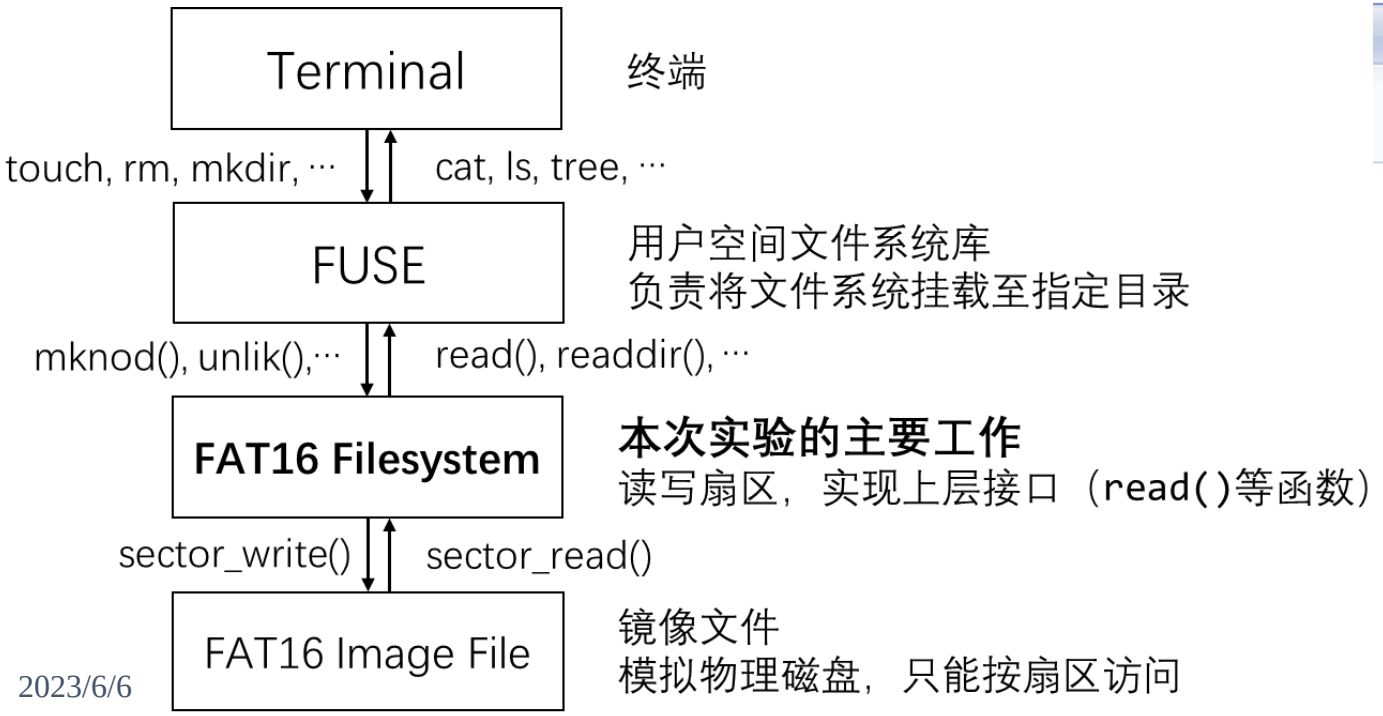


用户态的代码实现

- FUSE——Filesystem in Userspace 用户态文件系统
  - 在Linux 2.6.14以后，Kernel中提供了**FUSE内核模块**作为用户与VFS交互的中间件
  - 用户根据FUSE提供的**接口**实现对应的文件操作
  - 挂载后，对VFS的操作会经过FUSE，执行用户定义的行为

FUSE 接口介绍

- FUSE——Filesystem in Userspace 用户态文件系统
  - 基于FUSE实现——调用libfuse库，补充对应operation的代码
  - http://libfuse.github.io/doxygen/structfuse__operations.html
  - 必须要补充的操作实验代码中已经列出



libfuse

Main Page | Data Structures | Files

fuse_operations Struct Reference

```
#include <fuse.h>
```

Data Fields

```
int(*  getattr )(const char *, struct stat *, struct fuse_file_info *fi)
int(*  readlink )(const char *, char *, size_t)
int(*  mknod )(const char *, mode_t, dev_t)
int(*  mkdir )(const char *, mode_t)
int(*  unlink )(const char *)
int(*  rmdir )(const char *)
int(*  symlink )(const char *, const char *)
int(*  rename )(const char *, const char *, unsigned int flags)
int(*  link )(const char *, const char *)
```

FUSE 接口介绍

- FUSE——Filesystem in Userspace 用户态文件系统
  - 基于FUSE实现——调用libfuse库，补充对应operation的代码
  - http://libfuse.github.io/doxygen/structfuse__operations.html
  - 必须要补充的操作实验代码中已经列出

```
1329  struct fuse_operations fat16_oper = {
1330      .init = fat16_init,
1331      .destroy = fat16_destroy,
1332      .getattr = fat16_getattr,
1333
1334      // TASK1: tree [dir] / ls [dir] ; cat [file] / tail [file] / head [file]
1335      .readdir = fat16_readdir,
1336      .read = fat16_read,
1337
1338      // TASK2: touch [file]; rm [file]
1339      .mknod = fat16_mknod,
1340      .unlink = fat16_unlink,
1341      .utimens = fat16_utimens,
1342
1343      // TASK3: mkdir [dir] ; rm -r [dir]
1344      .mkdir = fat16_mkdir,
1345      .rmdir = fat16_rmdir,
1346
1347      // TASK4: echo "hello world!" > [file] ;  echo "hello world!" >> [file]
1348      .write = fat16_write,
1349      .truncate = fat16_truncate
1350      };
```

libfuse

Main Page | Data Structures | Files

fuse_operations Struct Reference

#include <fuse.h>

Data Fields

int(*	**getattr**)(const char *, struct stat *, struct **fuse_file_info** *fi)
int(*	**readlink**)(const char *, char *, size_t)
int(*	**mknod**)(const char *, mode_t, dev_t)
int(*	**mkdir**)(const char *, mode_t)
int(*	**unlink**)(const char *)
int(*	**rmdir**)(const char *)
int(*	**symlink**)(const char *, const char *)
int(*	**rename**)(const char *, const char *, unsigned int flags)
int(*	**link**)(const char *, const char *)

实验运行介绍

- 我们的任务实现了一个 simple_fat16 程序，如何运行?
- 将文件系统 **挂载** 到某个目录下

```
./simple_fat16 -s -d ./fat16 --img=./fat16.img
```

- 上述命令将我们的文件系统挂载到 ./fat16 目录下
  - 参数 –s 表示单线程，防止并发访问问题
  - 参数 –d 表示Debug 模式，程序会保持前端运行，并输出调试信息
  - ./fat16 是我们要挂载到的目录
  - --img=./fat16.img 指定了fat16.img文件作为我们读取的镜像

实验运行介绍

- 新开一个终端，访问./fat16目录，即可访问我们的文件系统：

```
# ldeng @ DL-LAPTOP-G14 in ~/oslab4 on git:master x [14:24:28]
$ ./simple_fat16 -d ./fat16 --img=./fat16.img
FUSE library version: 3.10.5
nullpath_ok: 0
unique: 2, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.34
flags=0x33fffffb
max_readahead=0x00020000
   INIT: 7.31
   flags=0x0040f039
   max_readahead=0x00020000
   max_write=0x00100000
   max_background=0
   congestion_threshold=0
   time_gran=1
   unique: 2, success, outsize: 80
```

终端1：运行我们的程序

```
ldeng@DL-LAPTOP-G14:~/oslab4$ ls ./fat16
large.txt  small  tree
```

终端2：查看./fat16目录下的文件

```
   unique: 6, success, outsize: 32
unique: 8, opcode: READDIRPLUS (44), nodeid: 1, insize: 80, pid
readdirplus[0] from 0
   unique: 8, success, outsize: 504
unique: 10, opcode: LOOKUP (1), nodeid: 1, insize: 50, pid: 350
LOOKUP /large.txt
getattr[NULL] /large.txt
   NODEID: 2
   unique: 10, success, outsize: 144
```

终端1的输出（部分）：ls命令被转换为调用FUSE文件系统中fat16_readdir、和一系列fat16_getattr 函数

FAT文件系统

- FAT——File Allocation Table 文件分配表
  - 维护一个表来记录文件分配信息
  - FAT12/16/32 —— 数字代表簇的地址长度
  - MS-DOS 6.x及以下版本使用FAT16

A block is named a **cluster**.

File System	FAT12	FAT16	FAT32
Cluster addr length	12 bits	16 bits	32 bits (28?)
Number of clusters	4K	64K	256M

名词	使用场景	释义
簇 (cluster)	文件系统	文件的最小空间分配单元，通常为若干个扇区，每个文件最小占用一个簇
扇区 (sector)	磁盘管理	磁盘上的磁道被等分为若干个弧段，这些弧段被称为扇区，对文件系统的读写以扇区为基本单位

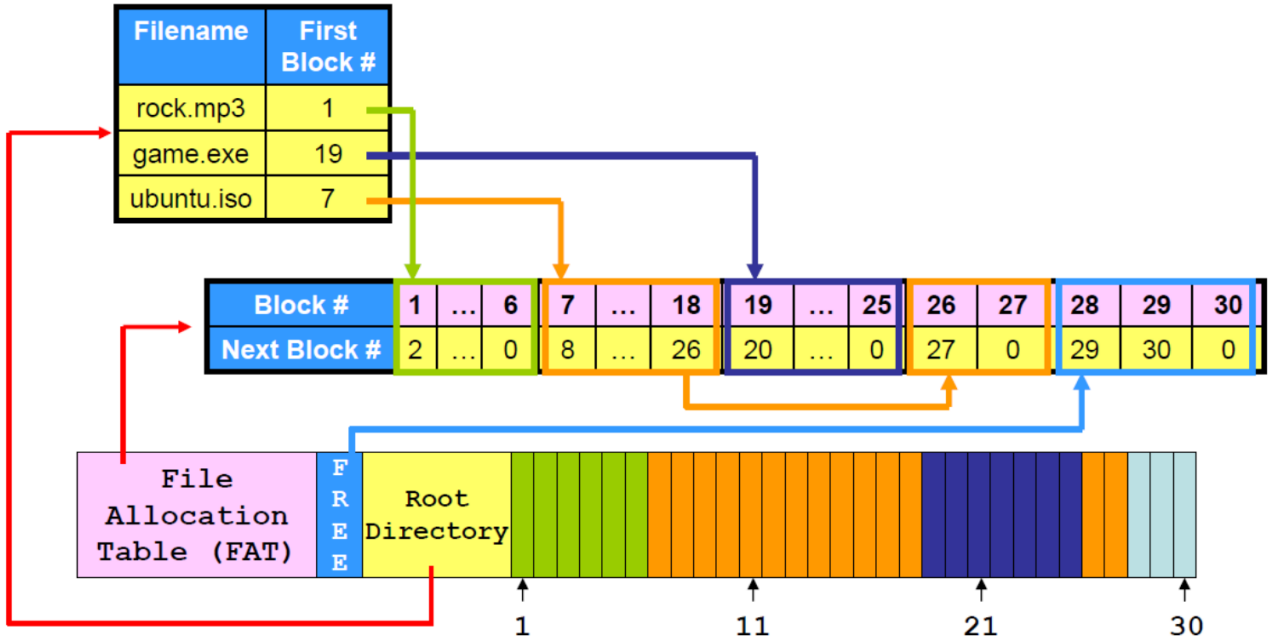
簇与扇区

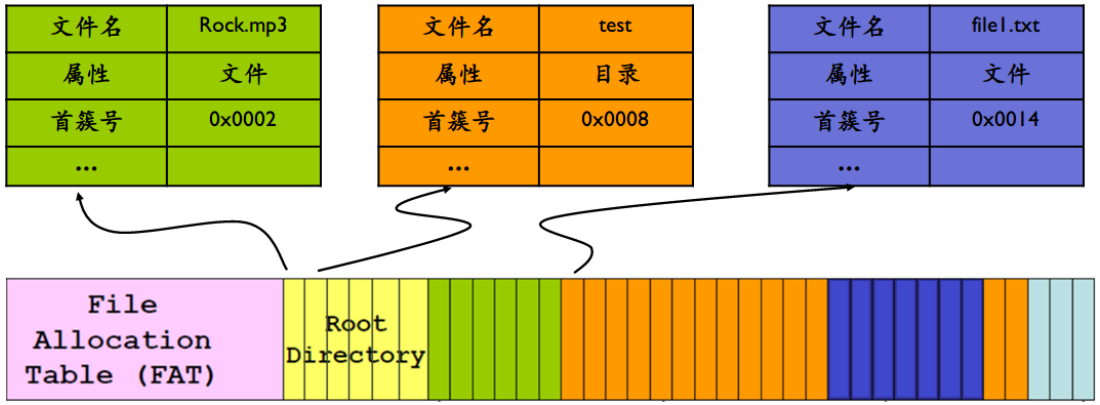
内容	主引导区	文件系统信息扇区	额外的保留空间	文件分配表(FAT表1)	文件分配表2(FAT表2)	根目录(Root Directory)	数据区
大小(Bytes)	保留扇区数*扇区大小			FAT扇区数* 扇区大小	FAT扇区数* 扇区大小	根目录条目数*文件条目大小	剩下的磁盘空间

FAT16的扇区划分

FAT文件系统

- FAT——File Allocation Table 文件分配表





内容	主引导区	文件系统信息扇区	额外的保留空间	文件分配表(FAT表1)	文件分配表2(FAT表2)	根目录(Root Directory)	数据区
大小(Bytes)	保留扇区数*扇区大小			FAT扇区数 * 扇区大小	FAT扇区数* 扇区大小	根目录条目数 *文件条目大小	剩下的磁盘空间

FAT16的扇区划分

FAT16文件系统的扇区划分

- FAT——File Allocation Table 文件分配表
  - 主引导区：引导程序、BPB分区参数记录表
    - BPB：起始扇区、结束扇区、存储格式、根目录大小、FAT个数、分配单元的大小
  - FAT表：簇的链表，每个簇对应位置存储了下一个簇的位置
  - 根目录：以32字节目录表项为单位记录根目录下的文件与子目录
  - 数据区：写入文件内容，在删除文件时并不会立即清空对应的数据区

内容	主引导区	文件系统信息扇区	额外的保留空间	文件分配表(FAT表1)	文件分配表2(FAT表2)	根目录(Root Directory)	数据区
大小(Bytes)	保留扇区数*扇区大小			FAT扇区数* 扇区大小	FAT扇区数* 扇区大小	根目录条目数*文件条目大小	剩下的磁盘空间

FAT16的扇区划分

FAT16文件系统的DBR扇区

- BPB记录的关键信息：
  - BPB_BytsPerSec 每扇区字节数
  - BPB_SecPerClus 每簇扇区数
  - BPB_RsvdSecCnt 保留扇区的数目
  - BPB_RootEntCnt 根目录项数
  - BPB_FATSz16 一个FAT表所占扇区数
- 可以根据BPB算出FAT、根目录和数据区的偏移地址

内容	主引导区	文件系统信息扇区	额外的保留空间	文件分配表(FAT表1)	文件分配表2(FAT表2)	根目录(Root Directory)	数据区
大小(Bytes)	保留扇区数*扇区大小			FAT扇区数* 扇区大小	FAT扇区数* 扇区大小	根目录条目数*文件条目大小	剩下的磁盘空间

FAT16的扇区划分

FAT16文件系统的FAT表

- FAT表：簇的链表，每个簇大小为2个字节，对应位置存储了下一个簇的位置
- 0和1个簇对应的“F8 FF FF FF”为介质描述单元，因此第一个可用簇为2号簇
- 假设一个文件的首簇号是8，对应的内容为0x0009，即下一个簇为9号簇

modem | factory | asdf

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ANSI ASC
00011264	F8	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	øÿÿÿÿÿÿÿÿÿÿÿÿ
00011280	09	00	0A	00	0B	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿ
00011296	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011312	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011328	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011344	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011360	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011376	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011392	41	00	42	00	FF	FF	44	00	45	00	FF	FF	47	00	48	00	A B ÿÿD E ÿÿG
00011408	FF	FF	4A	00	4B	00	FF	FF	4D	00	4E	00	FF	FF	50	00	ÿÿJ K ÿÿM N ÿÿ
00011424	51	00	FF	FF	53	00	54	00	FF	FF	56	00	57	00	FF	FF	Q ÿÿS T ÿÿV W
00011440	59	00	5A	00	5B	00	FF	FF	5D	00	5E	00	FF	FF	60	00	Y Z [ÿÿ] ^ ÿÿ
00011456	61	00	62	00	FF	FF	64	00	65	00	66	00	FF	FF	68	00	a b ÿÿd e f ÿÿ

取值	对应簇含义
0x0000	未使用
0x0002-0xFFEF	已分配的下一个簇
0xFFF0-0xFFF6	系统保留
0xFFF8-0xFFFF	文件结束

FAT16文件系统的根目录

- 根目录：存储文件和子文件夹的目录项，子目录都存在数据区（特殊的“文件”）
- 每个目录项大小为32字节，关键数据有：

名称	偏移(字节)	长度(字节)	说明
DIR_Name	0x00	11	文件名（前8个字节为文件名，后3个为拓展名）
DIR_Attr	0x0B	1	文件属性，取值为0x10表示为目录，0x20表示为文件
DIR_WrtTime	0x16	2	文件最近修改时间
DIR_WrtDate	0x18	2	文件最近修改日期
DIR_FstClusLO	0x1A	2	文件首簇号(FAT32用作第一个簇的两个低字节)
DIR_FileSize	0x1C	4	文件大小

- 若一个目录项0x0偏移处取值为0x0，则表示目录项为空；

  取值为0xE5，表明曾被使用但现在已经删除。

目录

实验代码说明

- 包含文件：
  - fat16.h：文件系统数据结构和函数的定义
  - simple_fat16.c：基础部分的代码，补充挖空的代码即可完成
  - simple_fat16_fixed.c: 不需要更改的代码，实现了sector_read、sector_write 和 fuse 的启动等。

读写IO

- 以扇区为单位对齐，读出或写入数据

```
int sector_read(sector_t sec_num, void *buffer);

int sector_write(sector_t sec_num, const void *buffer)
```

- sec_num：扇区号
- buffer：用于存放数据的缓冲区

```
int func1() {   //读扇区示例
    char sector_buffer[512];
    sector_read(12, sector_buffer);
}

int func2() {   //写扇区示例
    char sector_buffer[512];
    memset(sector_buffer, 'c', sizeof(sector_buffer));
    sector_read(12, sector_buffer);
}
```

目录

只读文件系统需要实现的代码接口

- 读目录 `int fat16_readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi, enum fuse_readdir_flags flags)`

- 读文件 `int fat16_read(const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi)`

```
/**
 * @brief 读取path对应的目录，结果通过filler函数写入buffer中
 *
 * @param path     要读取目录的路径
 * @param buf   结果缓冲区
 * @param filler   用于填充结果的函数，本次实验按filler
 *                 你也可以参考<fuse.h>第58行附近的函数
 * @param offset   忽略
 * @param fi       忽略
 * @return int     成功返回0，失败返回POSIX错误代码的负
 */
int fat16_readdir(const char *path, void *buf, fuse_
                  struct fuse_file_info *fi, enum
```

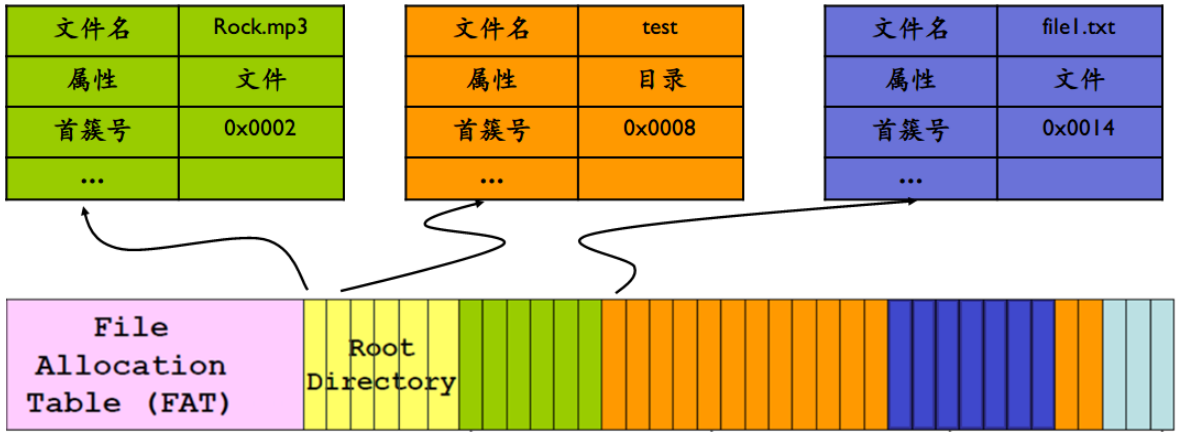
```
/**
 * @brief 从path对应的文件的offset字节处开始读取size字节的数据到buffer中，并返回实际读取的字节数。
 * Hint: 文件大小属性是Dir.DIR_FileSize。
 *
 * @param path     要读取文件的路径
 * @param buffer   结果缓冲区
 * @param size     需要读取的数据长度
 * @param offset   要读取的数据所在偏移量
 * @param fi       忽略
 * @return int     成功返回实际读写的字符数，失败返回0。
 */
int fat16_read(const char *path, char *buffer, size_t size, off_t offset,
               struct fuse_file_info *fi) {
```

读目录的原理

- 举例：读取目录 /a/b/c
  - 运行命令行： **`ls ./fat16/a/b/c`**
  - FUSE 调用相应函数： **`readdir("/a/b/c", buf, filler, offset, ...)`**
    - （我们可以忽略除了 path 和 filler 以外的参数，用不到）
  - 我们实现的该函数完成：
    - 找到路径 /a/b/c 目录里有哪些东西
      - 假设有 apple、banana、orange 三个文件或目录
    - 调用 filler（是个函数指针），告诉fuse有这个三个东西：
      - `filler(buf, "apple", NULL, 0, 0);`
      - `filler(buf, "banana", NULL, 0, 0);`

读目录的原理

- 如何找到路径 /a/b/c 目录里的东西?
  - 根据 FAT16 文件系统的结构。
  - 如果要找 / 里的东西? **根目录区域**
  - 那/a 里的东西?
    - 先在 / 里找到 a 对应的目录项，目录项里有 /a 对应的簇号
    - 找到相应的簇，里面存放了 /a 目录里文件的目录项
  - /a/b ?
    - 在 /a 对应的簇里，找到 b 对应的目录项，读取相应簇……
  - /a/b/c …
- 一级一级往下找!



寻找目录项

- 在文件系统里执行几乎任何功能，都需要找到对应文件的目录项！
- 提供的代码中，寻找目录项的功能被抽象为 find_entry 函数。
  - 实际上调用了 find_entry_internal 函数。

```
int find_entry(const char* path,
DirEntrySlot* slot)
```

```
typedef struct {
    DIR_ENTRY dir;
    sector_t sector;
    size_t offset;
} DirEntrySlot;
```

find_entry: 找到path对应的目录项位置，并把目录项内容和目录项所在位置（扇区和偏移记录在slot中）

```
/**
 * @brief 找到path所对应路径的目录项，如果最后一级路径不存在，则找到能创建最后一集文件/目录的空目录项。
 *
 * @param path
 * @param slot
 * @param remains
 * @return int 成功返回0，失败返回错误代码的负值，可能的错误参见brief部分。
 */
int find_entry_internal(const char* path,
DirEntrySlot* slot, const char** remains)
```

寻找目录项

- 任务一的关键：实现 find_entry_internal
  - 为什么不直接实现 find_entry：方便后续任务，不用写重复代码
    - readdir 要找目录项
    - mknod 要找空槽，并判断有无重复文件
    - unlink 要找目录项，和目录项对应的位置
    - 全部可以用 find_entry_internal 实现
- 定义：
```
int find_entry_internal(const char* path, DirEntrySlot* slot,
const char** remains)
```
  - path：要找的路径；
  - slot：找到的目录项（如果未找到，返回找到的第一个空槽）；
  - remains：路径中未找到的部分（寻找/a/b/c,结果/a里没有b）

寻找目录项

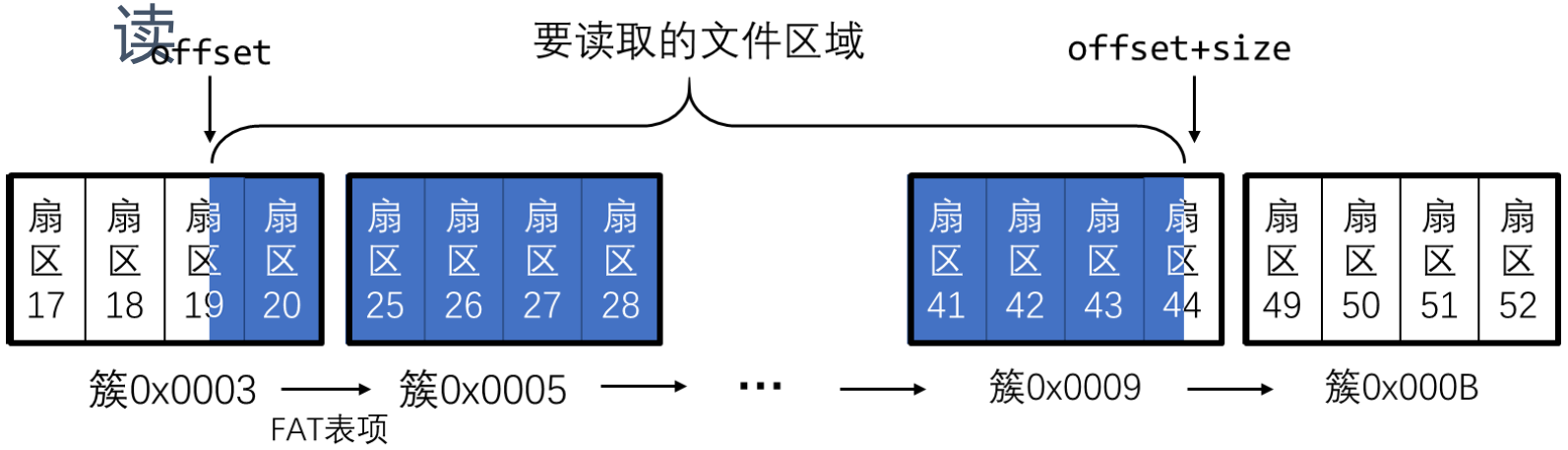
- find_entry_internal 的定义
  - 返回值三种结果：
    - 找到path：返回 FIND_EXIST
    - 没找到path，但找到空槽：FIND_EMPTY
    - 发生错误：负数
      - 一些常见错误
      - -ENOTDIR：要找/a/b/c，结果/a/b是个文件而不是目录
      - -ENOTENT：要找/a/b/c，结果/a/b不存在
        - 注意如果/a/b存在，c不存在，在find_entry_internal中不认为是错误，找到空槽即可（为何要这样？）

读目录

- 读目录：
  - 从给定路径目录的第一个扇区开始读（需要先找到子目录的地址等信息）
  - 每次读一个条目（32字节）
  - 读到属性为文件或文件夹，且没有删除标记的文件时
    - 使用filler函数填充文件名（需先decode）到返回buffer中
  - 条目数量上限可增长，会存在跨扇区和跨簇的问题
  - 读到0x00的空白表项结束

读文件过程需要关注的问题

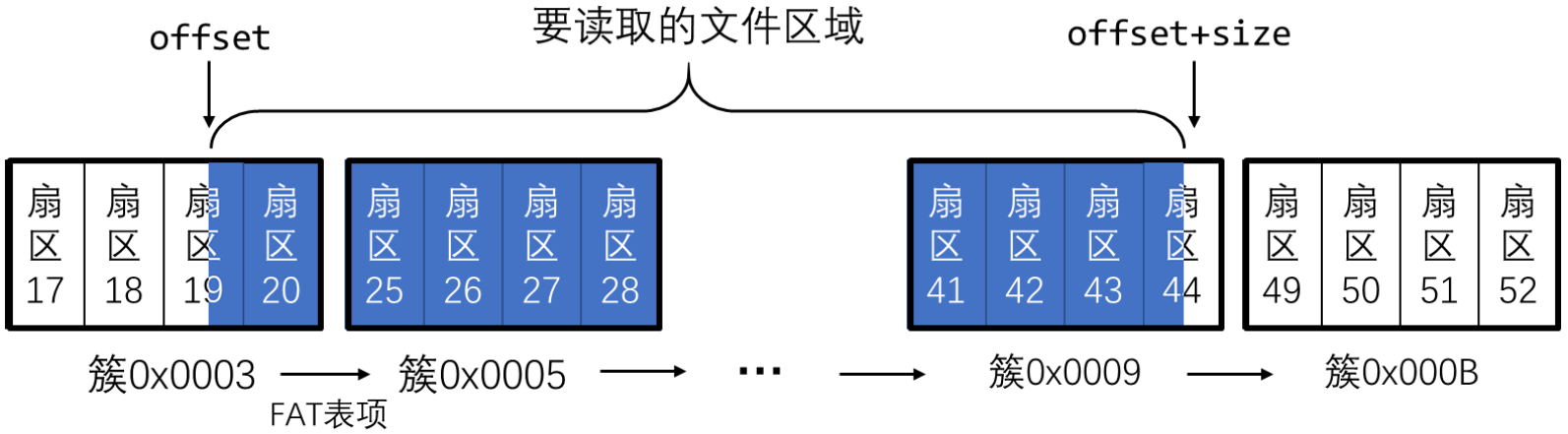
- 除了读根目录，其他读（子目录、文件）都需要注意跨扇区和跨簇的问题
- 跨扇区读：
  - 在读一个扇区读到结束后，需要顺着读下一个扇区
- 跨簇读：
  - 若读到了簇的最后一个扇区结束，则需要**找到下一个簇**的首扇区继续读



read()的读取范围

读文件

- 读文件流程：
  - 从文件的所需第一个扇区开始读（需要先找到文件的地址等信息）
  - 每次读一个扇区，注意**起始和结束对应扇区读出的内容不一定都需要**
  - 会存在跨扇区和跨簇的问题，读到结束地址，或者读到文件末尾结束
  - 越界读可以自行选择直接返回0报错或者读到文件末尾返回来处理



目录

创建删除文件需要实现的代码接口

- 功能函数
  - 创建新目录项 `dir_entry_create`（已实现）
  - 写入目录项：`int dir_entry_write(DirEntrySlot slot)`
  - 释放簇 `int free_clusters(cluster_t clus)`
    - 释放簇链到结尾。
- 接口函数
  - 创建文件 `int fat16_mknod(const char *path, mode_t mode, dev_t dev)`
  - 删除文件 `int fat16_unlink(const char *path)`

创建文件

- 创建文件的主要流程
  - 找到文件的父目录路径，查找父目录中可用的目录项
    - 如果正确实现了 find_entry_internal，这两步已由调用该函数的 find_empty_slot 完成
  - 创建新的目录项（已完成）
    - 写入文件名、文件属性、首簇号(0x0)、文件大小(0)
  - 写入目录项
    - 读取目录项所在扇区
    - 修改扇区中目录项对应位置
    - 写回目录项

删除文件

- 删除文件的主要流程
  - 找到文件的首簇地址
    - 循环回收该文件占用的所有簇（修改FAT表，不擦除簇的数据）
    - 注意FAT表有两个，都要修改
  - 找到文件所在的目录项
    - 和创建文件的流程类似
  - 将对应目录项标记为删除

创建、删除目录

- 过程和创建、删除文件类似
- 创建目录：
  - 要给目录分配一个簇（实现 `alloc_clusters` ）
  - 给目录下添加 . 和 .. 两个目录项（和创建文件最后一步过程类似）
- 删除目录：
  - 要确认目录为空（不为空直接返回错误）
  - 确认目录为空的方法和 readdir 类似

目录

实验检查方式

- 自动化检查脚本
  - `./test/run_test.sh`
  - 自动生成测试镜像
  - 自动挂载你的程序
  - 对你的程序进行一系列测试
  - 报告测试结果
- 运行时需要输入密码（但不要用sudo运行）
- 需要首先安装pytest
  - `sudo apt install python3-pip`
  - `pip install pytest`

实验检查方式

- 自动化检查脚本
  - 结果示例（成功）：
    - TestFat16List 读目录
    - TestFat16Read 读文件
    - TestFat16CreateRemove
      - 创建删除文件、目录

```
============================= test session starts ==============================
platform linux -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0 -- /home/ldeng/m
iniconda3/bin/python
cachedir: .pytest_cache
rootdir: /home/ldeng/oslab4/test
collected 13 items

fat16_test.py::TestFat16List::test1_list_tree PASSED                    [  7%]
fat16_test.py::TestFat16Read::test1_file_exists PASSED                   [ 15%]
fat16_test.py::TestFat16Read::test2_seq_read_large_file PASSED           [ 23%]
fat16_test.py::TestFat16Read::test3_rand_read_large_file PASSED          [ 30%]
fat16_test.py::TestFat16Read::test4_seq_read_small_files PASSED          [ 38%]
fat16_test.py::TestFat16CreateRemove::test1_file_create_root PASSED      [ 46%]
fat16_test.py::TestFat16CreateRemove::test2_file_remove_root PASSED      [ 53%]
fat16_test.py::TestFat16CreateRemove::test3_dir_create_root PASSED       [ 61%]
fat16_test.py::TestFat16CreateRemove::test4_dir_remove_root PASSED       [ 69%]
fat16_test.py::TestFat16CreateRemove::test5_file_create_subdir PASSED    [ 76%]
fat16_test.py::TestFat16CreateRemove::test6_file_remove_subdir PASSED    [ 84%]
fat16_test.py::TestFat16CreateRemove::test7_create_tree PASSED           [ 92%]
fat16_test.py::TestFat16CreateRemove::test8_remove_tree PASSED           [100%]

============================== 13 passed in 0.73s ==============================
> $ fusermount -u ./fat16
(base)
# ldeng @ DL-LAPTOP-G14 in ~/oslab4 on git:master x [15:25:21]
$
```

实验检查方式

- 自动化检查脚本
  - 结果示例（失败）：
  - 注意最后提示了错误的测试样例
    - TestFat16CreateRemove::
    - test1_file_create_root
    - 在根目录创建文件失败

```
============================= test session starts ==============================
platform linux -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0 -- /home/ldeng/m
iniconda3/bin/python
cachedir: .pytest_cache
rootdir: /home/ldeng/oslab4/test
collected 13 items

fat16_test.py::TestFat16List::test1_list_tree PASSED                    [  7%]
fat16_test.py::TestFat16Read::test1_file_exists PASSED                   [ 15%]
fat16_test.py::TestFat16Read::test2_seq_read_large_file PASSED           [ 23%]
fat16_test.py::TestFat16Read::test3_rand_read_large_file PASSED          [ 30%]
fat16_test.py::TestFat16Read::test4_seq_read_small_files PASSED          [ 38%]
fat16_test.py::TestFat16CreateRemove::test1_file_create_root FAILED      [ 46%]

================================== FAILURES ===================================
_____________ TestFat16CreateRemove.test1_file_create_root ______________

self = <fat16_test.TestFat16CreateRemove testMethod=test1_file_create_root>

    def test1_file_create_root(self):
        os.chdir(FAT_DIR)
        name = os.path.join(FAT_DIR, 'newfile.txt')
>       os.mknod(name, mode=0o666)
E       PermissionError: [Errno 1] Operation not permitted

/home/ldeng/oslab4/test/fat16_test.py:90: PermissionError
=========================== short test summary info ============================
FAILED fat16_test.py::TestFat16CreateRemove::test1_file_create_root - Permiss
ionError: [Errno 1] Operation not permitted
!!!!!!!!!!!!!!!!!!!!!! stopping after 1 failures !!!!!!!!!!!!!!!!!!!!!!
======================== 1 failed, 5 passed in 0.24s ========================
```


```
=========================== short test summary info ============================
FAILED fat16_test.py::TestFat16CreateRemove::test1_file_create_root - Permiss
ionError: [Errno 1] Operation not permitted
!!!!!!!!!!!!!!!!!!!!!! stopping after 1 failures !!!!!!!!!!!!!!!!!!!!!!
======================== 1 failed, 5 passed in 0.24s ========================
```

实验检查方式

- 第一部分十三个测试样例说明(1~8):
- TestFat16List::test1_list_tree: 读取tree目录下的文件（readdir)
- TestFat16Read::test1_file_exists: 读取的文件是否存在（readdir)
- TestFat16Read::test2_seq_read_large_file: 顺序读取大文件(read)
- TestFat16Read::test3_rand_read_large_file: 随机读取大文件(read)
- TestFat16Read::test4_seq_read_small_files: 顺序读取小文件(read)
- TestFat16CreateRemove::test1_file_create_root: 根目录创建文件(mknod)
- TestFat16CreateRemove::test2_file_remove_root: 根目录删除文件(unlink)
- TestFat16CreateRemove::test3_dir_create_root: 根目录创建目录(mkdir)

实验检查方式

- 第一部分十三个测试样例说明(9~13):
- TestFat16CreateRemove::test4_dir_remove_root: 根目录删除文件 (rmdir)
- TestFat16CreateRemove::test5_file_create_subdir:
  - 子目录创建文件 (mknod)
- TestFat16CreateRemove::test6_file_remove_subdir:
  - 子目录删除文件 (unlink)
- TestFat16CreateRemove::test7_create_tree:
  - 创建一系列的目录、子目录和文件的测试 (mkdir, mknod)
- TestFat16CreateRemove::test8_remove_tree:
  - 删除一系列目录、子目录和文件的测试 (unlink, rmdir)

谢谢！

Q&A

邓龙
2023-06-06