实验四 FAT文件系统的实现 Part1

> 本文档对一部分内容进行了修正。修正日志详见Part2文档的第一部分。

实验目标

- 熟悉文件系统的基本功能与工作原理（理论基础）
- 熟悉FAT16的存储结构，利用FUSE实现一个FAT文件系统：
  - 文件目录与文件的读（只读文件系统，基础）
  - 文件的创建与删除（基础）
  - 文件目录的创建与删除（进阶）
  - 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（进阶，选做）
  - 如何做文件系统的错误修复——三备份与纠删码
  - 如何做文件系统的日志恢复——元数据日志

实验环境

- VMware / VirtualBox
- OS: Ubuntu 20.04.4 LTS

实验时间安排

注：此处为实验发布时的安排计划，请以课程主页和课程群内最新公告为准

- 5月13日晚实验课，发布、讲解基础部分及检查实验
- 5月20日晚实验课，发布、讲解进阶部分及检查实验
- 5月27日晚实验课，检查实验
- 6月3日晚实验课，实验补检查

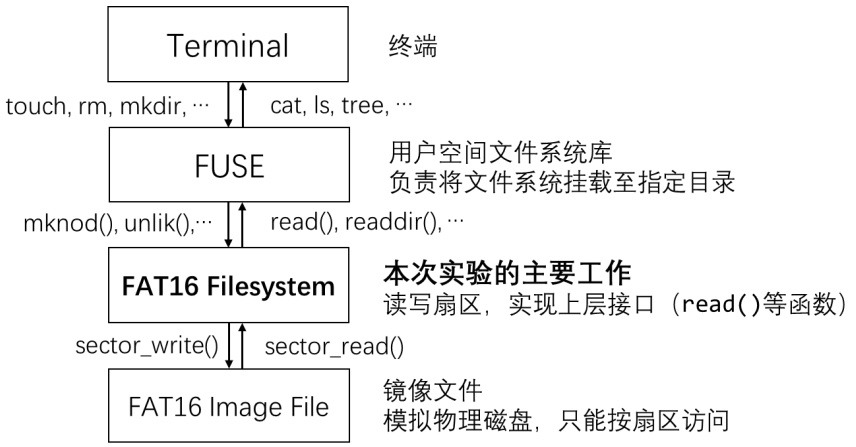
友情提示

- **本次实验总工作量较大，请尽早开始实验。**
- **本次实验有600余行注释，以解决实验文档对于代码实现的提示量不足的问题，代码实现过程中务必注意看注释！看注释！看注释！**
- **实验多个任务难度依次递增，靠前的任务代码提示较为详尽，你可以相对容易地获得这部分分数。**
- **实验多个任务之间较为独立，你也可以选择只完成靠后的任务。**
- **本实验文档较为详尽，是为了尽可能提前解决大家实验中可能遇到的问题。你并不一定需要完全阅读所有实验文档才能完成实验、获得分数。**

第一部分 实验环境

1.1 实验框架概述

在本次实验中，我们将实现简单的FAT16文件系统，来读写一个FAT16镜像文件。本次实验的大体框架如下图。



首先，实验文件中提供了一个已经格式化为FAT16格式的镜像 `fat16_test.img` ，这个文件就是本次实验的“硬盘”，这块“硬盘”中预先存储了一些目录和文件，需要我们通过实现的文件系统去读取它们。

为了隐藏文件系统与操作系统的交互细节，将实验重点聚焦在文件系统的格式和实现上。本实验采用FUSE库以便在用户空间实现文件系统。FUSE库能够帮助处理文件系统和内核交互的部分，将终端命令行对文件系统的访问，转化成对 `fuse` 提供的文件系统接口的调用。所以**本次实验中，只需要实现相应的 文件系统接口** ，就可以在终端中像访问其它目录一样访问实验中提供的“磁盘”。

在实验提供的 `simple_fat16.c` 文件中，定义了所有需要完成的接口，包括 `readdir` 、 `read` 、 `mknod` 等共8个函数，这些函数的具体含义将在实验内容部分详细的进行介绍。同学们只需要根据FAT16文件系统的结构，实验要求和实验代码中的提示，**补全** `simple_fat16.c` **中对应的函数**，实现相应的文件系统功能，即可获得相应的分数。

值得注意的是，磁盘是一种块设备，任何对磁盘的访问都是以扇区为单位的。为了模拟这种访问，实验提供的代码中，抽象了 `sector_read` 和 `sector_write` 函数，分别用于模拟读扇区和写扇区。因此，本次实验**所有对镜像文件的读写操作，最终都是需要通过调用** `sector_read` **和** `sector_write` **两个函数完成。**

1.2 实验流程概述

本次实验中，同学们需要完成以下步骤：

1. 安装并测试 `libfuse` 库。
2. 补充 `simple_fat16_part1.c` 中空缺的一个或多个函数代码（代码中用TODO进行了标注）。
3. 编译并运行代码，在终端中访问文件系统，测试对应功能。

后续实验文档将会给出了完成每一个步骤的所需的流程和背景知识。**如果同学们在实验中遇到问题，请先查阅实验文档的相应章节和 在线文档 。**

1.3 FUSE环境配置

FUSE（Filesystem in Userspace，用户态文件系统）是一个实现在用户空间的文件系统框架，通过FUSE内核模块的支持，使用者只需要根据FUSE提供的接口并实现具体的文件操作就可以实现一个用户态的、自定义的文件系统。 如果感兴趣，同学们可以在本文的附录中了解更多关于FUSE的信息。

1.3.1 配置FUSE环境

我们此处可以选择两种方式安装libfuse：

1. **方法一：**使用 Ubuntu的包管理器直接安装，**推荐大家使用这种方式**，较为简单快捷。

```
sudo apt install libfuse-dev libfuse2
```

2. **方法二：**下载源码并编译安装。

```
$ sudo apt install pkg-config pkgconf
$ wget -O libfuze-2.9.5.zip https://codeload.github.com/libfuse/libfuse/zip/fuse_2_9_5
unzip libfuze-2.9.5.zip
$ cd libfuse-fuse_2_9_5/
$ ./makeconf.sh
$ ./configure --prefix=/usr
$ make -j4
$ sudo make install
```

1.3.2 测试FUSE

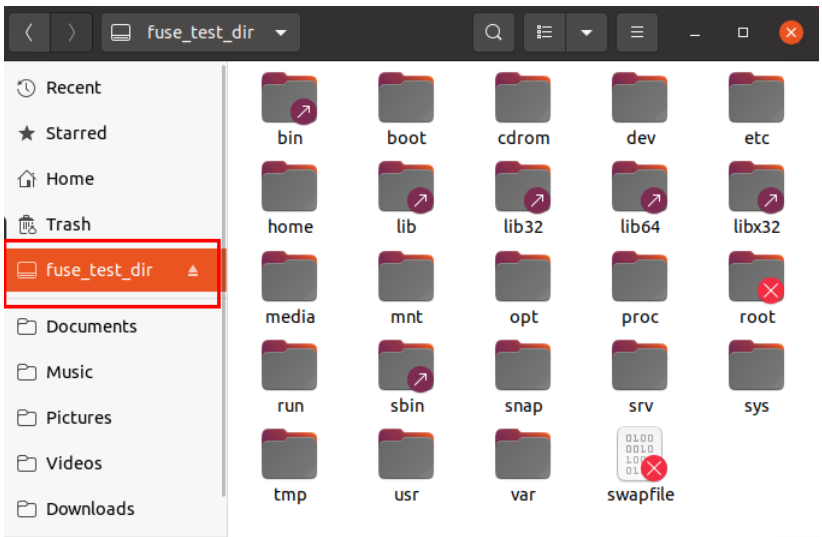
如果使用方法一安装 `libfuse`，**可以跳过本步骤**，或者单独下载 fusexmp.c 编译测试：

```
mkdir example; cd example
wget https://raw.githubusercontent.com/libfuse/libfuse/fuse_2_9_bugfix/example/fusexmp.c
gcc -o ./fusexmp fusexmp.c `pkg-config fuse --cflags --libs`
mkdir fuse_test_dir
./fusexmp -d fuse_test_dir
```

如果使用方法二编译安装 `libfuse`，可以通过 `libfuse-fuse_2_9_5/example` 下的fusexmp进行测试：

```
cd example
mkdir fuse_test_dir
./fusexmp -d fuse_test_dir
```

这时候在文件管理器中打开fuse_test_dir，可以看到当前Linux系统的根目录 `/` 被挂载到这个目录下，如下图：



结束测试可以直接在当前终端中Ctrl + C结束程序，或者在新的终端中输入：

```
fusermount -u fuse_test_dir
```

> **提示**：当执行用户自己实现的fuse程序时，如果出现错误（"fuse: bad mount point ...Transport endpoint is not connected"），可通过执行上面这条命令卸载对应的文件夹来解决。

第二部分 FAT文件系统初识

FAT（File Allocation Table）是“文件分配表”的意思。顾名思义，就是用来记录文件在磁盘中所处位置的表格。FAT对于磁盘的使用是非常重要的，假若丢失文件分配表，那么磁盘上的数据就会因无法定位而不能使用。不同的操作系统所使用的文件系统不尽相同，在个人计算机上常用的操作系统中，MS-DOS 6.x及以下版本使用FAT16。操作系统根据整个磁盘空间所需要的簇的数量来确定使用多大的FAT。其中簇是磁盘空间的配置单位，就象图书馆内一格一格的书架一样。FAT16使用了16位的空间来表示FAT表项（或者簇号），故称之为FAT16。

从上述描述中我们够得知，一个FAT16分区或者磁盘最多能够使用的簇数是2^16=65536个，因此簇大小是一个变化值，通常与分区大小有关，计算方式为：(磁盘大小/簇个数)向上按2的幂取整。在FAT16文件系统中，由于兼容性等原因，簇大小通常不超过32K，这也是FAT分区容量不能超过2GB的原因。

2.1 专有名词

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，每个文件最小将占用一个簇。
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，硬盘的读写以扇区为基本单位。

2.2 磁盘布局

一个FAT分区或磁盘的结构布局							
内容	引导扇区	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

一个FAT文件系统包括四个不同的部分：

- **保留扇区**。位于磁盘最开始的位置。 第一个保留扇区是**引导扇区**（分区启动记录）。引导扇区中通常包括了操作系统启动所需的一些信息和代码，也包含了一些**文件系统需要的元数据信息**，例如簇的大小，保留扇区的数目等，都保存在引导扇区中，所以需要特别关注，我们将在下面详细介绍。保留扇区还包括文件系统信息扇区）（仅FAT32使用）和额外的保留扇区，本次实验无需用到这些扇区。
    - 保留扇区包括一个称为基本输入输出参数块的区域（包括一些基本的文件系统信息尤其是它的类型和其它指向其它扇区的指针），通常包括操作系统的启动调用代码。保留扇区的总数记录在引导扇区中的一个参数中。引导扇区中的重要信息可以被DOS和OS/2中称为驱动器参数块的操作系统结构访问。
- **FAT区域**。它包含有两份文件分配表，这是出于系统冗余考虑，尽管它很少使用。它是分区信息的映射表，指示簇是如何存储的。本实验中两个FAT表都要修改。
- **根目录区域**。它是在根目录中存储文件和目录信息的目录表。在FAT32下它可以存在分区中的任何位置，但是在FAT16中它永远紧随FAT区域之后。根目录区域的大小一定是**整数个扇区大小**。
- **数据区域**。这是实际的文件和目录数据存储的区域，它占据了分区的绝大部分。通过简单地在FAT中添加文件链接的个数可以任意增加文件大小和子目录个数（只要有空簇存在）。然而需要注意的是每个簇只能被一个文件占有：如果在32KB大小的簇中有一个1KB大小的文件，那么31KB的空间就浪费掉了。

2.3 引导扇区

引导扇区包含很多条目，但我们只需关注和文件系统有关的部分，这里列出如下：

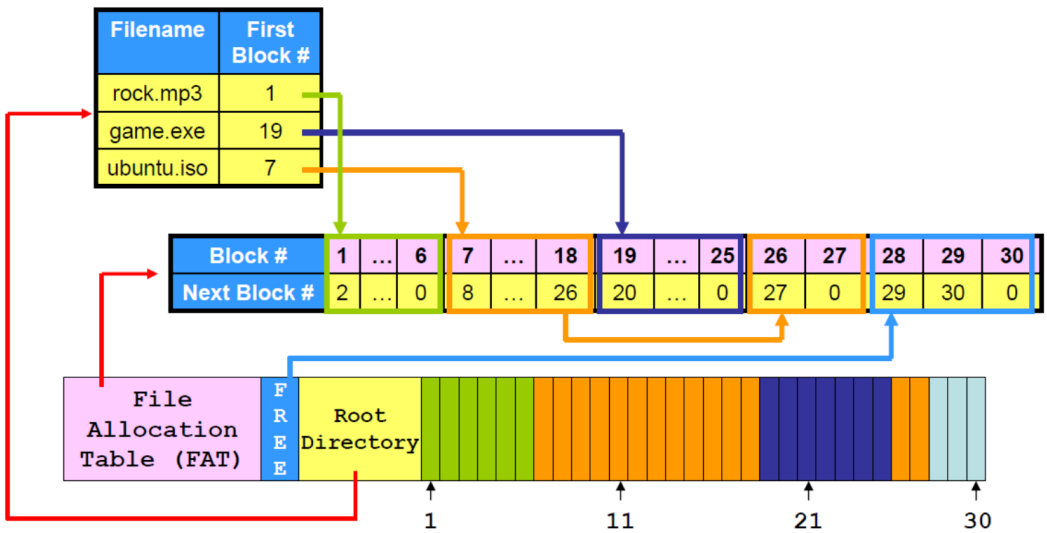
名称	偏移(字节)	长度(字节)	说明
...	...	...	...
BPB_BytsPerSec	0x0b	2	**每个扇区的字节数。基本输入输出系统参数块从这里开始。**
BPB_SecPerClus	0x0d	1	**每簇扇区数**
BPB_RsvdSecCnt	0x0e	2	**保留扇区数（包括引导扇区）**
BPB_NumFATS	0x10	1	**文件分配表数目，FAT16文件系统中为0x02,FAT2作为FAT1的冗余**
BPB_RootEntCnt	0x11	2	**最大根目录条目个数。一定是16的倍数，因为必须保证根目录区域对齐完整扇区。**
...	...	...	...
BPB_FATSz16	0x16	2	**每个文件分配表的扇区数（FAT16专用）**
...	...	...	...

完成实验仅需理解上述条目。引导扇区的完整结构详见附录。

通过这些条目，我们可以计算出一些偏移量，你可以结合2.4的布局示意图理解这些偏移量：

- FAT1偏移地址：保留扇区之后就是FAT1。因此可以得到，FAT1的偏移地址就是 `BPB_RsvdSecCnt` * `BPB_BytsPerSec`
- 根目录偏移地址： FAT1表后两个FAT表地址就是根目录区，即FAT1偏移地址 + `BPB_NumFATS` *`BPB_FATSz16`* `BPB_BytsPerSec` 。
- 数据区偏移地址：根目录偏移地址+根目录大小，即根目录偏移地址+ `BPB_RootEntCnt` * `BYTES_PER_DIR`

2.4 文件存储

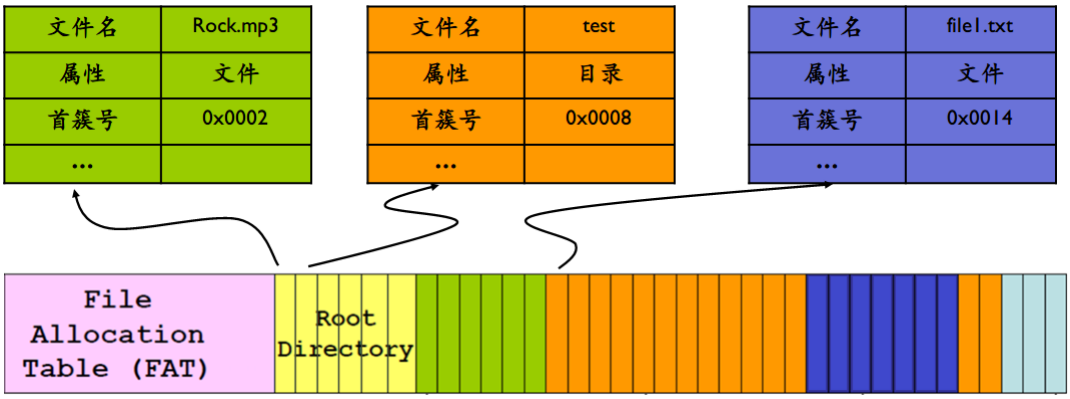


在FAT文件系统中，文件分为两个部分，第一个部分为目录项，记录该文件的文件名，拓展名，以及属性，首簇号等元数据信息（文件夹也是一种文件，它的目录项和普通文件结构相同）；第二部分为实际存储内容，若为文件，则存储文件内容，若为文件夹，则存储子文件夹的文件目录项。

- 例一，文件处于Root目录下 (假设该文件路径为/rock.mp3)
  若文件处于Root目录下，那么该文件的目录项将会存储在`Root Directory`区域中，其中记录了该文件的文件名，拓展名，以及第一个文件簇的地址。
  FAT Table中存储了所有可用的簇，通过簇地址查找使用情况，我们能够得知该文件的下一个簇地址，若下一个簇地址为END标记符，则表示该簇为最后一个簇。通过查询FAT Table，我们能够得知该文件所有簇号。
- 情况二，文件处于Root目录下子文件夹中 (假设该文件路径为/test/file1.txt)，那么可知test目录项在`Root Directory`中，而rock.mp3的目录项将会存储在test文件的数据区域。

2.4.1 目录项

文件目录是文件的元数据信息，存储在`Root directory`以及数据区中，下图是两个文件/rock.mp3和/test/file1.txt的文件目录项存储示例：



每个文件目录项的大小为0x20字节（32字节），每一项的结构如下：

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。注意第一位取值会有特殊含义，参见本表后的说明。**
DIR_Attr	0x0B	1	**文件属性，取值为0x10表示为目录，0x20表示为文件**
DIR_NTRes	0x0C	1	保留
DIR_CrtTimeTenth	0x0D	1	保留(FAT32中用作创建时间，精确到10ms))
DIR_CrtTime	0x0E	2	保留(FAT32中用作创建时间，精确到2s)
DIR_CrtDate	0x10	2	保留(FAT32中用作创建日期)
DIR_LstAccDate	0x12	2	保留(FAT32中用作最近访问日期)
DIR_FstClusHI	0x14	2	保留(FAT32用作第一个簇的两个高字节)
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**

目录项的状态：若一个目录项0x00偏移处取值为**0x00**，则表示**目录项为空**，取值为**0xE5**则表示曾被使用但目前**已删除。**

2.4.2 文件分配表(FAT表)

FAT表是由FAT表项构成的，我们把FAT表项简称为FAT项。每个FAT项的大小有12位，16位，32位，三种情况，对应的分别FAT12，FAT16，FAT32文件系统。 本次实验中，我们需要实现FAT16文件系统，故**每个FAT项大小为16bit，即2字节。**每个FAT项都有一个固定的编号，这个编号是从0开始。

FAT表的前两个FAT项有专门的用途：0号FAT项通常用来存放分区所在的介质类型，例如硬盘的介质类型为“F8”，那么硬盘上分区FAT表第一个FAT项就是以"F8"开始，1号FAT项则用来存储文件系统的肮脏标志，表明文件系统被非法卸载或者磁盘表面存在错误。

分区的数据区每一个簇都会映射到FAT表中的唯一一个FAT项。因为0号FAT项与1号FAT项已经被系统占用，无法与数据区的簇形成映射，所以从2号FAT项开始跟数据区中的第一个簇映射，正因为如此，数据区中的第一个簇的编号为2，这也是没有0号簇与1号簇的原因。

分区格式化后，用户文件以簇为单位存放在数据区中，一个文件至少占用一个簇。当一个文件占用多个簇时，这些簇的簇号不一定是连续的，但这些簇号在存储该文件时就确定了顺序，即每一个文件都有其特定的“簇号链”。在分区上的每一个可用的簇在FAT中有且只有一个映射FAT项，通过在对应簇号的FAT项内填入“FAT项值”来表明数据区中的该簇是已占用，空闲或者是坏簇三种状态之一，FAT项具体取值及含义见下表：

簇取值	对应含义
0x0000	空闲簇
0x0001	保留簇
0x0002 - 0xFFEF	被占用的簇；指向下一个簇
0xFFF0 - 0xFFF6	保留值
0xFFF7	坏簇
0xFFF8 - 0xFFFF	文件最后一个簇，在本次实验中，为了方便起见，我们均使用**0xFFFF**作为最后一个簇的标志。

第三部分 实验任务

3.0 代码框架

3.0.1 文件说明

在编写自己的代码之前，建议同学们先大概阅读下已经给出的代码框架，了解代码的执行流程以及封装好的函数的功能及使用方法。本实验涉及的代码文件主要有：

- fat16.h：文件系统数据结构和函数的定义
- simple_fat16_part1.c：基础部分的代码，补充挖空的代码即可完成；
- simple_fat16_part2.c：进阶部分的代码，目前只有接口定义，会在下周发布含挖空和更多提示的代码，有能力的同学也可以尝试自主完成。
- fat16.img：FAT16格式镜像文件。

3.0.2 功能函数

为了减少代码实现中的工作量，框架中提供了一系列功能函数来辅助完成（simple_fat16_part1.c 16~618行）。我们在代码的注释中已经具体给出了每个函数的功能和输入输出含义，请各位同学认真阅读。

3.0.3 编译、运行

1. 在代码目录下执行 `make` （Makefile已经写好），并确认没有error（可能会存在warning，可以忽略）；
    - Makefile会将simple_fat16_part1.c和simple_fat16_part2.c编译成一个二进制文件。不完成TODO不影响成功编译。
2. 创建一个空文件夹 `fat_dir` 用于挂载镜像文件；
3. 然后使用命令 `./simple_fat16 -s -d fat_dir` 运行程序。我们提供的代码框架会打开名为全局变量 `FAT_FILE_NAME` 的镜像文件挂载。我们实验检查使用fat16.img，你也可以选择挂载其他的镜像。

> `-s` 参数代表 `single-thread` ，会强制FUSE使用单线程模式运行，一次只能有一个线程访问挂载的文件系统，这避免了数据争用带来的Bug，也简化了我们的调试。

3.1 任务一：实现FAT文件系统读操作

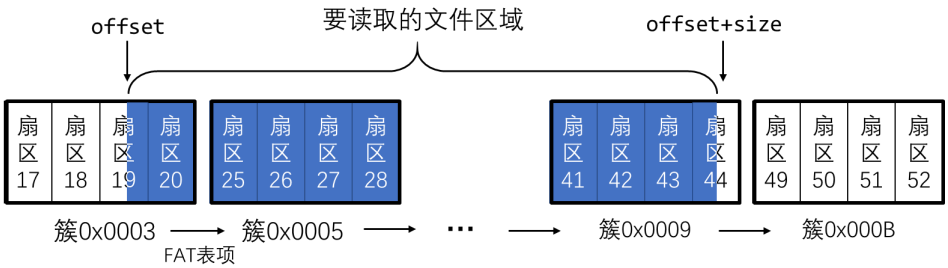
3.1.1 读目录

文件系统需要有获取文件目录结构的能力，从而保证 `ls` 和 `tree` 命令的实现。在我们的代码框架中，对应要实现的是 `int fat16_readdir(const char *path, void *buffer, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi)` ，该函数对于给定的路径path，首先找到对应目录的数据区，然后读出其中每条目录项的信息，通过传入的filler函数记录到buffer中，直到遇到0x00的结束项。
在读目录对应簇的时候，需要注意两点：

1. 一个扇区读完如果并没有读到0x00，那么就需要继续读下一个扇区。
2. 一个簇的扇区如果都读完了，需要找到下一个簇继续读。
3. 请跳过LFN（长文件名）项。该问题详见Part2.pdf的1.3。

3.1.2 读文件

当调用 `cat` / `head` / `tail` 等读文件的命令时，就会调用到读文件的接口，即 `int fat16_read(const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi)` 。读文件的实现逻辑与读目录基本相同，也需要注意跨扇区与跨簇的问题。



read()的读取范围

如图，读的起始地址和结束地址可能位于某个扇区的中间，即除了读覆盖的第一个扇区和最后一个扇区以外，中间的扇区都是全部读出来的，而第一个扇区只读出起始地址之后的内容，最后一个扇区只需要读结束地址之前的内容。所以，需要小心地计算要读取的第一个簇和扇区，以及最后一个簇和扇区，并正确处理他们的读取范围。

此外，读文件可能出现越界的问题，我们这里不做特殊要求，检测到越界返回0或者读到文件结束返回都可以。

3.1.3 实验要求

完成simple fat16中关于读文件的部分。

- **待补全部分**：完成simple_fat16.c文件中的TODO。**我们推荐在BEGIN和END间填充代码。**主要包括以下功能：
  1. `fat16_readdir` 遍历给定目录下的所有文件名，返回给fuse；
  2. `fat16_read` 从给定文件中读取一定长度的数据，返回给fuse。

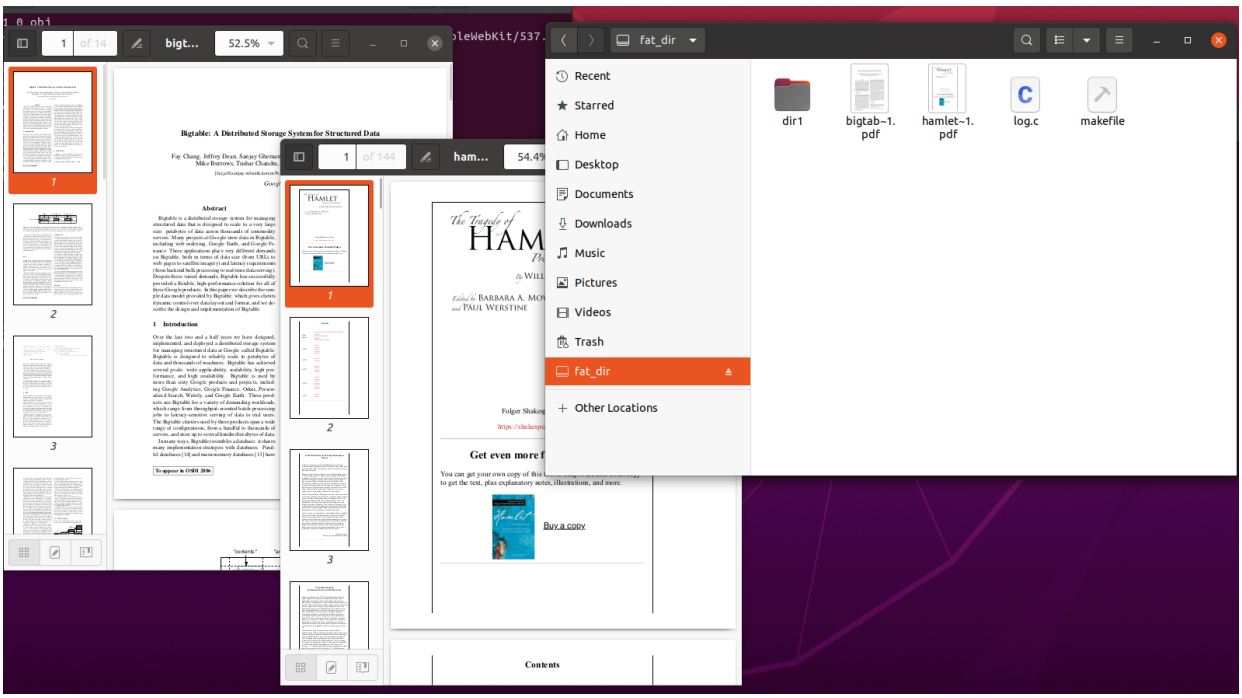
完成实验要求的代码能够从另一个shell中进入该目录，并执行tree和tail等命令得到正确的结果，如下图：

1. 能够运行tree/ls命令查看文件目录结构。输出示例：

```
blahaj@blahaj-VirtualBox:~/lab4-code-complete/fat_dir$ tree
.
├── bigtab~1.pdf
├── dir1
│   └── dir2
│       └── dir3
│           └── test.c
├── hamlet~1.pdf
├── log.c
└── makefile
```

2. 能够运行cat/head/tail命令查看目录下文件内容，或在文件管理器GUI下直接打开文件。
   - 我们在镜像中提供了若干文件。其中，一些文件大小较小，不存在跨簇/跨扇区的情况，一些文件较大，会出现跨簇/扇区的情况，大家可以据此分析代码错误所在。

挂载后用文件管理器GUI打开文件的示例：



3.2 任务二：实现FAT文件系统创建/删除文件操作

在实现了支持读操作的FAT 16文件系统基础上，实现创建文件和删除文件的功能。

3.2.1 创建文件

也就是需要保证文件系统支持 `touch` 命令，具体来讲，创建文件需要完成以下工作：

- 找到新建文件的父目录，有可能是根目录或者是子目录，根目录只有固定大小，子目录可以有多个簇；
- 在父目录中找到可用的entry，即未用过或用过但已删除的entry；
- 将新建文件的信息填入该entry，我们这里需要补充的是填入文件名、文件属性、首簇号以及文件大小。

因为创建文件会涉及到一些扩容相关的事情，但是扩容目前并不要求解决，会在进阶任务中进一步完成，这里解释两点可能会出现的疑问：

1. 创建文件会占用一个目录表项，FAT 16文件系统根目录大小是固定的，也就意味着最多只能有固定数量的表项可用。但是子目录和正常文件一样存放在数据区，这意味着子目录空间是可变的。如果子目录当前簇用完了，可以通过申请新的簇来增加子目录存放表项数量，这个扩容的操作在进阶任务中做进一步讨论，故完成本部分任务时，可以暂不考虑目录用满的解决方案。
2. 在创建文件时，可以传入 `0xffff` 作为新文件的首簇号，并将文件大小定为0。扩容同样在进阶任务中完成，此处不用担心文件后面写入的问题。

3.2.2 删除文件

也就是需要保证文件系统支持 `rm` 命令。删除一个文件，不需要改变文件具体的内容，而是在文件目录中更新对应的表项，文件系统在遍历时就会跳过被删掉的目录项，并在新文件创建时，允许覆盖掉对应的目录项。删除文件的思路大致分为两步：

- 释放该文件使用的簇（遍历簇，修改对应的FAT表项使其标记为空闲）
- 在父目录中释放该文件的entry（和创建文件类似）

代码流程上和创建文件相似，此处不做过多说明。

3.2.3 实验要求

同学们需要补全下面4个函数：

1. `fat16_mknod` 在给定path路径下创建新文件。

   > 在实现 `mknod` 时，你需要能正确识别出LFN项是非空的目录项，不将它们用于创建新文件、文件夹。（一般来说，这条不需要特别实现，因为LFN项的首个字节不会为 `0x00` 或 `0xe5` 等代表未使用或已删除条目字符）。LFN相关问题详见Part2.pdf的1.3。

2. `fat16_unlink` 删除给定path路径下对应的文件。
3. `dir_entry_create` 创建新目录表项。
4. `free_cluster` 释放簇号对应的簇，只需修改FAT对应表项，并返回下一个簇的簇号。

函数的主体框架已经给出，同学们需要实现每个函数中的TODO标记部分。**我们推荐在BEGIN和END间填充代码。**调试过程与任务一相同。

检查要求：需要能够创建文件；需要能够删除涉及跨簇的文件。

> 关于 `touch` 命令： `touch` 命令本身会检查文件是否存在，存在则不会创建文件。 `touch` 的基本作用是更新文件的修改时间，创建新文件其实是副作用，但我们目前只使用这个副作用。所以， `touch` 一个已存在的文件时不会触发 `fat16_mknod` 函数。

3.3 任务三：实现FAT文件系统创建/删除目录操作

> 本部分是进阶内容，挖过空的代码将于第二周发布。有能力的同学可以【先行完整地】自己完成两个函数。

实现 FAT16 文件夹创建/删除操作，主要包括如下两个函数的实现：

```
int fat16_mkdir(const char *path, mode_t mode);
int fat16_rmdir(const char *path);
```

这两个函数分别负责目录的创建及删除功能。FUSE库会将来自终端的 `mkdir` 命令转为函数 `fat16_mkdir` 的执行，将 `rmdir` 命令转为函数 `fat16_rmdir` 的执行。

3.3.1 创建文件夹

`fat16_mkdir` 的实现思路：

1. 找到新建文件夹的父目录；
2. 在父目录中找到可用的entry；
3. 将新建文件夹的信息填入该entry；

   这里的实现思路看起来与 `fat16_mknod` 完全一致（事实上，找到父目录以及在父目录中找到可用entry两部分的思路确实一致）。但是，我们在填入entry一步还需要其他操作。

4. 在FAT表中申请一个空闲簇，将其作为新建文件夹的初始簇，并在该簇中插入 `.` 和 `..` 两个目录项。

   这一操作是有其必要性的。第一，课程内容讲过目录文件内前两条entry分别是该文件本身和父目录文件的符号链接，它们在目录文件创建时就已经存在；第二，本次实验中不要求分配新簇，如果不在创建时就分配簇的话，会导致无法在该文件夹中创建文件，这是不可容忍的bug。

   但由于本次实验不涉及链接操作，所以不需要把 `.` 和 `..` 链接到对应目录，FUSE会在涉及这些目录时自动识别。填写它们的entry时，按照实际涉及的两个目录信息填写参数。

3.3.2 删除文件夹

`fat16_rmdir` 的实现思路：

1. 通过 `path` 获取到待删除目录对应的 `DIR_ENTRY` 所在位置；
2. 将 `DIR_ENTRY` 中 `DIR_Name[0]` 置为 `0xe5`，表示该目录已删除；
3. 返回 `0` 表示正常结束，否则表示异常。

注：`fat16_rmdir` 不需要删除待删目录下的子目录与文件。

3.4 任务四：实现FAT文件系统写操作

> 本部分是进阶内容，挖过空的代码将于第二周发布。有能力的同学可以【先行完整地】自己完成两个函数。

实现 FAT16 文件的写操作，主要包括如下两个函数的实现：

```
int fat16_write(const char *path, const char *data, size_t size, off_t offset, struct
fuse_file_info *fi);
int fat16_truncate(const char *path, off_t size);
```

函数解释说明：

- `fat16_write`：将 `data[0..size-1]` 的数据写入到 `path` 对应的，且偏移为 `offset` 的文件中；
- `fat16_truncate`：将 `path` 对应的文件大小设置为 `size`，如果 `size` 过大，初始化数据为 `0x00`。

`fat16_write` 函数的实现思路如下：

1. 通过 path 获取到对应的目录项，即 `DIR_ENTRY` 结构体，通过调用 `find_root` 即可；
2. 比较新写入的数据和文件原来的数据所需的簇数量，如果新写入的数据所需的簇数量大于原来所需的簇数量，则需要为文件扩容簇数量，具体流程可以在 FAT 表项中查找未使用的簇项，然后将其链接到原来文件的末尾；
3. 按需扩容后，只需要实现写文件操作即可，通过 `DIR.FstClusLo` 获取第一个簇号，之后可以通过链接依次遍历，直到达到 `offset` 对应的簇即可，写的流程可以参考3.1.2读流程中的说明（如写区间开头块和结束块的讨论）；
4. 实现目录项的更新，通过 `find_root` 可以获取到目录项对应的偏移量，更新对应的目录项数据写入即可；
5. 返回对应的写入数据字节数。

`fat16_truncate` 函数的实现思路如下：

1. 通过文件路径 path 获取到对应的目录项，即 `DIR_ENTRY` 结构体，通过调用 `find_root` 即可；
2. 比较新数据大小（size）所需簇数量 n1 和文件原来的数据所需的簇数量 n2，如果新写入的数据所需的簇数量大于原来所需的簇数量，则需要为文件扩容簇数量，具体流程可以在 FAT 表项中查找未使用的簇项，然后将其链接到原来文件的末尾：
    1. n1 = n2：此时不需要任何操作；
    2. n1 > n2：此时需要扩容文件大小，可以参照 `fat_write` 实现思路；
    3. n1 < n2：此时需要截断文件，遍历文件的簇号，直到第 n1 次，此时将对应的内容改为 `0XFFFF`，表示该簇是文件结尾。
3. 按需写入，比较新文件的大小 new_size（就是参数中的 `size`） 和原来的文件大小old_size（`Dir.DIR_FileSize`）：
    1. new_size > old_size：需要写入 `0x00` 到末尾的 `new_size - old_size` 个字节处；
    2. new_size <= old_size：不需要进行任何操作；
4. 实现目录项的更新，通过 `find_root` 可以获取到目录项对应的偏移量，更新对应的目录项数据写入即可；
5. 返回 0 表示正常结束，否则表示异常。

第四部分 检查内容 & 评分标准

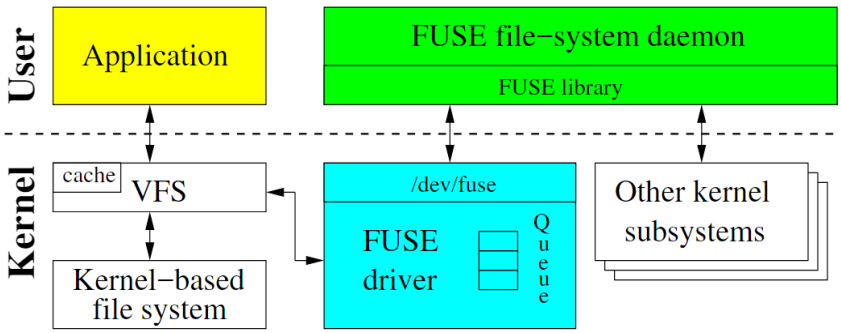
1. 任务一：实现FAT文件系统读操作（**满分2分**）
    - 能够运行 `tree` ， `ls` 命令查看文件目录结构 （**1分**）
    - 能够用过 `cat` / `head` / `tail` 命令，或直接打开文件（如直接打开pdf）查看目录下文件内容（**1分**）
2. 任务二：实现FAT文件系统创建/删除文件操作（**满分2分**）
    - 能够运行 `touch` 命令创建新文件，且保证文件属性的正确填写（**1分**）
    - 能够运行 `rm` 命令删除已有文件，且保证簇的正确释放（**1分**）
3. 任务三：实现FAT文件系统创建/删除文件夹操作（**满分2分**）
    - 能够运行 `mkdir` 命令创建新目录，且新目录中默认具有 `.` 和 `..` 两项（**1分**）
    - 能够运行 `rm -r` 命令删除已存在的目录（**1分**）
4. 任务四：实现FAT文件系统写操作（**满分2分**）
    - 能够使用 `echo` 命令和重定向符 `>` ， `>>` 正确进行文件写入（**2分**）
5. 知识问答（**满分2分**）
    - 回答从知识问答一节中随机抽取的三道问题，每回答正确一题得一分。回答对两题即可得到满分（**2分**）

知识问答

会同进阶部分的文档一并发布。

附录

1. FUSE简介



- FUSE主要由三部分组成：FUSE内核模块、用户空间库libfuse以及挂载工具fusermount：
    1. fuse内核模块：实现了和VFS的对接，实现了一个能被用户空间进程打开的设备。
    2. fuse库libfuse：负责和内核空间通信，接收来自/dev/fuse的请求，并将其转化为一系列的函数调用，将结果写回到/dev/fuse；提供的函数可以对fuse文件系统进行挂载卸载、从linux内核读取请求以及发送响应到内核。
    3. 挂载工具：实现对用户态文件系统的挂载。
- 更多详细内容可参考 **这个文章**。

2. FAT 16 文件系统中的时间

```
struct tm{
    int tm_sec; //代表目前秒数，正常范围为0-59，但允许至61秒
    int tm_min; //代表目前分数，范围0-59
    int tm_hour;  //从午夜算起的时数，范围为0-23
    int tm_mday;  //目前月份的日数，范围01-31
    int tm_mon;  //代表目前月份，从一月算起，范围从0-11
    int tm_year;  //从1900 年算起至今的年数
    int tm_wday;  //一星期的日数，从星期一算起，范围为0-6
    int tm_yday;  //从今年1月1日算起至今的天数，范围为0-365
    int tm_isdst;  //日光节约时间的旗标，也就是夏令时
};

// 用timer的值来填充tm结构。timer的值被分解为tm结构，并用本地时区表示。
struct tm *localtime(const time_t *timer)

// 偏移地址0x16：长度2，表示时间=小时*2048+分钟*32+秒/2。
// 也就是：0x16地址的0~4位是以2秒为单位的量值，
//         0x16地址的5~7位和0x17地址的0~2位是分钟，
//         0x17地址的3~7位是小时。

// 偏移地址0x18：长度为2，表示日期=(年份-1980)*512+月份*32+日。
// 也就是：0x18地址的0~4位是日期数，
//         0x18地址5~7位和0x19地址的0位是月份，
//         0x19地址的1~7位为年号。
```

3. 引导扇区的完整结构

这里列出了引导扇区的完整结构，但本实验只需要关注加粗的几个字段。

名称	偏移(字节)	长度(字节)	说明
BS_jmpBoot	0x00	3	跳转指令（跳过开头一段区域）
BS_OEMName	0x03	8	OEM名称，Windows操作系统没有针对这个字段做特殊的逻辑，理论上说这个字段只是一个标识字符串，但是有一些FAT驱动程序可能依赖这个字段的指定值。常见值是"MSWIN4.1"和"FrLdr1.0"
BPB_BytsPerSec	0x0b	2	**每个扇区的字节数。基本输入输出系统参数块从这里开始。**
BPB_SecPerClus	0x0d	1	**每簇扇区数**
BPB_RsvdSecCnt	0x0e	2	**保留扇区数（包括主引导区）**
BPB_NumFATS	0x10	1	**文件分配表数目，FAT16文件系统中为0x02,FAT2作为FAT1的冗余**
BPB_RootEntCnt	0x11	2	**最大根目录条目个数。一定是16的倍数，因为必须保证根目录区域对齐完整扇区。**
BPB_TotSec16	0x13	2	总扇区数（如果是0，就使用偏移0x20处的4字节值）
BPB_Media	0x15	1	介质描述：F8表示为硬盘，F0表示为软盘
BPB_FATSz16	0x16	2	**每个文件分配表的扇区数（FAT16专用）**
BPB_SecPerTrk	0x18	2	每磁道的扇区数
BPB_NumHeads	0x1a	2	磁头数
BPB_HiddSec	0x1c	4	隐藏扇区数
BPB_TotSec32	0x20	4	总扇区数（如果0x13处不为0，则该处应为0）
BS_DrvNum	0x24	1	物理驱动器号，指定系统从哪里引导。
BS_Reserved1	0x25	1	保留字段。这个字段设计时被用来指定引导扇区所在的
BS_BootSig	0x26	1	扩展引导标记（Extended Boot Signature）
BS_VollID	0x27	4	卷序列号，例如0
BS_VollLab	0x2B	11	卷标，例如"NO NAME "
BS_FilSysType	0x36	8	文件系统类型，例如"FAT16"
Reserved2	0x3E	448	引导程序
Signature_word	0x01FE	2	引导区结束标记

参考资料

- **FAT文件系统实现**
- **文件分配表**
- **fat32文件系统示例**