实验四：

FAT文件系统的实现

1 实验目标

- 熟悉FAT16的存储结构，利用FUSE实现一个FAT文件系统

2 实验环境

- OS：Ubuntu 18.04

  注意：本次实验不需要在 Linux 0.11 环境下编程，因平台技术限制，本实验无法在vlab平台完成

3 实验说明

- **本次实验可选做，由两个独立任务组成，完成任务能获得课程成绩额外加分（两个任务获取的分数可叠加）**
  - **任务一：**利用FUSE实现一个功能完善的FAT 16文件系统
  - **任务二：**根据自己对文件系统的理解，设计一个关于文件系统的实验，包括*设计文档+实验的原型代码实现*
- **实验提交要求和截止时间见文末**
- 本次答疑的助教为：**王千里 王霄阳 来逸瑞**，助教答疑时会给出完成实验相关的线索提示，但**不会提供代码编写和调试相关细节的帮助**，请同学们独立完成代码实现。
- **如何提一个好的问题? & 如何Debug?**

  为了更快的解决大家的问题, 麻烦大家按照这个格式提问.

  - 问题的简单描述

    这里的问题请简明扼要, 并且避免模糊不清的字样(如代码无法执行). 一个好的问题描述如下:

    问题: 我的问题是XXXX(如编译报错), **报的错误是: XXXX(截图报错信息)**, 以及大家在提问之前, 请**先在搜索引擎上搜索报错信息**, 我们不可能做到实时回复, 但是搜索引擎可以. 请**避免一些比较简单的提问**(如函数参数格式不正确, 使用了未定义的变量等).
  - 出问题的代码(具体到行)

    如果出现了**无法自己解决的**问题, 我们要求提供**精确到行**的代码信息. 如:

    错误的代码在xxx文件的第xxx行(通过**截图标注**的形式), 这行代码运行引发了XXX错误(如返回数据不正确, 引发了段错误等).

    那么如何找到错误的行呢? 我们在代码中提供了调试区, 可以在这里调用已经编写好的函数, 具体运行方法可以参看4.3.1(2), 也可以参考bench.h中的对应函数查看如何调用. 关于调试, 我们有两个工具

    - Print方法(推荐)

      如果程序可以执行, 我们首先通过printf函数, 观察输入到函数内的变量是否正确. 接着, 我们依次打印中间的变量, 看是具体在哪一个变量附近出了问题, 进而找到对应的错误行.
    - GDB方法

如果程序不能执行(如遇到了段错误), 我们可以通过GDB程序对程序进行调试. GDB的基本使用方法实验一中已经说明, 这里不再赘述. 需要注意的是, 我们必须要在进入gdb调试之后输入`set args --debug`, 才可以执行到debug区的语句, test区的语句同理.

4 任务一实验内容

4.1 熟悉FUSE的使用

4.1.1 FUSE概述

- 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提供的接口实现具体的文件操作就可以实现一个文件系统。
- 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博客。

4.1.2 配置FUSE环境

- linux kernel在2.6.14后添加了FUSE模块，因此对于目前的大多数发行版来说只需要安装libfuse库即可。
- 在https://github.com/libfuse/libfuse里下载libfuse 2.9.5，然后编译安装：

```
$ 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
```

- **编译出错处理：**在执行./makeconf.sh可能会遇到“ libtoolize: not found ”或 “autoreconf: not found”，需要安装libtool和autoconf软件包，ubuntu下可以通过下面的命令安装：

```
sudo apt install libtool
sudo apt install autoconf
```

4.1.3 测试FUSE

- 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目录，可以看到当前Linux系统的“/”被挂载到这个目录下。

- 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，如果出现下图的错误（"fuse: bad mount point"），可通过执行上面这条命令卸载对应的文件夹来解决。

```
emulation@emulation-VirtualBox:~/os-lab4-fs/lab4-code$ ./simple_fat16 -d fat_dir
/
fuse: bad mount point `fat_dir/': Transport endpoint is not connected
emulation@emulation-VirtualBox:~/os-lab4-fs/lab4-code$
```

4.2 熟悉FAT文件系统

4.2.1 FAT格式磁盘镜像的制作过程

- 分为三步：创建文件，格式化文件，挂载使用

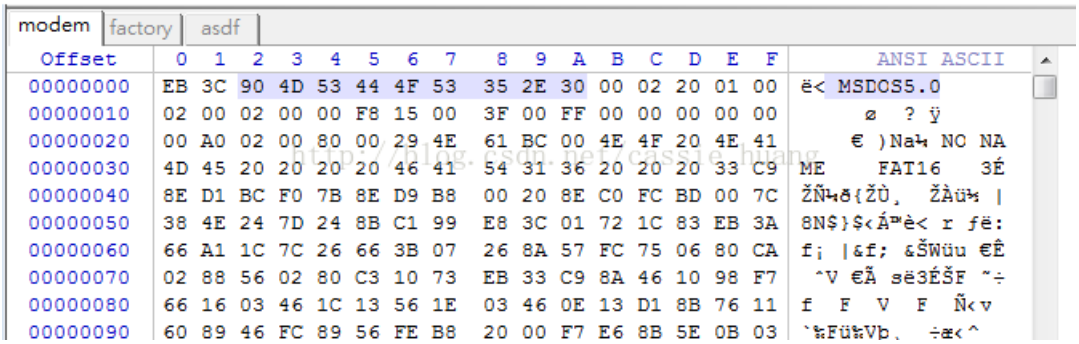
```
$ dd if=/dev/zero of=fat-disk.img bs=1M count=100

$ mkfs.vfat -F 16 fat-disk.img

$ mkdir mdir
$ mount fat-disk.img mdir
```

4.2.2 FAT16的存储结构

- FAT16文件系统的基本结构依次为：DBR扇区、FAT表1、FAT表2、根目录和数据区，FAT16格式的磁盘的组织方式如下图所示：

DBR扇区	保留扇区	FAT 1	FAT 2 （重复的）	根目录	数据区	剩余扇区
占1扇区	可能存在	大小取决于实际情况	与FAT 1相同	一般占32个扇区	2号簇开始编号	不足1簇

- **DBR扇区:** DBR是操作系统可以直接访问的第一个扇区, 包括一个引导程序和一个称为**BPB的本分区参数记录表**。BPB参数块记录着本分区的启始扇区、结束扇区、文件存储格式、硬盘介质描述符、根目录大小、*FAT*个数、分配单元的大小等重要参数。下图是一个FAT16文件系统的DBR扇区：



DBR扇区每个字段代表的含义如下：

偏移地址	长度（字节）	含义	数值
0h	3	跳转命令	EB 3C 90
3h	8	OEM NAME	4D 53 44 4F 53 35 2E 30 = MSDOS5.0
Bh	2	每扇区字节数	00 02 = 0x200 = 512bytes
Dh	1	每簇扇区数	20 = 0x20 = 32 扇区
Eh	2	保留扇区数	01 00 = 0x1=1 扇区
10h	1	FAT数量	02 = 0x2 FAT
11h	2	根目录项数	00 02 =0x200=512
16h	2	每个FAT所占扇区数	15 00 = 0x15 = 21个扇区
18h	2	每个道所占扇区数	3F 00 = 0x3F = 63
1Ch	4	隐藏扇区数	0
36h	8	FAT NAME	FAT16
3Eh	448	引导程序执行代码	…
1FEh	2	扇区结束标志	AA 55

而BPB参数记录表中的对应表述如下(这里只摘取了一些BPB的对应参数, 具体可以看这里)

名称	Offset (byte)	大小 (byte)	描述
BS_jmpBoot	0	3	跳转指令。指向启动代码，允许以下两种形式： **jmpBoot[0] = 0xEB, jmpBoot[1] = 0x??, jmpBoot[2] = 0x90** 和 **jmpBoot[0] = 0xE9, jmpBoot[1] = 0x??, jmpBoot[2] = 0x??** 0x??表示该字节可以为任意 8-bit 值，这是 Intel x86 架构 3 字节的无条件转移指令，跳转到操作系统的启动代码，这些启动代码往往紧接 BPB 后面 0 扇区里的剩余字节，当然也可能位于其他扇区。以上的两种形式任取。**jmpBoot[0] = 0xEB** 是较常用的一种格式
BS_OEMName	3	8	建议值为“MSWIN4.1” 此域经常引起人们的误解，其实这只是一个字符串而已，Microsoft 的操作系统似乎并不关心此域。但其他厂商的 FAT 驱动程序可能会检测此项，这就是为什么建议将此域设为 “MSWIN4.1”的原因，这样可以尽量避免兼容性的问题。你可以更改它的内容，但这有可能造成某些 FAT 驱动程序无法识别该磁盘。很多情况下该域用于显示格式化该 FAT 卷的操作系统的名称。
BPB_BytsPerSec	11	2	每扇区字节数，取值只能是以下的几种情况：512、1024、2048 或是 4096，设置为 512 将取得最好的兼容性，目前有很多的 FAT 代码都是硬性的规定每扇区字节数为 512，而不是实际检测该此域的值，Microsoft 的操作系统能够很好地支持 1024、2048 和 4096 各种数值。 **NOTE:** 请勿曲解此处“最好的兼容性”的意思，如果某些存储介质的物体特性决定其值为 N,那么你就必须使用该数值 N,该 N 值一定是小于或是等于 4096。那么取得“最好的兼容性”的办法就是使用该特定的 N 值。
BPB_SecPerClus	13	1	每簇扇区数，其值必须是 2 的整数次方（该整数必须>=0），如 1、2、4、8、16、32、64 或 128，同时还必须保证每簇的字节数不超过 32K，既：保证（BPB_BytsPerSec * BPB_SecPerClus <= 32K (1024*32) 该值大于 32K 是绝对不允许的，虽然有些版本的操作系统支持每簇字节数最大到 64K，但很多应用程序的安装程序都无法在这样的 FAT 文件系统上正常运行。
BPB_RsvdSecCnt	14	2	保留区中保留扇区的数目，保留扇区从 FAT 卷的第一个扇区开始，此域不能为 0，对于 FAT12 和 FAT16 必须为 1，FAT32 的典型取值为 32,目前很多 FAT 程序都是硬性规定 FAT12/FAT16 的保留扇区数为 1，而不对此域进行实际的检测，Microsoft 的操作系统支持任何非零的值.
BPB_NumFATs	16	1	此卷中 FAT 表的份数。任何 FAT 格式此域都建议为 2 。虽然此域取值为其他>=1 的数值也是合法的，但是很多 FAT 程序和部分操作系统对于此项不为 2 的时候将无法正常工

			作。当不为 2 时，Microsoft 的操作系统仍能良好的工作。但仍然**强烈建议**此域为 2 。 选择此项的标准值为 2 的原因是为了提供一份 FAT 表的备份，当其中一个 FAT 表所在的扇区被损坏时我们可以从备份的 FAT 表中读取正确的数据。但是对于一些非磁盘介质的存储器（如 FLASH 卡）这一特性变得毫无用处，如果想

			使用 1 个 FAT 表来节省空间，那么带来的问题将是某些操作系统无法识别该 FAT 卷。
BPB_RootEntCnt	17	2	对于 FAT12 和 FAT16 此域包含根目录中的目录项数（每个项长度为 32 bytes），对于 FAT32,此项必须为 0。对于 FAT12 和 FAT16,此数值乘以 32 必须为 BPB_BytsPerSec 的偶数倍，为了达到最好的兼容性，FAT12/FAT16 应该取值为 512.
BPB_TotSec16	19	2	早期版本中 16-bit 的总扇区数，这里的总扇区数包括 FAT 卷上四个基本区的全部扇区，此域可以为 0，若此域为 0，那么 BPB_TotSec32 必须非 0，对于 FAT32,此域必须为 0。对于 FAT12/FAT16，此域填写总扇区数， 如果该数值小于 0x10000 的话，BPB_TotSec32 必须为 0。
BPB_Media	21	1	对于“固定”（不可移动）存储介质而言，0xF8 是标准值，对于可移动存储介质，经常使用的数值是 0xF0,此域合法的取值可以取 0xF0, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE 和 0xFF。另外要提醒的一点是，无论此域写入什么数值，同时也必须在 FAT[0]的低字节写入相同的值，这是因为早期的 MSDOS 1.x 使用该字节来判定是何种存储介质。
BPB_FATSz16	22	2	FAT12/FAT16 一个 FAT 表所占的扇区数，对于 FAT32 此域必须为零，在 BPB_FATSz32 中有指定其 FAT 表的大小
BPB_SecPerTrk	24	2	每磁道扇区数，用于 BIOS 中断 0x13，此域只对于有“特殊形状”（由磁头和柱面分割为若干磁道）的存储介质有效，同时必须可以调用 BIOS 的 0x13 中断得到此数值
BPB_NumHeads	26	2	磁头数，用于 BIOS 的 0x13 中断， 类似于上面的 BPB_SecPerTrk,只有对特殊的介质才有效,此域包含一个至少为 1 的数值，比如 1.4M 的软盘此域为 2
BPB_HiddSec	28	4	在此 FAT 分区之前所隐藏的扇区数，必须使得调用 BIOS 的 0x13 中断可以得到此数值，对于那些没有分区的存储介质，此域必须为 0，具体使用什么值由操作系统决定。
BPB_TotSec32	32	4	该卷总扇区数（32-bit），这里的总扇区数包括 FAT 卷上四个基本区的全部扇区， 此域可以为 0,若此域为 0，BPB_TotSec16 必须为非 0，对于 FAT32,此域一定是非 0。对于 FAT12/FAT16,如果总扇区数大于或等于 0x10000 的话，此域就是总扇区数，同时 BPB_TotSec16 的值为 0

根据上面的DBR扇区，我们可以算出各FAT的偏移地址，根目录的偏移地址，数据区的偏移地址。

FAT1偏移地址：保留扇区（**FAT1之前的扇区，包括DBR扇区**）之后就是FAT1。因此可以得到，FAT1的偏移地址就是1个扇区的位置，也就是512。

FAT2偏移地址：FA1偏移地址+FAT1的大小，512+21*512 = 11264。

根目录偏移地址： FAT2偏移地址+FAT2的大小，11264+21*512= 22016。

数据区的偏移地址：根目录偏移地址+根目录大小，22016+32*512=38400。其中根目录大小是由根目录项数决定的，每项占32字节。

- **FAT表：**FAT是簇的链表，FAT2与FAT1的内容通常是即时同步的，可以认为两个FAT表完全相同。在根据目录项获取文件的首簇号后，在FAT中找到对应的簇，可以找到下一个簇，一直到文件结束。每个簇用2字节表示簇的状态，具体意义如下表所示：

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

- **根目录：**FAT文件系统的一个重要思想是把目录当作一个特殊文件来处理，在FAT16中，虽然根目录地位并不等同于普通的目录，但其组织形式和普通的目录并没有不同。不管目录文件所占空间为多少簇，系统都会以32个字节为单位进行目录文件所占簇的分配。每个32字节目录项表示的具体含义如下：

偏移地址	长度	含义
0	8	文件名
8	3	扩展名
B	1	属性
C	10	保留
16	2	文件最近修改时间
18	2	文件最近修改日期
1A	2	文件的首簇号
1C	4	文件长度大小

  系统将文件名分成两部分进行存储，即主文件名+扩展名。不记录主文件名与扩展名之间的"."，并且存储成**大写字母**。主文件名不足8个字符以空白符(20H)填充，扩展名不足3个字符同样以空白符(20H)填充。0x0偏移处的取值若为0x0，表明目录项为空；若为0xE5，表明目录项曾被使用，但对应的文件或文件夹已被删除。文件名中的第一个字符若为“.”或“..”表示记录的是一个子目录的目录项，“.”代表当前目录；“..”代表上级目录。

- **FAT表与文件定位：**FAT16文件系统从根目录所占的32个扇区之后的第一个扇区开始以簇为单位进行数据的处理，这之前仍以扇区为单位。对于根目录之后的第一个簇，系统并不编号为0号簇或1号簇，而是编号为2号簇，也就是说数据区顺序上的第1个簇也是编号上的2号簇。

  FAT表以"F8 FF FF FF" 开头，为介质描述单元，并不参与FAT表簇链关系，下图展示了FAT表的内容。为了定位一个文件或目录，需要先找到其对应的目录项，然后根据首簇号查找FAT中的相应表项。

modem | factory | asdf

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ANSI ASCII
00011264	F8	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	øÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00011280	09	00	0A	00	0B	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿ
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 H
00011408	FF	FF	4A	00	4B	00	FF	FF	4D	00	4E	00	FF	FF	50	00	ÿÿJ K ÿÿM N ÿÿP
00011424	51	00	FF	FF	53	00	54	00	FF	FF	56	00	57	00	FF	FF	Q ÿÿS T ÿÿV W ÿÿ
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 ÿÿh

  假设一个文件，其首簇号是2，则查看FAT的相对偏移0x4~0x5的内容，此处为0xFFFF，表示存储在2号簇上的文件是个小文件，只占用1个簇便结束了。

  假设另一个文件的首簇号是8，则查看FAT的相对偏移0x10-0x11的内容，为0x0009，就是说该文件继8号簇后的内容会存放在9号簇中；继续查看9号簇的FAT表项内容是0x000A，该文件继9簇后的内容会存放在10号簇中，继续这个过程，会发现文件在11号簇结束。

4.3 实验要求

4.3.1 支持读操作的FAT16文件系统 (占此任务总分的40%)

(1) 具体要求

- 下载代码：

```
$ wget https://github.com/ZacharyLiu-CS/USTC_OS/raw/master/Lab4-File-System/lab4-code.tar.gz
$ tar zxf 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。
- **提示：**有需要的话，可采用Linux下的xxd和hexdump等命令或者Windows下的WinHex等十六进制文件编辑工具，分析对应的磁盘镜像文件。
- **在实验过程中，不允许修改目录下bench.h中的任何内容，一经发现，本次实验按作弊处理**

（2）运行与测试

- 代码调试

  请在main函数中的指定位置编写调试相关代码(如对上面的补充好了的函数进行调用，并观察其返回的结果符不符合你的预期). 可以使用

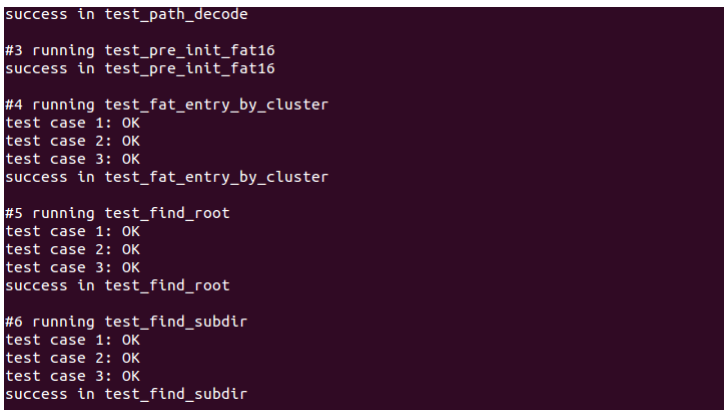
```
#进入源码目录
make clean
make
./simple_fat16 --debug
```

  执行调试部分的代码.

- 使用如下的命令编译并测试程序：

```
#进入源码目录
make clean
make
#测试一
./simple_fat16 --test
```

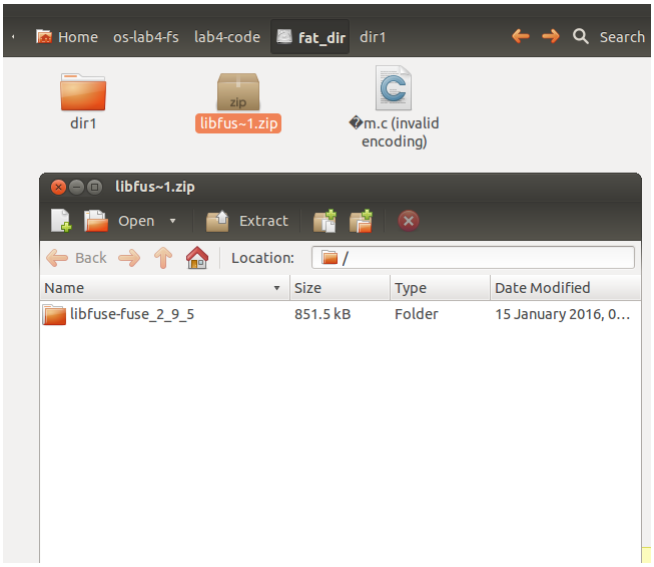
  测试一是为了验证程序的FAT相关部分的代码正确性（以fat16_test.img作为磁盘镜像文件），一共有6个测试目标：



  通过测试一后，运行如下的命令进行FUSE功能的测试（以fat16.img作为磁盘镜像文件）：

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

  这时候在文件管理器中打开fat_dir目录，需要能够看到里面的文件列表，以及正常访问里面的文件：



(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
- 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。

4.3.2 支持完整读写功能的FAT16文件系统 (占此任务总分的60%)

(1) 支持长文件名的FAT 16文件系统

- 在上一部分的simple_fat16.c代码中，我们只关注了短文件名（主文件名+扩展名的8+3格式）的实现，这种方式有很大的局限性：限制了文件名的长度，以及不支持大小写混合的命名方式（短文件名的目录项的文件名需要全大写）。
- **要求：**参考相关资料，实现能够支持长文件名的FAT文件系统，**要求能够正常显示fat.img中的文件名**(通过文件管理器打开文件夹时，文件名没有波浪号)。

(2) 支持写操作的FAT 16文件系统

- 在已有的支持只读操作基础上，实现可支持创建文件，写文件，删除文件的功能，同时能够支持文件属性的修改。
- **要求：能够在文件系统中支持touch，cp，rm命令**

(3) 实现线程安全的FAT 16文件系统

- **要求：**在多线程访问文件系统时，需要保证读写请求被正确处理，**能够正常使用fio工具测试FAT 16文件系统的文件**
  - 能够通过多线程读写混合的fio测试（下面的命令对fat_dir/fio.img文件进行多线程随机读写测试）：

```
$ fio -filename=fat_dir/fio.img -direct=1 -iodepth 1 -thread -rw=randrw
-ioengine=psync -bs=4k -size=50m -numjobs=4 -runtime=60 -
group_reporting -name=mytest
```

4.3.3 评分标准

- 该任务由4.3.1和4.3.2两部分组成，各占40%和60%的分数，具体得分由两部分构成：代码运行得分，文档得分：
  - 代码运行得分：通过4.3.1的**测试一和测试二**，以及4.3.2的**三个点中提到的要求。**
  - 文档得分是指：以及在实验报告中**简要描述自己实现的代码**，并且**回答4.3.1中提出的问题。**

5 任务二实验内容

5.1 实验要求

- 根据自己对文件系统的理解，设计一个关于文件系统的实验（如可以结合Linux 0.11内核设计相关的实验）
- 评分标准：主要由***设计文档***和***实验的原型代码实现***两部分组成
    - 设计文档：详细阐述自己为了设计实验所调研的文件系统相关内容，以及实验的目的，和具体的实验方案（如该修改哪些代码，增加哪些功能等内容）
    - 原型代码：根据自己的能力，实现相应的实验方案

6 实验提交

本次实验要求提交**实验录屏、实验报告和代码**

（1）录屏要求

- 任务一：需要保证展示以下内容（缺少任意一项则视为该部分实验未完全做完，酌情给分）
    - 内容1（**“4.3.1 支持读操作的FAT16文件系统”**）
        - 一边**展示修改部分**一边**口述代码编写思路、代码做了些什么**
        - 展示在代码目录下执行`make clean;make`的结果
        - 展示测试一与测试二的结果
        - 展示在代码目录下执行`md5sum bench.h -c bench`的结果
    - 内容2 （**“4.3.2 支持完整读写功能的FAT16文件系统”**）
        - 一边**展示修改部分**一边**口述代码编写思路、代码做了些什么**
        - 通过文件管理器打开fat.img所挂载的文件夹，查看里面的文件名是否显示正常
        - 在挂载的文件系统中执行touch，cp，rm命令
        - 在挂载的文件系统中进行多线程读写混合的fio测试
- 任务二：如果有自己的原型代码实现，简要说明自己的实验内容，并展示相应的结果

（2）文档要求

- 任务一：需要包含以下内容
    - 实验修改部分截图, 以及代码说明
    - 实验运行结果截图
    - 问题回答
- 任务二：阐述自己设计的实验方案

（3）提交要求

- 按下面描述的方式组织相关文件（具体的实验报告和录屏的要求见实验内容部分）

```
- 顶层目录（可自行命名，如EXP4）
  - task1
    - simple_fat16.c
    - 实验录屏(10分钟之内)
    - 实验报告(学号+姓名+实验四命名，提交PDF版本)
  - task2
    - 设计文档(学号+姓名+实验四命名，提交PDF版本)
    - 原型代码
    - 实验录屏(10分钟之内)
```

- 将上述文件压缩
  - 格式为 .7z/.rar/.zip
  - 命名格式为 学号_姓名_实验4，如果上传后需要修改，由于ftp服务器关闭了覆盖写入功能，需要将文件重新命名为学号_姓名_实验4_修改n (n为修改版本)，以最后修改版本为准。
    - 如PB10001000_张三_实验4.zip ，PB10001000_张三_实验4_修改3.zip

3、提交到ftp服务器

- 服务器地址：ftp://OS2020:OperatingSystem2020@nas.colins110.cn:2001/
- 上传至文件夹: 第四次实验
- 实验截止日期：2020-06-xx 23:59
- 说明：请尽早上传，截止时间前十二小时内的提交错误概不处理。