FAT32 挑战性任务

任务说明

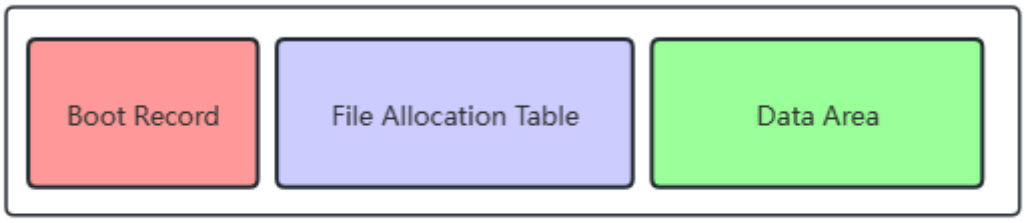
本次挑战性任务的目标是在用户态进程中实现 FAT32 文件系统，并支持 Lab6 的正常运行。具体而言，同学们需要移除 mos 之前的用户态文件系统，并按照给定的接口开发新的文件系统。完成开发后，还需要通过挑战性任务截止日期后的答辩和检查以获得分数。详细的任务要求将在后续说明中提供。

FAT 演化

`File Allocation Table`(FAT)文件系统是一个经典的文件系统，曾被微软广泛应用于 DOS 系列和 Windows 操作系统，其简单的结构使其成为了许多存储设备所采用的文件系统，并且受到了大部分系统的支持。现有的 FAT 文件系统有`FAT12`,`FAT16`,`FAT32`,`exFAT`等。这些文件系统主要区别在于支持的容量和文件大小不同。

FAT32 组成结构

FAT32 文件系统主要由以下几个部分组成：



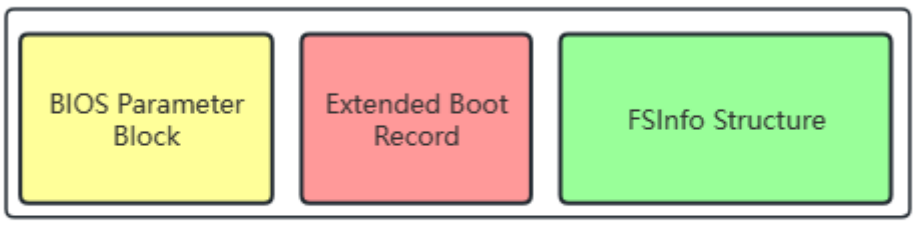
Fat FS Organization

- **Boot Sector**：引导扇区，用于存储文件系统的基本信息，如文件系统类型、文件系统版本、磁盘容量等，只占用一个扇区。
- **FAT**：文件分配表，用于记录文件数据簇的分配情况，每个表项对应一个簇，表项的值表示下一个簇的编号，最后一个簇的表项值为 0xFFFFFFF8(0x0FFFFFFF)。
- **Data Area**： 数据区，用于存储文件数据，由多个簇组成，每个簇的大小由文件系统的参数决定。

> 注：在FAT32中，所有数据端序都为**小端**。

Boot Sector

在 FAT32 文件系统中，Boot Sector 的结构如下：



Fat32 Boot Record

- **BPB**：BIOS Parameter Block，BIOS 参数块，用于存储文件系统的基本信息，如扇区大小、簇大小、FAT 表大小等。

表明media 描述符种类字段，不需考虑

偏移	大小	描述
0x00	3	x86 短跳转指令，不需考虑。
0x03	8	OEM 标识符，不需考虑。
0x0B	2	每扇区字节数。
0x0D	1	每簇扇区数。
0x0E	2	保留扇区数。引导扇区的数量。
0x10	1	FAT 表数目。通常为 2。
0x11	2	根目录项数。FAT32 中为 0。
0x13	2	总扇区数。如果该值为 0，则使用 *Large Sector Count* 字段。
0x15	1	
0x16	2	每 FAT 表扇区数,仅限FAT12/16。
0x18	2	每磁道扇区数。
0x1A	2	磁头数。
0x1C	4	隐藏扇区数。引导扇区之前的扇区数。
0x20	4	大扇区数。如果 *Total Sector Count* 字段为 0，则使用该字段。

- **FAT32 Extended**：FAT32 扩展信息，用于存储 FAT32 文件系统的扩展信息，如 FAT 表大小、根目录簇号等。

偏移	大小	描述
0x24	4	每FAT 表占据扇区数。
0x28	2	扩展标志。
0x2A	2	FAT 版本号。
0x2C	4	根目录簇号。
0x30	2	文件系统信息扇区号。
0x32	2	备份引导扇区号。
0x34	12	保留字段。
0x40	1	驱动器号。
0x41	1	保留字段。
0x42	1	引导标志。

偏移	大小	描述
0x43	4	卷序列号。
0x47	11	卷标。
0x52	8	文件系统类型标签。
0x5A	420	启动代码，这里不需考虑
0x1FE	2	0xAA55,表示引导分区标志

- **FsInfo**：文件系统信息，用于存储文件系统的信息，如空闲簇数、下一个可用簇号等。

偏移	大小	描述
0x00	4	0x41615252,标志。
0x04	480	保留
0x1E4	4	0x61417272，标志。
0x1E8	4	当前空闲簇号。
0x1EC	4	用于搜寻可用簇的开始簇号,为0xFFFFFFFF则需从2开始
0x1F0	12	保留。
0x1FC	4	0xAA55000标志。

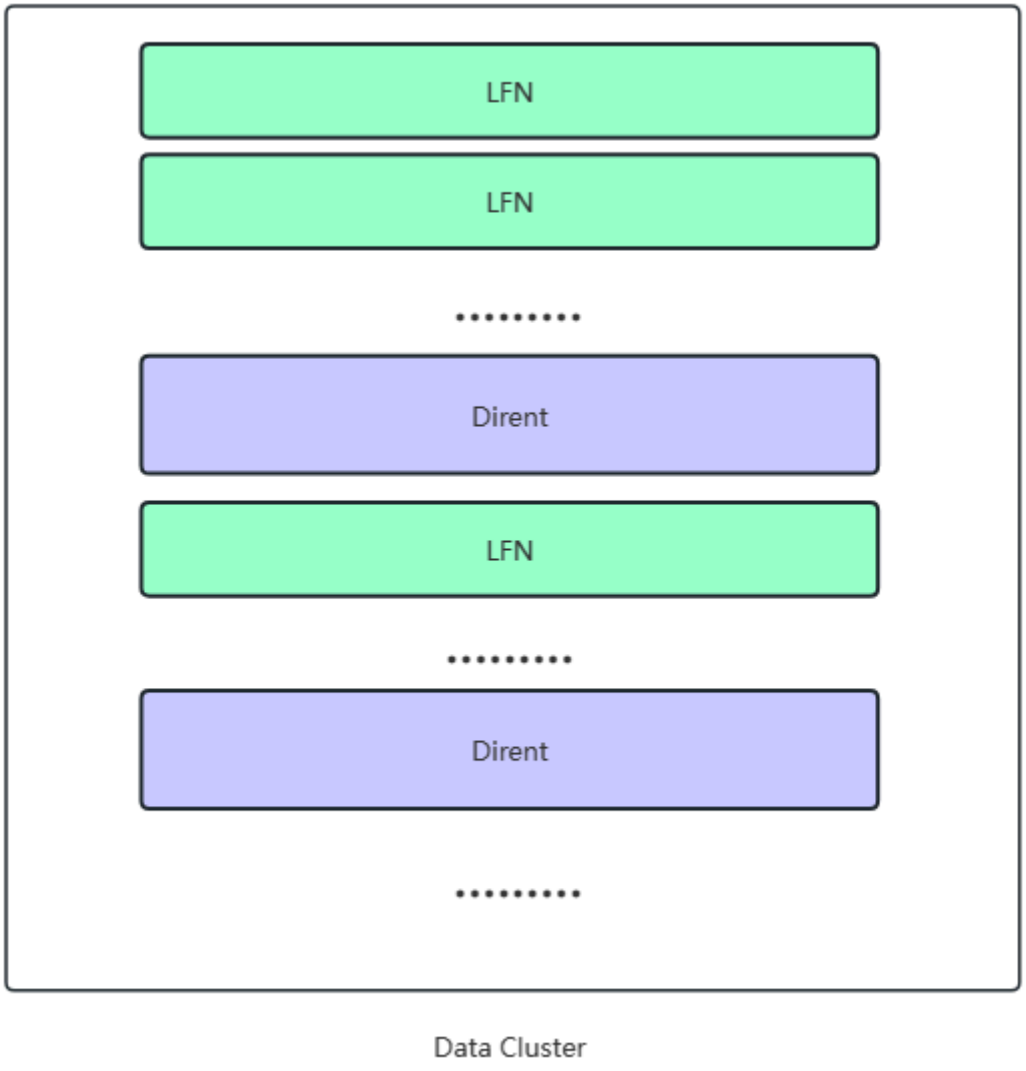
FAT

FAT 表用于记录文件数据簇的分配情况，每个表项对应一个簇，表项的值表示下一个簇的编号，最后一个簇的表项值大于等于0x0FFFFFF8；如果为0x0FFFFFF7，则表示该簇为坏簇。在FAT32中，每个表项占用4字节，但是只使用第28位。

Data Area

数据区用于存储文件数据，由多个簇组成，每个簇的大小由文件系统的参数决定。在 FAT32 文件系统中，数据区的第一个簇为根目录簇，簇号从2开始。

Dirent与LFN



FAT32 文件系统中的目录项(Dirent)结构如下,只需要注意文件名，文件属性，起始簇号以及文件大小：

- 文件名： 前8为为文件名，后3位为文件扩展名，不足8位的用空格 `0x20` 填充。
- 文件属性： 用于描述文件的属性，如只读、隐藏、系统文件等。当属性全都具备时，那么此时该Dirent则为一个LFN(长文件名),否则为普通Dirent。
- 文件大小： 文件的大小，以字节为单位，**请务必仔细观察 `.` 与 `..` 文件夹在Dirent中显示的文件大小。**

偏移	大小	描述
0x00	11	文件名。
0x0B	1	**文件属性。**
0x0C	1	保留。
0x0D	1	创建时间的 10 毫秒位。
0x0E	2	创建时间。
0x10	2	创建日期。

偏移	大小	描述
0x12	2	最后访问日期。
0x14	2	保留。
0x16	2	最后修改时间。
0x18	2	最后修改日期。
0x1A	2	起始簇号。
0x1C	4	文件大小。

LFN(长文件名)的结构如下：

偏移	大小	描述
0x00	1	序号。
0x01	10	文件名的前 5 个字符。
0x0B	1	文件属性。
0x0C	1	保留。
0x0D	1	校验和。
0x0E	12	文件名的中间 6 个字符。
0x1A	2	保留。
0x1C	4	文件名的最后 2 个字符。

需要注意，一个文件名最长为255字符，因此可能会有多个LFN来描述一个文件名。对于一个Dirent，它的LFN必定在它的前面，且序号从1开始；文件名中的字符是2字节，规定必须每个Dirent都有LFN；并且区分LFN与Dirent的方法是Dirent的文件属性，LFN的文件属性为0x0F。

FAT32 生成和挂载

同学们可以通过执行类似以下指令来创建所需的 FAT32 文件系统镜像文件(在 Debian 系统中，如果发现缺少工具，可以尝试使用 `sudo apt-cache search "<key_word>"` 来查找所需安装的包)。

```
sudo dd if=/dev/zero of=FAT32.img bs=4K count=16K status=progress
# dd指令用于数据复制，if表示数据的输入(这里使用/dev/zero表示将全0复制到该文件中)，of表示数据的输出，bs表示文件块的大小，count表示文件块的数量(1k=1024)，status=progress表示展示复制进度。执行完该命令会生成一个名为FAT32.img的64M空文件。
sudo mkfs.vfat FAT32.img -F 32
# 将FAT32.img文件转化为FAT32文件系统，-F 指定生成 FAT32文件系统,关于该指令的其他参数，请使用man指令查阅。
sudo mount FAT32.img <any_empty_dir>
# 使用挂载指令将该文件系统挂载到该文件夹下，挂载后可以直接将文件复制到该文件夹中从而达到向该系统传入文件的目的。
# ...
# 传输一些文件内容
# ...
sudo umount <the mounted dir>
# 需要解除对该文件系统的挂载从而保存相应内容
```

> 注: 跳板机上不允许执行 mount 指令，跳板机由 docker 容器构建，所以**没有 mount 的权限**，不能执行 mount 指令,这里推荐两种解决方法:
>
> 1. **强烈推荐**：使用`mtools`在跳板机创建和写入 FAT32 镜像
>
> `mtools`是 MS-DOS 文件系统（即 FAT 类文件系统）的工具程序，可模拟许多 MS-DOS 的指令，如 format、copy、dir 等等。使用此工具可以创建和写入 FAT32 镜像，步骤如下:
>
> ```
> #  安装 mtools
> sudo apt install mtools
>
> # fat.img为使用mkfs.vfat创建的空文件系统
>
> # 向其中添加一个文件 (include.mk)
> mcopy -i fat.img include.mk ::
>
> # 添加一个文件夹 (fs/)
> mcopy -i fat.img fs/ ::
>
> #  列出镜像内的文件或目录
> mdir -i fat.img ::
>
> #  从镜像中提取一个文件
> mcopy -i fat.img ::/include.mk extracted.txt
> ```
>
> 其中，命令中的 :: 表示参数 -i fat.img 中指定的镜像。如 ::/include.mk 就表示镜像根目录下的 include.mk 文件，详情可参见mtools
>
> ---
>
> 1. 在自己的本地 Linux 环境创建镜像并上传到跳板机
>
> 按照与指导书相同的方法创建镜像 fat.img 并向其中写入目录和文件后，通过 scp 指令或者是 vscode 即可拷贝到跳板机。

任务要求

- 通过自动评测。

- 在提交代码中附上说明文档。
- 查重。

接口定义

你需要在用户态实现如下接口。

> 注：对于**未特别标注**的接口，函数功能，参数以及返回值**与原文件系统中同名接口定义一致**

```
// file.c
int open(const char *path, int mode);
int read_map(int fd, u_int offset, void **blk);
int remove(const char *path);

int ftruncate(int fd, u_int size);
int sync(void);
// fd.c
int close(int fd);
int read(int fd, void *buf, u_int nbytes);
int write(int fd, const void *buf, u_int nbytes);
//! 请按照Linux中的实现，添加LSEEK_START,LSEEK_CURRENT和LSSEK_END（对应whence参数）
int seek(int fd, u_int offset, u_int whence);
void close_all(void);
int readn(int fd, void *buf, u_int nbytes);
int fstat(int fdnum, struct Stat *stat);
int stat(const char *path, struct Stat *stat);
```

测试内容

- 文件的增删读写。
- 文件缓冲区的使用。

参考文档

- 微软 FAT 规格书 Microsoft FAT Specification