EXT2 挑战性任务

本次挑战性任务的目标是在用户态进程中实现EXT2文件系统的基本文件读写功能，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。

EXT文件系统与演化

EXT文件系统自最早的`EXT`起，经历`EXT2`、`EXT3`的发展，至最新的`EXT4`文件系统，不断完善与进化。

`EXT`文件系统于1992年首次作为Linux的官方文件系统出现，旨在克服当时UNIX使用的`Minix`文件系统的若干限制，同时标志着世界上第一个采用`VFS`（虚拟文件系统）技术的文件系统的诞生。随着Linux的持续更新和迭代，EXT文件系统亦步亦进，现行的EXT4文件系统修复了先前版本的多项缺陷，被视为索引文件系统和日志文件系统优点集于一身的典范。

在`EXT2`文件系统中，基于EXT的基础，引入了块组概念以改善文件碎片化问题，这一创新显著提升了数据访问速度。此外，EXT2支持扩展属性，使文件和目录能关联更多的元数据。值得一提的是，EXT2提供了对软硬链接更好的支持（与`FAT`文件系统相比，后者不支持链接功能）。不过，EXT2并未支持日志功能，这一特性是在其后继`EXT3`中实现的。

EXT2的组织结构

EXT2文件系统以逻辑块作为存储单位，其中每个逻辑块包含若干个扇区，这些扇区数量是2的整数次幂；同时，若干个逻辑块构成一个块组，整个文件系统被划分为了这样的多个块组。

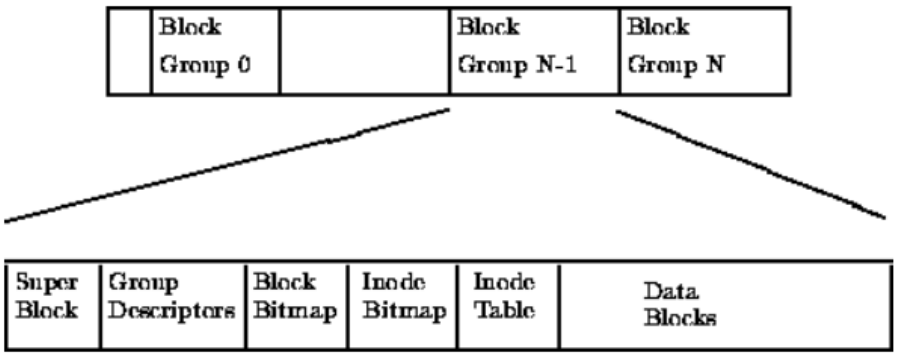


Figure: Physical Layout of the EXT2 File system

EXT2块组中最典型的结构如上，该块组包含了：

- SuperBlock：存放整个文件系统的相关信息。
- Group Descriptor Table： 存放每个组元数据结构的数组。
- InodeTable： 存放Inode的数组。
- DataBlockBitMap：数据块是否被使用的Bitmap。
- InodeBitmap：Inode是否被使用的Bitmap。
- DataBlock：存放Inode对应的对应的数据。

需要注意SuperBlock和Group Descriptors 在特定块组中都有备份，其他块组中不存在这些内容。

SuperBlock

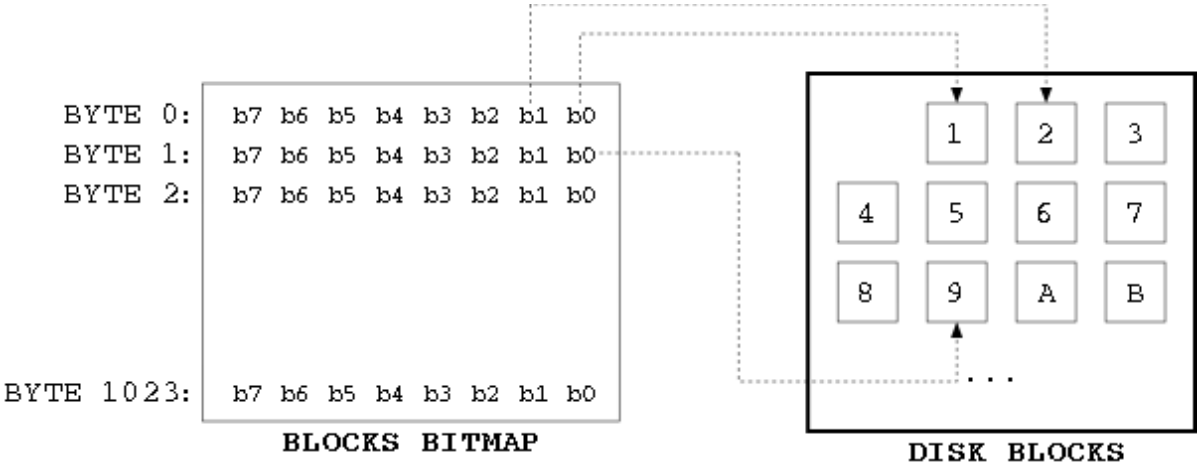
SuperBlock是EXT2文件系统的元数据之一，存放了文件系统的基本信息，如文件系统的大小、块大小、Inode数量等。SuperBlock中的信息可以帮助操作系统找到文件系统中的其他元数据，如Inode Table、Group Descriptor Table等。

GDT与IT

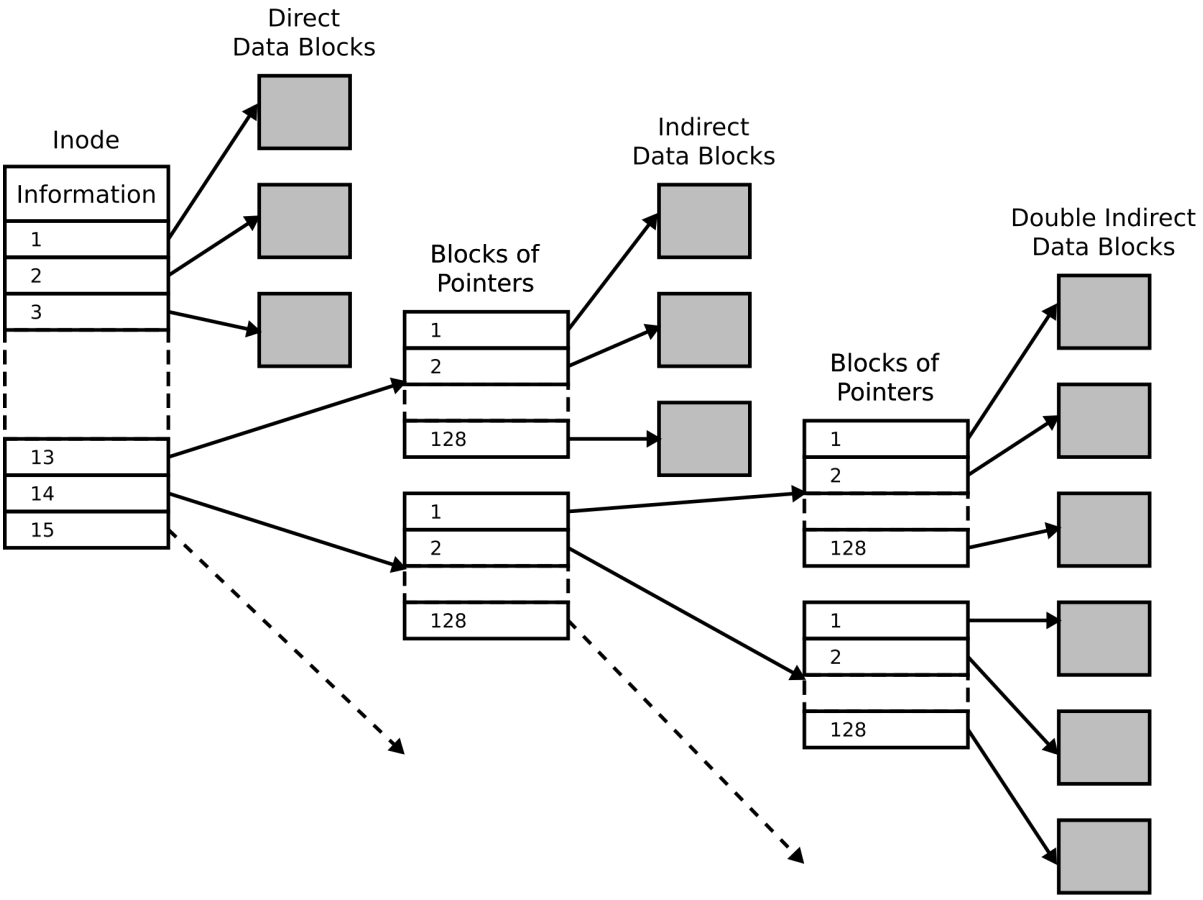
Group Descriptor Table（GDT）存放了每个块组的元数据，如每个块组的起始数据块号、空闲数据块数、空闲Inode数等。Inode Table（IT）存放了Inode的数组，每个Inode对应一个文件或目录的元数据，如文件大小、文件权限、文件类型等。这两个可看作为两个数组，通过GDT中的信息可以找到对应的Inode，通过Inode中的信息可以找到对应文件的数据块。

Bitmap

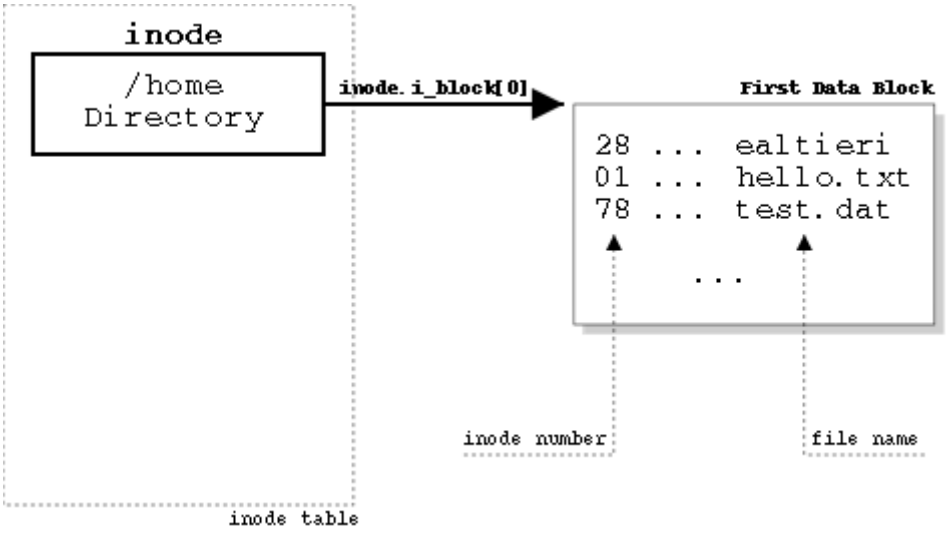
Bitmap是一种数据结构，用于记录数据块或Inode是否被使用。在EXT2文件系统中，每个块组都有一个数据块位图和一个Inode位图，用于记录数据块和Inode的使用情况。当一个数据块或Inode被使用时，对应的位图中的相应位会被置为1，否则为0,下图以block bitmap为例：



Inode与DataBlocks



EXT2文件系统中的Inode结构如上，Inode结构中包含了文件的元数据，如文件大小、文件类型、文件权限等。Inode结构中还包含了指向文件数据块的指针，这些指针指向了文件的实际数据块。



对于一个被Inode指向的Data Block，如果该Inode是一个目录文件，那么该Data Block中存放的是目录项，每个目录项包含了文件名和Inode号。如果该Inode是一个普通文件，那么该Data Block中存放的是文件的实际数据，上图展示了一个目录文件的Data Block的结构。

想要获取更多关于EXT2文件系统的内容，可以查阅参考文档中的Wiki或者早期Linux中的实现。

EXT2文件系统的生成与挂载

同学们可以通过执行以下指令来创建所需的EXT2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 search "<tool_name>"`来查找并安装所需的包）。

```
sudo dd if=/dev/zero of=EXT2.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EXT2.img的64M空文件。
sudo mkfs.EXT2 EXT2.img
# 将EXT2.img文件转化为EXT2文件系统（在其中导入所需的SuperBlock以及Inode等结构，关于该指令的其他参数，请使用man指令查阅。
sudo dumpe2fs EXT2.img
# 这里将会打印出生成的EXT2文件系统的相关信息。
sudo mount EXT2.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
```

执行完上述指令就得到了所需的EXT2文件系统。通过qemu的的drive选项挂载该文件系统（和mos原先的挂载方式相同）即可被内核使用。

任务要求

接口定义

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**同原文件系统**):

```
// 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);

//! 创建符号链接。
//! 成功则返回0，否则返回 < 0的值(这里不做特别要求)。
int symlink(const char *target, const char *linkpath);
//! 创建硬链接。
//! 成功则返回0，否则返回 < 0的值(这里不做特别要求)。
int link(const char *oldpath, const char *newpath);
```

测试内容

1. 文件的增删读写。
2. 软硬链接的创建，删除和读写。

参考文档

1. EXT2 - OSDev Wiki
2. nongnu.org/EXT2-doc/
3. EXTended file system - Wikipedia
4. EXT2文件系统详解-CSDN博客

这项挑战性任看起来有些复杂，但其实mos原来的FS层和EXT2有很多类似的地方，笔者相信所有选择该挑战性任务的人最终都能跳转到成功的时间线。El Psy Kongroo。