
同济大学计算机系

操作系统课程设计报告

Ljyux 设计及实现



学 号 1951444

姓 名 林佳奕

专 业 计算机科学与技术

授课老师 方钰

目录

一、	需求分析	4
1.1	课设任务要求.....	4
1.2	程序任务.....	5
1.3	开发运行环境.....	6
二、	概要设计	7
2.1	任务分解.....	7
2.2	数据结构定义.....	8
2.3	模块间调用关系.....	11
2.4	算法说明.....	12
三、	详细设计	18
3.1	高速缓存块算法.....	18
3.2	磁盘读写算法.....	20
3.3	空闲盘块管理算法.....	21
3.4	空闲 DiskInode 管理算法.....	23
3.5	文件索引结构.....	24
3.6	目录管理.....	25
3.7	文件管理.....	28
3.8	用户管理.....	32
3.9	项目结构.....	33
四、	运行结果分析	35
4.1	逐条命令简单测试.....	35
4.2	主程序测试.....	42
4.3	课设测试.....	46
4.4	多用户读写测试.....	47
五、	用户使用说明	51
5.1	操作说明.....	51
5.2	可用命令及命令格式说明.....	51
六、	总结与心得体会	58

6.1	心得体会.....	58
6.2	总结反思.....	58
七、	参考资料.....	59

一、需求分析

1.1 课设任务要求

使用一个普通的大文件(如 c:\myDisk.img ,称之为一级文件)来模拟 UNIX V6++的一个文件卷(把一个大文件当一张磁盘用)

(1) 磁盘文件结构

- 定义自己的磁盘文件结构
- SuperBlock 结构
- 磁盘 Inode 节点结构, 包括: 索引结构
- 磁盘 Inode 节点的分配与回收算法设计与实现
- 文件数据区的分配与回收算法设计与实现

(2) 文件目录结构

- 目录文件结构
- 目录检索算法的设计与实现

(3) 文件打开结构

(4) 磁盘高速缓存

(5) 文件操作接口

- fformat: 格式化文件卷
- ls: 列目录
- mkdir: 创建目录
- fcreat: 新建文件
- fopen: 打开文件
- fclose: 关闭文件
- fread: 读文件
- fwrite: 写文件
- flseek: 定位文件读写指针
- fdelete: 删除文件
- ...

(6) 主程序

- 格式化文件卷；
- 用 `mkdir` 命令创建子目录，建立如图所示的目录结构；
- 把你的课设报告，关于课程设计报告的 `ReadMe.txt` 和一张图片存进这个文件系统，分别放在 `/home/texts`，`/home/reports` 和 `/home/photos` 文件夹；
- 图形界面或者命令行方式，等待用户输入；
- 根据用户不同的输入，返回结果。

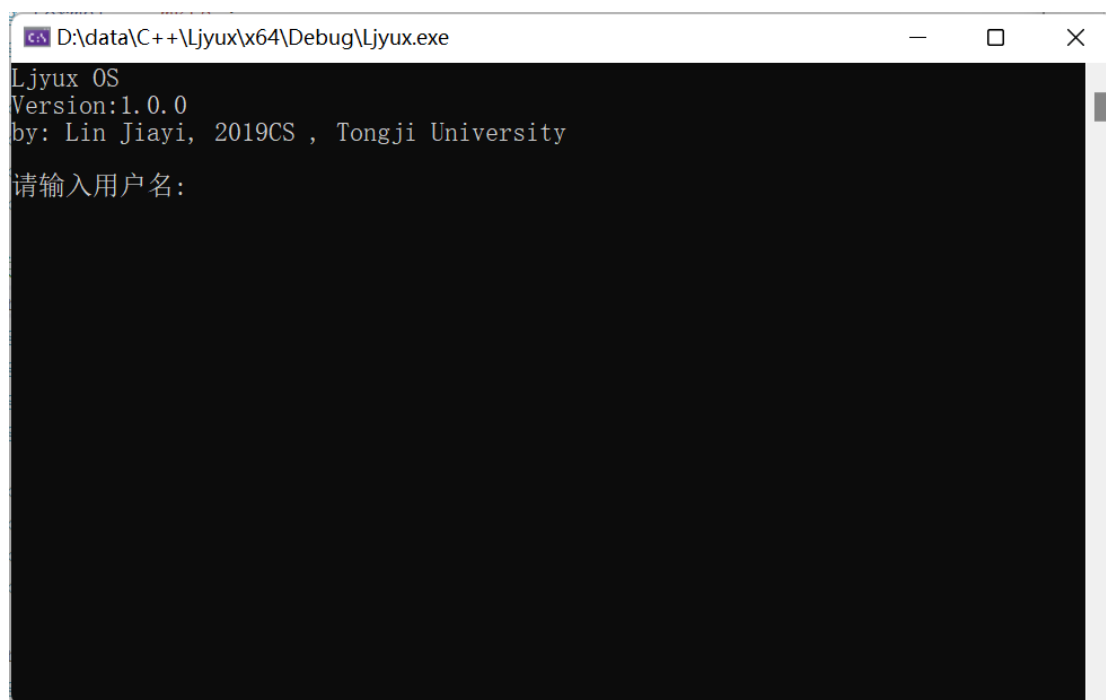
(7) 命令行测试

- 新建文件 `/test/Jerry`，打开该文件，任意写入 800 个字节；
- 将文件读写指针定位到第 500 字节，读出 500 个字节到字符串 `abc`。
- 将 `abc` 写回文件。

1.2 程序任务

1.2.1 输入输出形式

程序输入、输出均在命令行下实现，这与 Windows 的 `cmd` 命令行类似。



```
D:\data\C++\Ljyux\x64\Debug\Ljyux.exe
Ljyux OS
Version:1.0.0
by: Lin Jiayi, 2019CS , Tongji University
请输入用户名:
```

程序输入一行命令，有具体的格式要求，后文会介绍。

程序输出相应内容到命令行下，若有错误信息会给出相应提示，同样在后文

介绍。

1.2.2 程序功能

- 创建并格式化文件卷
- 目录相关操作，包括：创建目录、删除目录、查看目录、更改当前工作目录等。
- 文件相关操作，包括：创建文件、删除文件、打开关闭文件、读写文件、移动读写指针、更改文件权限等。
- 用户相关操作，包括：创建用户、删除用户、更换用户登录、修改用户分组、查看用户信息等。
- 其他操作，包括：查看 inode 节点信息，查看盘块数据，帮助命令等。

1.2.3 完成程度

- 实现了课设要求的必做部分
- 完成了高速缓存块和多用户读写部分
- 扩展了可用命令数量。

1.3 开发运行环境

开发环境所在操作系统为 Windows 11, 开发所用 IDE 为 Visual Studio 2022, 开发所用编程语言为 C++。

二、概要设计

2.1 任务分解

一个文件系统，包含了磁盘数据管理、目录管理、文件管理、用户管理等功能，因此将整个文件系统分为以下几个部分。

2.1.1 磁盘读写模块 DiskDevice

这部分的功能主要负责：

- 文件卷的读写（基于高速缓存块算法）

可以指定结构体的读写。

2.1.2 磁盘管理模块 FileSystem

这部分主要负责：

- 文件卷的底层格式化（写入数据）
- SuperBlock 管理（包括空闲盘块的申请释放，空闲 Inode 申请释放）

2.1.3 文件管理模块 FileManager

这部分的主要负责目录项的管理，主要功能包括：

- 目录项 Inode 的操作（例如目录搜索）
- 创建、删除目录项

2.1.4 内存 Inode 表 InodeTable

这部分主要封装了内存 Inode 的申请、释放和查找

2.1.5 打开文件表 FileTable

这部分主要封装了打开文件结构 File 的申请、释放和查找

2.1.6 用户表 UserTable

这部分主要封装了用户结构 User 的创建、删除和查找

2.1.7 内核模块 Ljyux

Ljyux，即 Lin Jiayi Unix，是本文件系统的核心。

该部分提供所有命令的接口，并调用前文提到的模块。

2.1.8 命令解析模块 CommandParser

对每一条命令进行解析，判断格式是否正确，若正确调用 Ljyux。若格式错误，给出提示。

2.1.9 顶层输入模块 CLI

提供输入，输出接口

2.2 数据结构定义

2.2.1 盘块结构定义

为了实现多用户读写，对给定的盘块划分做出了一些调整，如下：

OS保留	SuperBlock	Inode区	Data区
2 Blocks	2Blocks	128Blocks	65,404Blocks
0	2	4	132

以下是四个部分的说明：

- OS 保留区：主要负责存放用户信息。
- SuperBlock：同 Unix
- Inode 区：同 Unix，存放 DiskInode，每个 DiskInode 大小为 32B
- Data 区：存放数据盘块

2.2.2 高速缓存块 IO_Buffer

高速缓存块实际上就是一个盘块+相关信息。结构如下

```
struct IO_Buffer {  
    char    m_buf[BLOCK_SIZE]={0};    //缓存块  
    int     blk_id = -1;                //对应blk号  
    int     m_nutime = 0;               //上次未被使用时间  
};
```

blk_id 对应着磁盘块的 id，范围为 0-65535。

m_nutime 是 LRU 算法中的计数器，替换时选择缓存块的指标。

2.2.3 SuperBlock

这部分与 Unix V6++的 SuperBlock 定义类似。大小为 1024B

```
struct SuperBlock {  
    //管理Inode  
    int s_isize;                //inode区总数  
    int s_ninode;               //空闲inode数量  
    int s_inode[100];           //直接管理的空闲DiskInode
```

```

//管理数据盘块
int s_fsize;           //数据文件区盘块总数
int s_nfree;           //直接管理的空闲盘块数量
int s_free[100];       //直接管理的空闲盘块索引

int padding[52];       //填充
};

```

其中 `s_isize`、`s_ninode` 和 `s_inode` 负责管理 Inode 区，`s_fsize`、`s_nfree`、`s_free` 负责管理数据盘块。

2.2.4 DiskInode 和 Inode

DiskInode 为磁盘 Inode，存于磁盘的 Inode 区，大小为 32B

```

struct DiskInode {
    int    i_addr[10] = { 0 };    //逻辑盘块到物理盘块映射
    int    i_mode = 0;            //Inode的读写权限
    int    i_size = 0;            //Inode对应的文件大小
    short  i_uid = -1;            //Inode持有者的用户的id
    short  i_gid = -1;            //Inode持有者的用户所在组的id
    int    i_type = 0;            //Inode对应的文件类型
    int    i_inode_id = 0;        //Inode编号
    int    d_mtime = 0;          //最后访问时间
};

```

其中，`imode` 使用到最低 9 位，其中 2-0（最右为 0 位）对应其他用户读、写、执行（R、W、X）权限，5-3 位对应同组用户 RWX 权限，8-6 位对应持有者用户读、写、执行权限，使用一个枚举类型管理之。

```

enum InodeMode {
    USER_R = 0x400,
    USER_W = 0x200,
    USER_X = 0x100,
    GROUP_R = 0x040,
    GROUP_W = 0x020,
    GROUP_X = 0x010,
    OTHER_R = 0x004,
    OTHER_W = 0x002,
    OTHER_X = 0x001
};

```

`i_type` 表明文件类型（目录文件或数据文件），同样使用枚举类型管理。

```

enum InodeType {
    DATA_FILE = 0x01,

```

```

    DIRECTORY_FILE = 0x02,
};

```

Inode 即内存 Inode，除了 DiskInode 指针以外，还有一些其他信息。

```

class Inode{
private:
    FileSystem*& m_pfs=g_fs;
    //FileSystem指针（实为全局变量指针）
    DiskDevice*& m_Disk=g_Disk;
    //DiskDevice指针
public:
    int          m_count = 0;  //引用计数
    int          m_id = -1;    //磁盘Inode的id号
    DiskInode* m_pinode;      //磁盘Inode指针
    .....
};

```

2.2.5 打开文件结构体 File

文件打开结构定义如下：

```

class File{
public:
    Inode* m_pinode;    //Inode 指针
    int    m_inode_id;  //Inode id
    int    m_fmode;     //文件打开方式
    int    m_foffset;   //文件读写指针
public:
    File();
    ~File();
    void setF(Inode* pinode, int fmode);
};

```

f_mode 表明文件的打开方式（只读或读写），使用一个枚举类型管理。

```

enum OpenFileMode {
    FREAD = 0x1,
    FWRITE = 0x2,
};

```

m_foffset 是文件读写指针。

2.2.6 用户结构 User

首先是存放用户信息的结构体 UserInfo

```

struct UserInfo {
    short    m_uid = -1;
    short    m_gid = -1;
    char     m_name[USERNAME_MAXLEN + 1] = { 0 };
    char     m_passwd[PASSWD_MAXLEN + 1] = { 0 };
};

```

其中 m_uid 为用户 id 号, m_gid 为用户组 id 号, m_name 为用户名, m_passwd 为用户密码。

接下来是用户结构体 User

```

class User{
public:
    UserInfo* m_uinfo;
    OpenFiles* m_files; //打开文件表
    map<int, int> m_inodemap;//inode_id -> fd
public:
    .....
};

```

其中, m_uinfo 为用户信息, m_files 为用户打开的所有文件, m_inodemap 为 inode_id (磁盘 inode 号) 到 fd (文件句柄) 的映射。

2.3 模块间调用关系

由于某些模块间调用没有层次性, 因此本实验中定义了几个全局对象, 供其他模块调用。这与 Unix V6++ 的设计思想是类似的。

全局变量和对象定义如下:

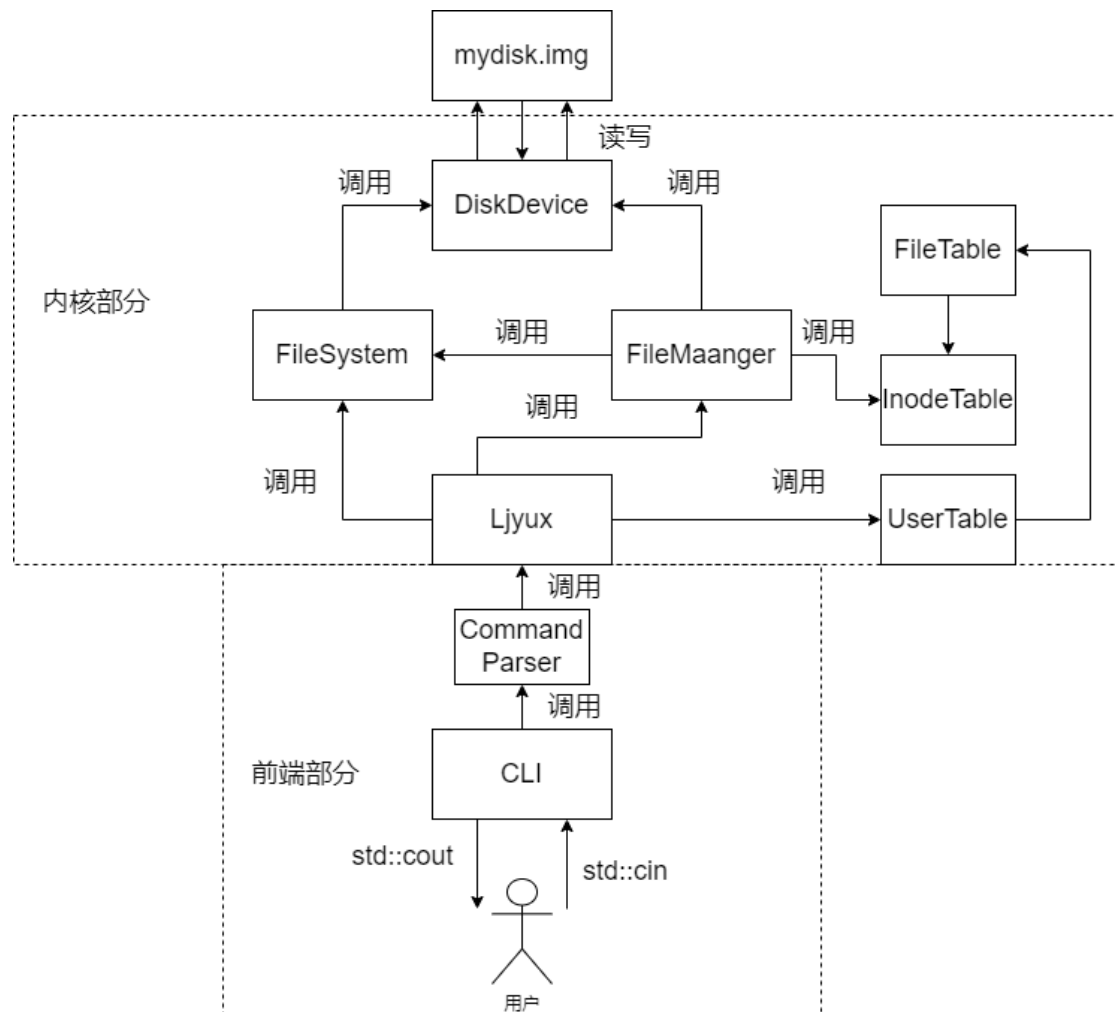
```

const char* disk_fp = "D:\\mydisk.img";
DiskDevice* g_Disk = nullptr;
FileSystem* g_fs = nullptr;
OpenFileTable* g_ftable = nullptr;
InodeTable* g_itable = nullptr;
UserTable* g_utable = nullptr;

```

其中, 这些指针的申请和释放在 Ljyux 类中实现。

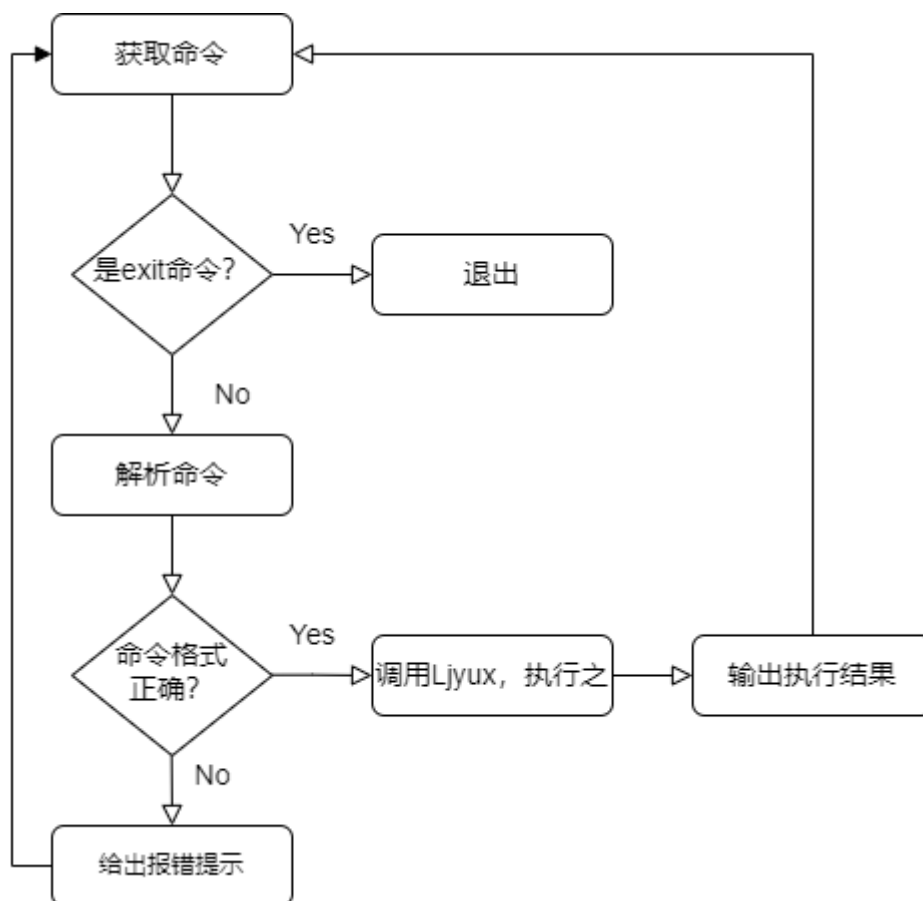
这里再给出具体的模块调用关系。



2.4 算法说明

2.4.1 命令执行流程

大致的流程是：从标准输入流 `cin` 获取命令字符串 → 判断命令类型，交付 `Command Parser` 解析 → 判断命令格式和参数，交付 `Ljyux` 执行 → 执行后，返回结果。



2.4.2 命令分析

● mkfs

由于格式化相当于直接把系统恢复成初始状态，因此所有的变量和对象都需要清空。因此，这里采取的方式是析构所有的全局对象，再重新构建（注意：DiskDevice 类一定要最后析构，最先构造，因为其他对象会调用这个类）。

之后，重写文件卷四个分区（OS 保留区，SuperBlock 区，Inode 区，Data 区）。

在之后，创建根目录/，修改当前工作目录为根目录，并设置当前登录用户为 root 用户。

● mkdir

要求：(1) 仅父目录存在，但不允许目录名重复。

（例如，有/bin 目录，则可以创建/bin/lib，不可以创建/dev/lib）

之后，获取路径名，申请新目录 Inode，并向其中写入. 和.. 目录项。

之后，父目录文件中加入子目录项。完成创建。

- `rmdir`

要求：(1)路径存在(2)目录为空(3)待删除目录不是根目录

满足以上要求后，可以删除目录。对于待删除的目录文件，回收其所有的盘块，并回收其 Inode。之后，父目录中，删除此目录项。

- `ls`

要求：(1)路径存在且必须是目录文件。

满足以上要求，便可以取出目录文件对应的 Inode，并输出相应内容。

- `cd`

要求：(1)路径存在且必须是目录文件。

满足以上要求，便可以更改目录，修改 `Ljyux` 中的 `m_curdir` 变量，并修改 `FileManager` 中的 `m_curInodeid` 即可。

- `mkfile`

要求：(1)路径中，上一级路径存在，此路径不存在（`Ljyux` 不支持覆盖）

满足以上要求，便可以创建数据文件。首先申请一个 Inode，并修改相关信息。之后，取出其父目录，添加新文件对应的目录项。

- `rmfile`

要求：(1)路径必须存在且为数据文件(2)当前用户对此文件具有读写权限(3)该文件未被打开。

符合以上条件，便可以删除文件。对于此文件，回收其所有的盘块和 Inode。之后，取出其父节点，删除这一文件对应的目录项。

- `fopen`

要求：(1)路径必须存在且为数据文件(2)用户具有打开该文件的权限。

满足以上要求，便可以打开文件。申请打开文件结构 `File`，并在用户打开文件表中进行勾连。

- `fclose`

要求：(1)路径必须存在且为数据文件(2)文件已经被打开

满足以上条件，便可以关闭文件。取出文件对应的 `File` 结构，递减 Inode

的引用数 `m_count`，若减为 0，回收此 Inode。之后，用户打开文件表中回收此 File 结构。在 `OpenFileTable` 中回收此 File 结构。

- `fread`

要求：(1) 路径必须存在且为数据文件 (2) 文件已被打开，且具有读方式
满足以上条件，便可以读取文件。取出 File 结构后，对其 Inode 进行读取，并调整 File 结构的 `m_offset`。

- `fwrite`

要求：(1) 路径必须存在且为数据文件 (2) 文件已被打开，且具有写方式
满足以上条件，便可以读取文件。取出 File 结构后，对其 Inode 进行读取，并调整 File 结构的 `m_offset`。

- `fseek`

要求：(1) 路径必须存在且为数据文件 (2) 文件已被打开
满足以上条件，便可以修改文件指针。根据调整方式，修改 `m_offset`。

- `cat`

要求：(1) 路径必须存在，且为数据文件 (2) 文件已被打开，且具有读模式
之后，便可以读取整个文件。过程类似于 `fread`，不再复述。

- `load`

要求：(1) 源文件（宿主机文件）存在 (2) 目标文件（Ljyux 文件）存在，且已打开。

满足以上条件，便可以导入文件。从源文件依次读取 512 字节，再写入目标文件中，重复直到源文件读完。

- `dump`

要求：(1) 源文件（Ljyux 文件）存在 (2) 目标文件（宿主机文件）存在，且已打开。

满足以上条件，便可以导出文件。从源文件依次读取 512 字节，再写入目标文件中，重复直到源文件读完。

- `whoami`

直接输出当前用户的名称，uid 和 gid。

- adduser

要求：(1)当前用户必须是 root 用户（只有 root 用户可以添加用户）(2)用户名未被使用(3)用户表未满

满足以上条件，便可以添加用户。申请好 User 结构，填入相关信息即可。

- deluser

要求：(1)当前用户必须是 root 用户（只有 root 用户可以删除用户）(2)用户存在且不是 root 用户（删除 root 用户？反了你！）

满足以上条件，便可以删除用户。将该用户对应的 User 项从 UserTable 中除去即可。

- su

要求：(1)用户存在(2)用户存在，且密码正确。

满足条件后，便可以更换用户。直接修改 Ljyux 的 m_user 结构即可。

- setgroup

要求：(1)当前用户必须是 root 用户(2)待修改 gid 的用户存在

满足条件后，便可以设置用户组。修改对应 User 结构的 gid 即可。

- lsinode

要求：(1)对应 Inode 号存在

满足条件后，取出对应 Inode，输出相关信息。

- lsblk

要求：(1)盘块号在 0-65535 之间

满足条件后，读出对应盘块，将每个字节按十六进制输出即可。

- lsuser

直接从 UserTable 取出所有用户，并输出即可。

- chmod

要求：(1)目标文件必须存在(2)待修改权限的文件必须是自己文件。

满足要求后，修改即可。

- setmod

要求：(1) 目标文件必须存在 (2) 待修改权限的文件必须是自己文件。

满足要求后，修改即可。

- help

根据命令名输出即可。

三、详细设计

以下将叙述各模块的重点算法。

3.1 高速缓存块算法

3.1.1 基本原理

本系统中，高速缓存块采取 LRU 算法，即：在需要替换的时候，优先选择最久未被使用的缓存块。

缓存块的定义如下：

```
struct IO_Buffer {  
    char    m_buf[BLOCK_SIZE]={0};    //缓存块  
    int     blk_id = -1;                //对应blk号  
    int     m_nutime = 0;               //上次未被使用时间  
};
```

可以看到，用于判断“未被使用时长”的变量是 `m_nutime`。而 `m_buf` 是磁盘块的镜像，`blk_id` 是标签。

3.1.2 替换算法

根据前文的描述，使用缓存块的前，我们需要准备一块，即在所有的缓存块中遍历。假设我们要写入的盘块号为 `k`，那么会遇到以下 3 种情况。

- 在所有的缓存块中，已有 `blk_id == k` 的缓存块
- 在所有的缓存块中，没有 `blk_id == k` 的缓存块，但有缓存块未被使用。
- 在所有的缓存块中，没有 `blk_id == k` 的缓存块，并且所有的缓存块都被占用。

对于情况 a，很简单，我们只需要取出这一块供使用即可；

对于情况 b，其实也很简单，我们只需要找出一个未被使用的缓存块即可；

而对于情况 c，则较为复杂，我们不仅要选出最久未被使用的盘块，还要将这个盘块同步到文件卷中以免丢失内容。此外，大部分情况下我们还要将这个盘块从文件卷中读出来，因为即使是写盘块，我们也要先取出来以免写入的时候丢失信息。

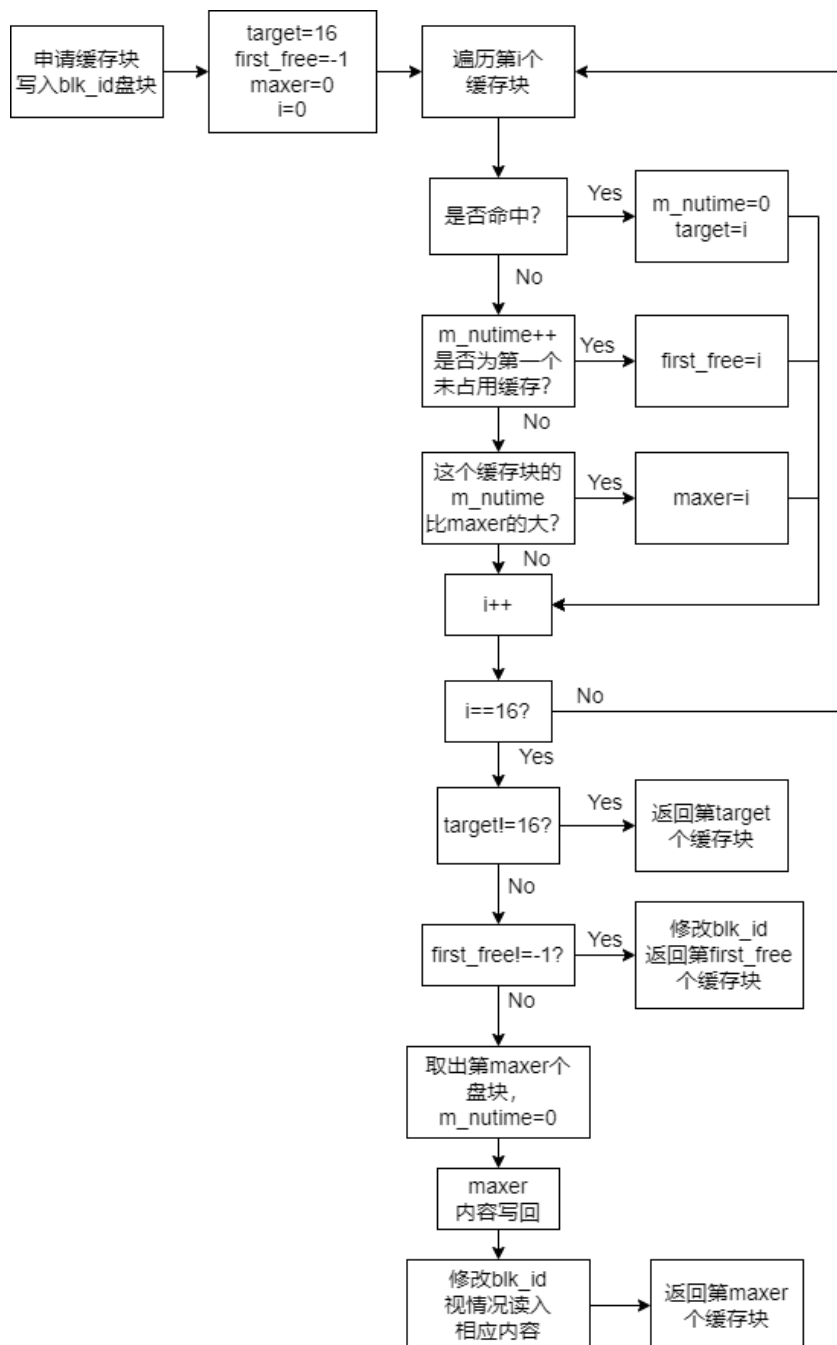
显然，为了处理情况 c，我们必须维护所有盘块的未使用情况。做法是：遍历的时候，如果缓存块已被占用，并且未命中，则其对应的 `m_nutime++`；若命

中，则修改 m_nutime 为 0。

因此大体的流程为：遍历所有缓存块，记下：若命中对应的缓存块号（没有则另标记，下同）、第一个未被使用的缓存块和 m_nutime 最大的缓存块（记为 $target$ ），遍历时，未命中的且被占用的缓存块的 $m_nutime++$ 缓存块。

之后判断，若命中，则直接返回对应的缓存块。否则，若有未被占用的缓存块，则返回之。否则，取出缓存块 $target$ ，将其写回盘块后，修改其相应信息，并视情况将新的盘块读出来。

整个过程的流程图如下：



3.1.3 与 Unix V6++的缓存块算法的对比与分析

相较于 Unix V6++的缓存块算法，本算法更加简单一些，并且理解和修改也更加容易。

局限性在于，本算法仅仅是对数据盘块的镜像，并不具备延迟写、异步写的功能（因为本系统使用了 C++的 `fstream` 对文件卷进行读写），并且此算法所有情况的时间复杂度均为 $O(n)$ 。

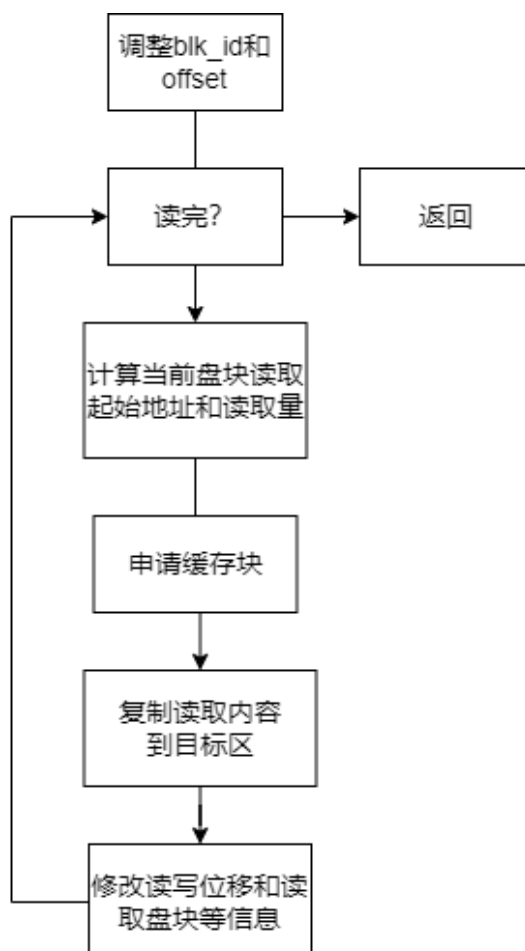
3.2 磁盘读写算法

3.2.1 磁盘读算法

输入：盘块号 `blk_id`，偏移 `offset`，目标变量指针 `t`，大小 `size`。

输出：无（读出目标以指针的形式返回）

流程：调整 `blk_id`，`offset`，使 `offset` 处于 $[0, 511]$ 之间 → 开始读，计算当前盘块读取首地址和读取量 → 申请盘块 → 将缓存块内容复制到目标区中 → 调整读取量，待读取盘块号递增。重复直到读完应读的大小。

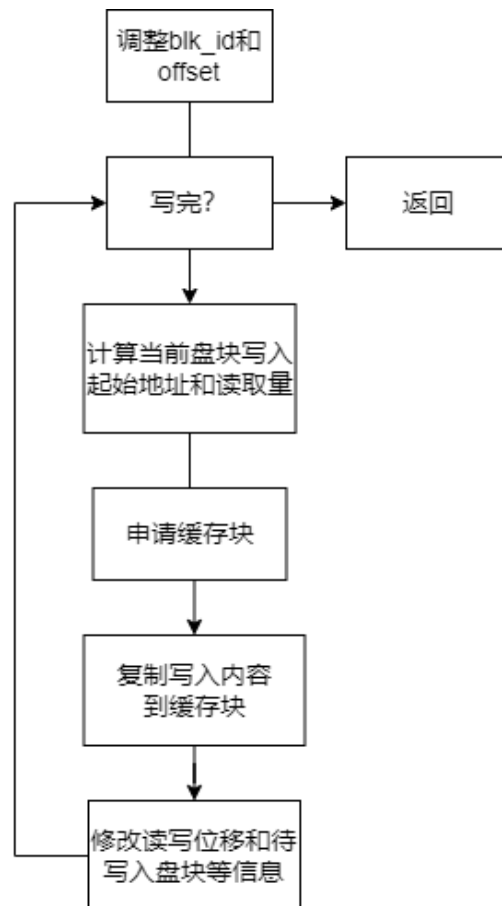


3.2.2 磁盘写算法

输入：盘块号 blk_id, 偏移 offset, 来源变量指针 t, 大小 size。

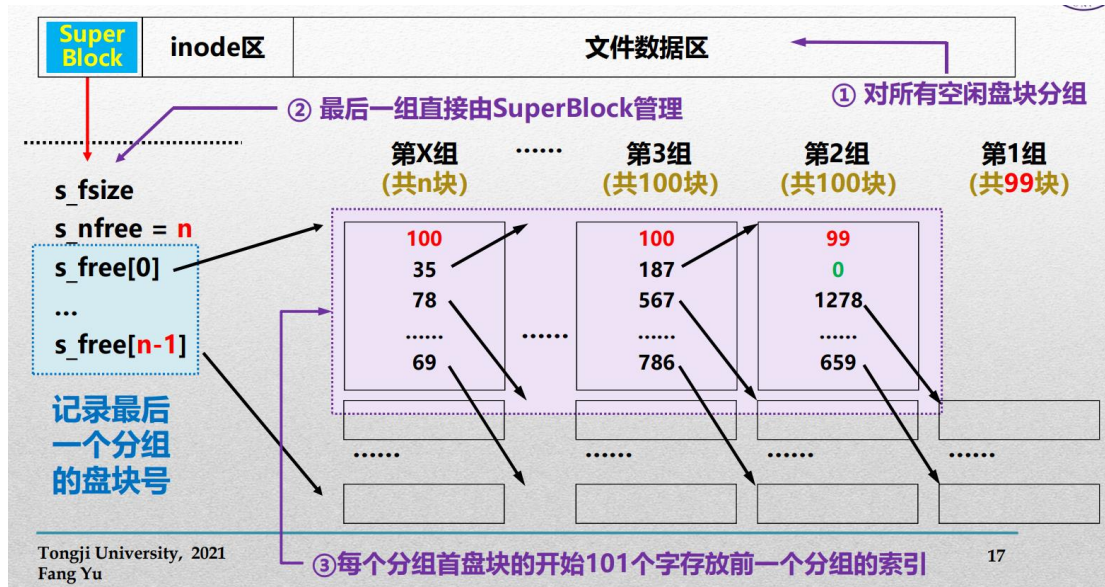
输出：无

流程：调整 blk_id, offset, 使 offset 处于 [0, 511] 之间 → 开始写, 计算当前盘块写入首地址和写入量 → 申请盘块 → 将待写入内容复制到缓存块中 → 调整写入量, 待写入盘块号递增。重复直到写完应写的大小。



3.3 空闲盘块管理算法

本实验中，空闲盘块的组织方式为成组连接法，这与 Unix V6++ 的实现方式是一样的。下图是我参考的成组连接法表示方式。



3.3.1 初始化

在格式化步骤中，我们就需要开始为盘块分组。具体的做法是从数据盘块区开始，每 100 个盘块为一组，每组序号最小者为“组长”。“组长”负责记录下一个小组的信息，包括盘块数目和组员编号（即组员的盘块号）。

这些信息占用盘块的 404 个字节。其中，组员编号按照升序排列。

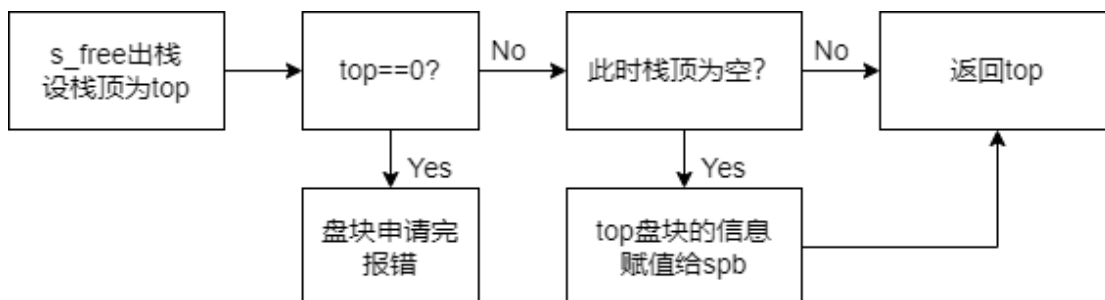
第一个小组的信息被写入到 SuperBlock 中。

3.3.2 空闲盘块申请

申请采用栈式的方式申请，其中栈空间即为 SuperBlock 结构的 s_free 数组，栈顶为 s_nfree。

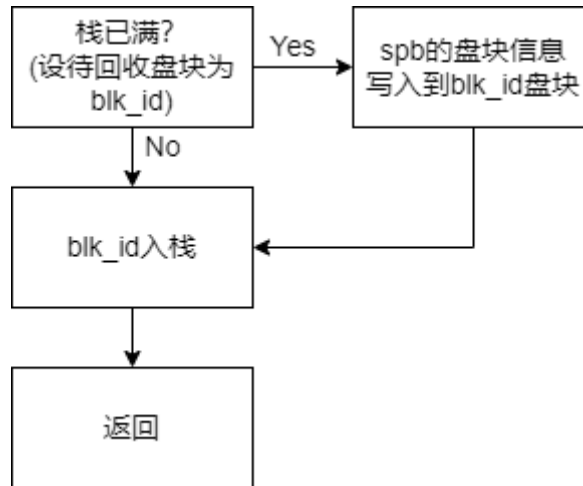
如果栈不为空，但盘块已经申请完，则说明空间不足，会给出相应提示。

否则，取出此盘块。如果此时栈为空，要注入下一组盘块信息，而这个信息就在刚才取出的盘块中，将其读出，并赋值给 s_free 和 s_nfree 即可。



3.3.3 空闲盘块释放

释放同样采用栈式的方式。如果栈已满，则把小组的信息放入回收的盘块中，并将栈清空。之后，待回收盘块号入栈。



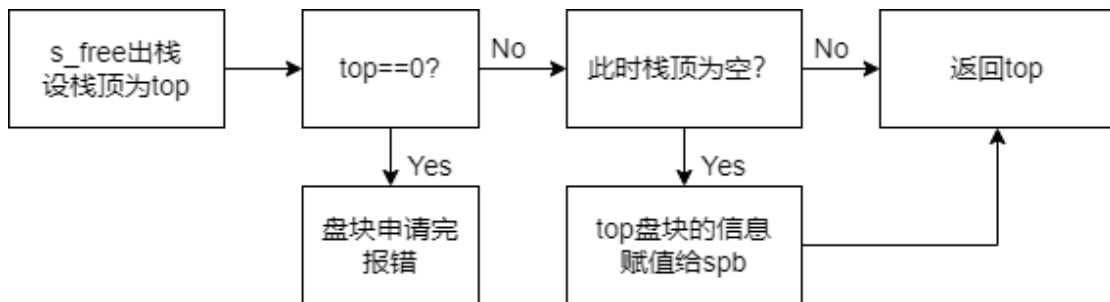
3.4 空闲 DiskInode 管理算法

同样采用栈式的管理方式，SuperBlock 直接管理的空闲 Inode 数目为 100，但不再采用成组的管理。

3.4.1 空闲 DiskInode 申请

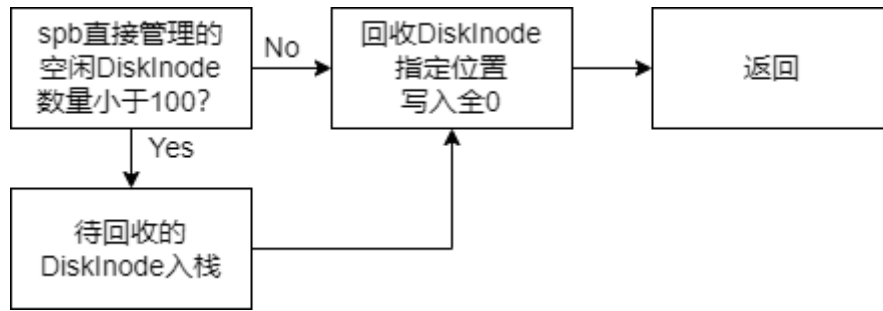
如果 SuperBlock 直接管理的空闲盘块已经用光了，则在 Inode 区进行搜索。搜索后，如果仍找不到空闲 Inode，则空闲 Inode 已申请完，给出报错信息。

否则，栈顶出栈并返回。



3.4.2 空闲 DiskInode 释放

如果 SuperBlock 直接管理的 Inode 尚未满，则入栈。之后，将空白 DiskInode 写回到相应位置即可。

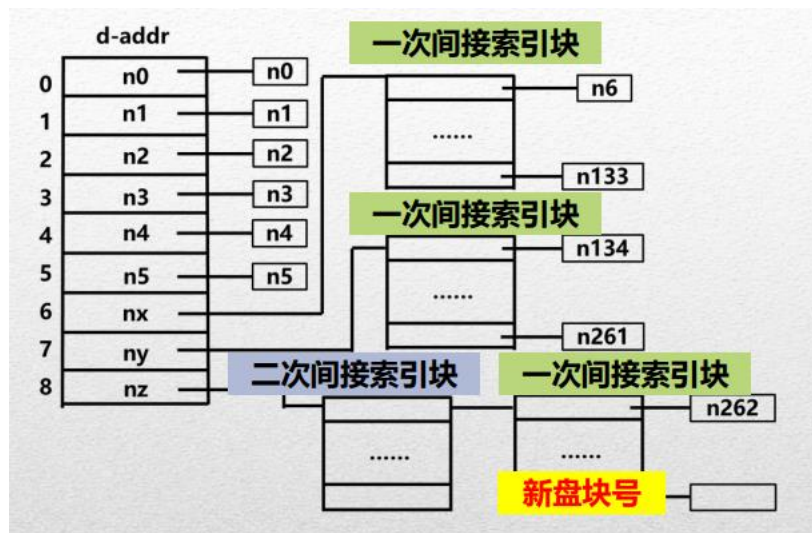


3.5 文件索引结构

3.5.1 索引结构

Ljyux 的文件索引结构与 Unix V6++ 的文件索引结构相同。在 DiskNode 中，用一个长为 10 的数组管理。其中下标 0-5 是 0 级索引，直接存放数据盘块的 blk_id；下标 6-7 对应 1 级索引盘块，1 级索引盘块存放着 0 级索引号，可存放 128 个；下标 8-9 对应 2 级索引盘块，存放 1 级索引盘块号，同样可存放 128 个。

Ljyux 参考了如下的文件索引结构。



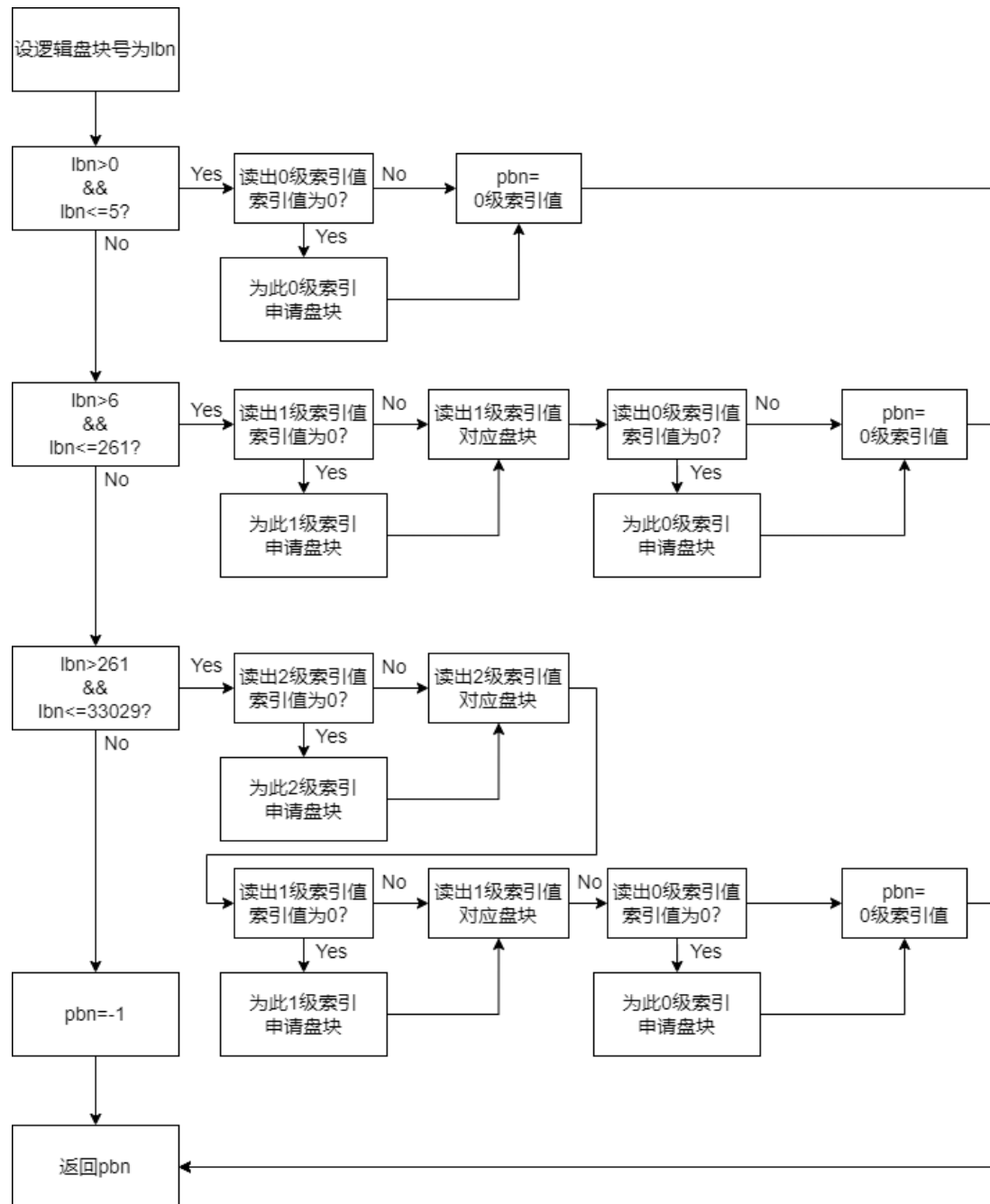
3.5.2 盘块映射算法

盘块映射的算法是将逻辑盘块号 (lbn, 从 0 开始表示, 0 号逻辑块对应着文件前 512 字节) 转换为物理盘块号 (pbn, 即数据存放在文件卷中的真实地址)。

首先, 我们要判断 lbn 位于哪一级。如果 $0 \leq lbn \leq 5$, 则使用 0 级索引; 如果 $6 \leq lbn \leq 261$, 则使用 1 级索引; 如果 $261 < lbn \leq 33029$, 则使用 2 级索引。其余认为超出界限, 返回错误信息 (其中 $261 = 5 + 128 \times 2$, $33029 = 6 + 128 \times 2 + 128 \times 128 \times 2$)。

若使用 0 级索引, 则取出对应的索引号。若为 0, 则说明此位置尚未被使用,

需要申请一个盘块。



3.6 目录管理

3.6.1 目录结构

目录结构与 Unix V6++ 结构类似，但占用大小有所变化。

为了支持长目录名，这里规定：目录项大小为 64B，其中 Inode 号占用 4 个字节，其余为目录名。考虑到 C++char 数组使用尾 0 作为结尾，因此允许的最长目录名为 59 字节。于是一个盘块允许存放 16 个目录项。

目录结构如下：

```
struct DirEntry {  
    int     m_inode;           //目录的inode号  
    char    m_dname[DIR_MAXLEN + 1]; //目录名  
};
```

3.6.2 目录查找算法

在 Ljyux 中，路径名的规则与 Unix/Linux 相同，需要以/分割不同目录，并且用.表示当前目录，用..表示上一级目录。

为了支持.和..，在目录文件中会包含这两项（根目录无..项），这个在创建目录的时候就会自动添加，并且不允许修改。

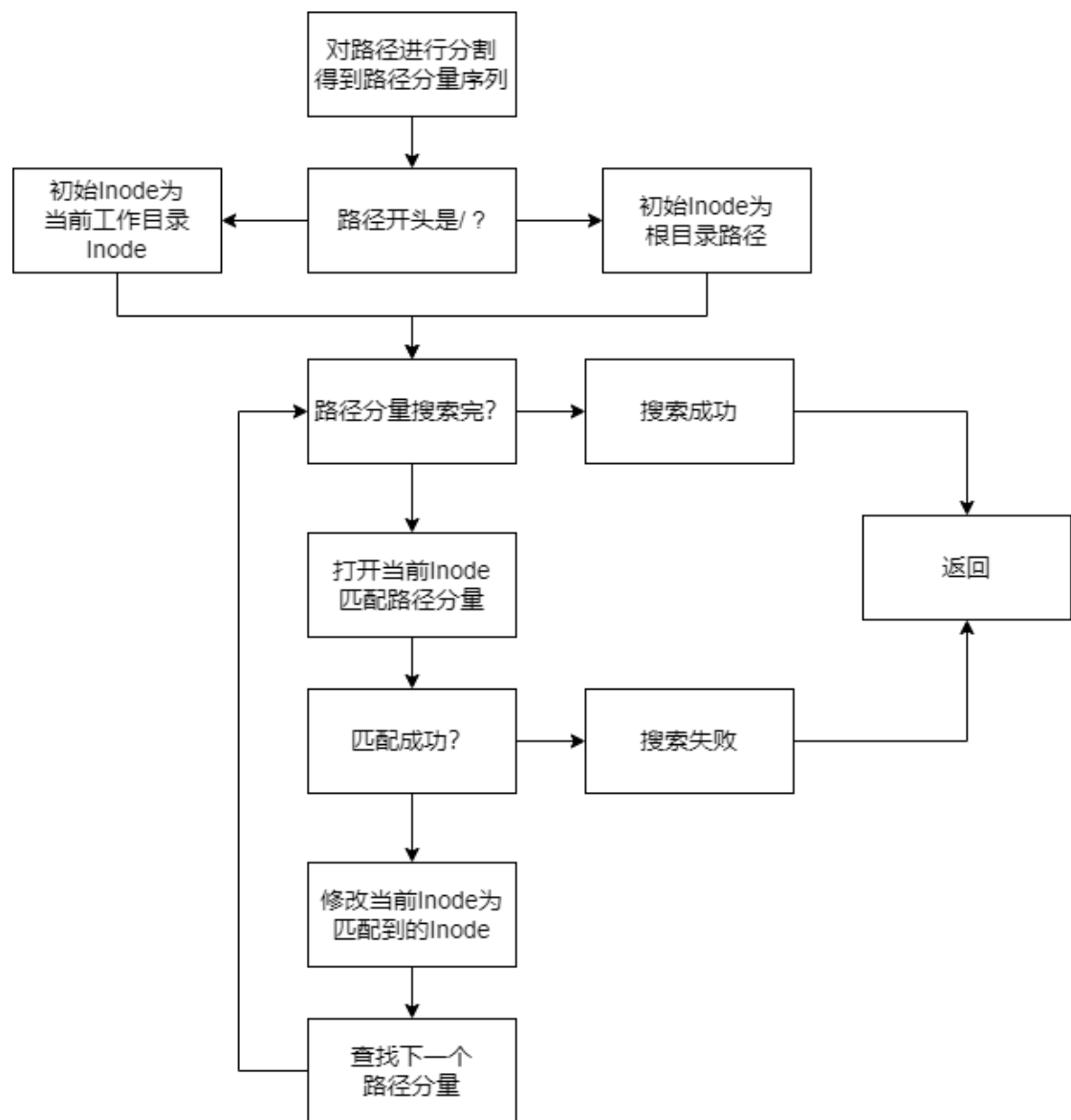
下面给出目录查找算法（注：文件查找也使用此算法）。

目录查找分为查找目录文件和数据文件。两者区别在于查找路径的最后一项时，需要判断 Inode 类型。

在查找的时候，首先我们将路径按照/进行分割，分割得到路径分量序列，每个序列是一个目录项。之后，我们看路径的第一个字符是不是/，若是，则代表路径从根目录开始，是绝对路径，初始 Inode 为根目录的 Inode；否则，是相对路径，初始 Inode 为当前工作目录。

每次查找的时候，打开目录文件，匹配有无与当前路径分量相同的目录项。若有，则打开其对应的 Inode，查找下一个路径目录；否则搜索失败，重复直到搜索失败或遍历完所有的路径分量。注意，搜索到最后一个路径分量的时候，需要判断要找的是目录文件还是数据文件，之前找的都是目录文件。

给出算法流程图如下：

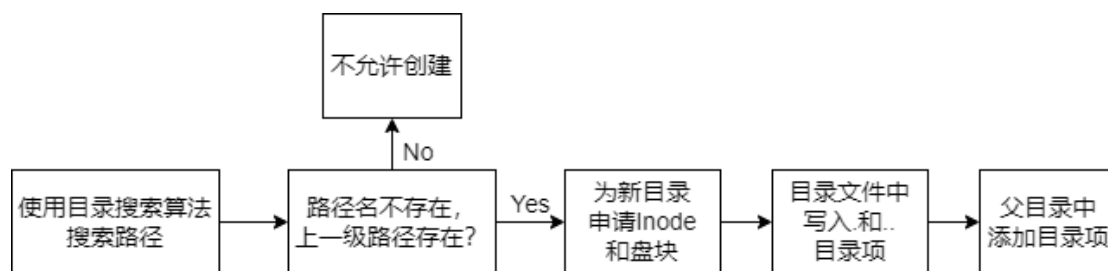


3.6.3 目录创建

目录创建中，需要判断目录路径是否存在，要求目录路径不存在，但上一级目录存在。

之后，便可以创建目录。首先为目录申请 Inode 和盘块，之后向盘块中写入目录项（Inode 号为自己的 Inode 号）和..目录项（Inode 号为父目录的 Inode 号，由目录搜索算法得到）。

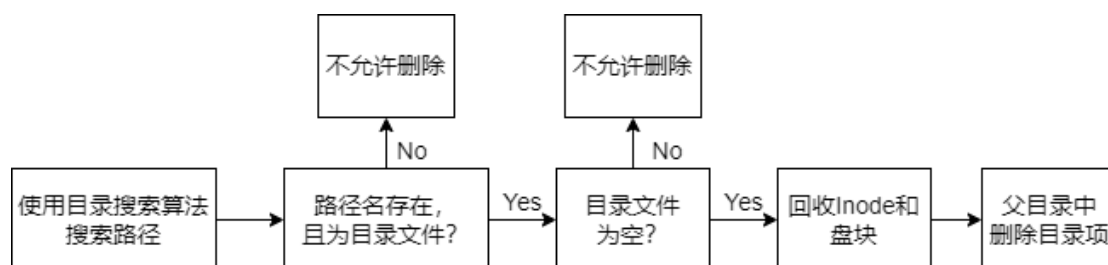
在之后，父目录中添加一目录项，目录名称为路径最后一个路径分量；目录 Inode 号为子目录的 Inode 号。



3.6.4 目录删除

目录删除算法，要求路径存在且为目录文件。

之后，取出此路径对应的 Inode，检查是否为空目录文件，若否，则不允许删除（Ljyux 只允许删除空目录）。若是，则释放其占有的盘块和 Inode。之后，父目录中删除子目录项。

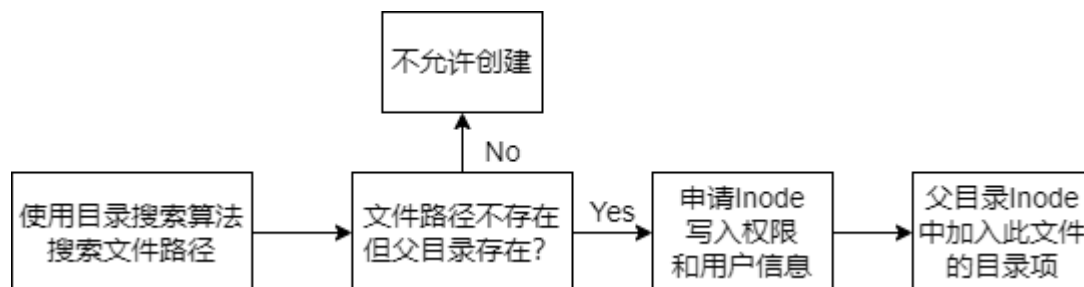


3.7 文件管理

3.7.1 文件创建

创建文件时，仍要搜索文件路径是否存在，要求路径不存在（不支持重复创建，大小写敏感），但要求上一级目录存在。

若满足以上条件，则可以创建。创建时要设置权限和文件持有者的 uid 和 gid。之后在父目录中添加此文件的目录项即可。



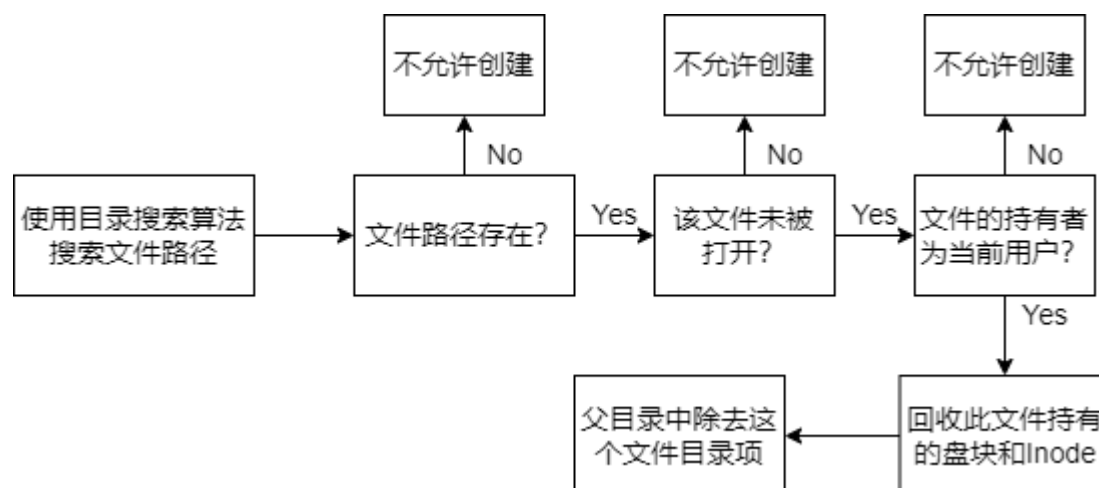
3.7.2 文件删除

删除文件时，要搜索文件路径，要求文件路径存在。

若存在，则检查是否有其他用户打开此文件（检查 Inode 的 i_count 是否为

0), 若无人打开, 再检查当前用户是否有读取写入的权限, 若有, 则可以由存储之。

删除时, 首先回收其持有的盘块和 Inode, 之后从父目录中删除此文件对应的目录项。

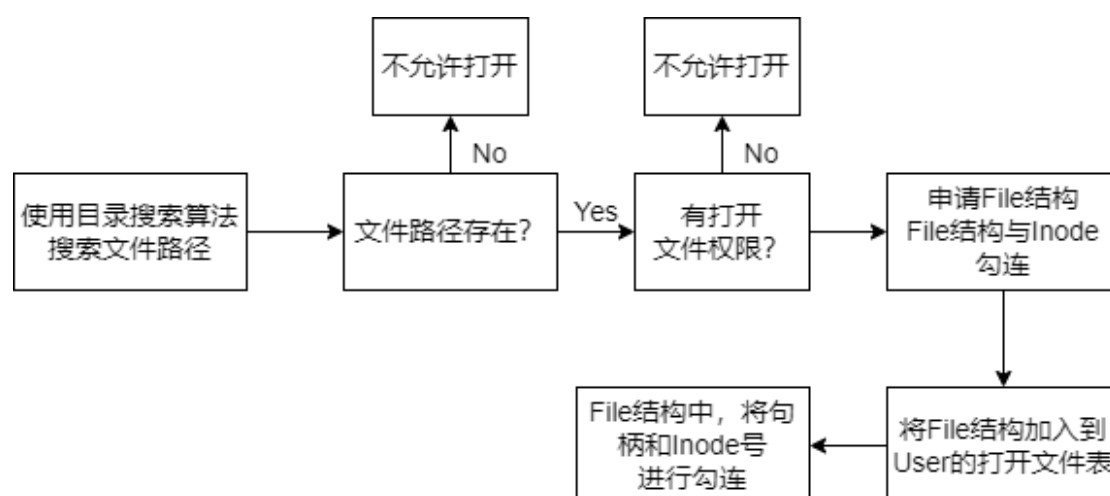


3.7.3 文件打开

打开文件前, 仍然要先进行搜索文件。要求文件路径存在。

之后, 检查有无打开文件的权限 (视打开文件方式决定, 打开文件方式包括只读模式和读写模式)。若具有权限, 则可以打开。

首先, 申请一个打开文件结构 File, 之后, 将这个 File 结构添加到用户打开文件表 OpenFiles 中, 并获得句柄。在 User 结构中, 将句柄和 Inode 号进行勾连, 以便后面读写。

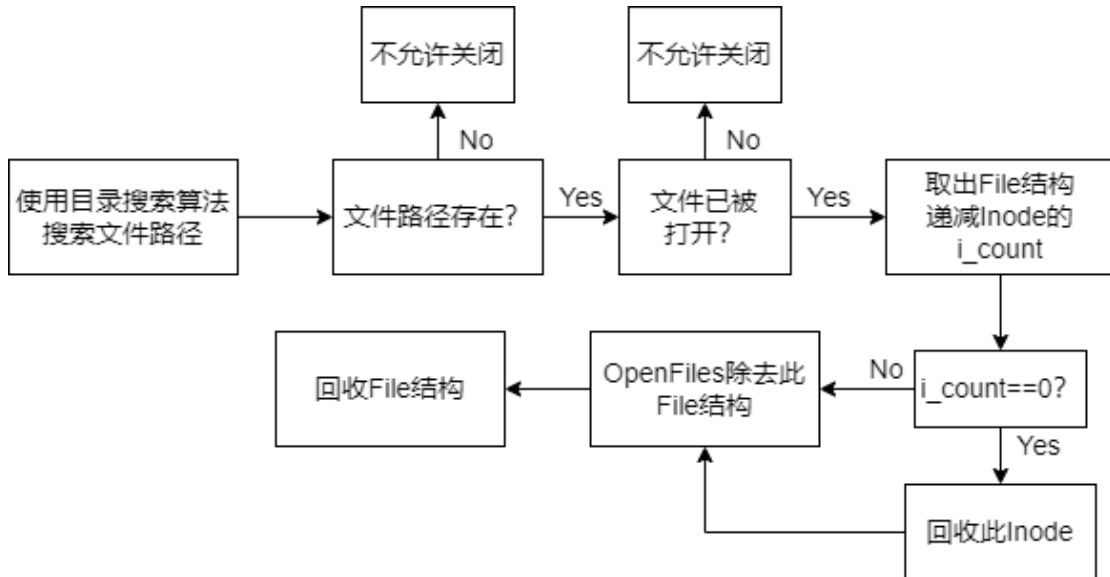


3.7.4 文件关闭

关闭文件前, 仍然要先进行搜索文件。要求文件路径存在。

之后，检查文件是否被打开，若未被打开，则不允许关闭。

从 User 的打开文件表 OpenFiles 中取出 File 结构，递减 File 中的 Inode 的 i_count，若减为 0，则回收此内存 Inode。之后，回收 File 结构，并在 User 的打开文件表中除去此一结构。

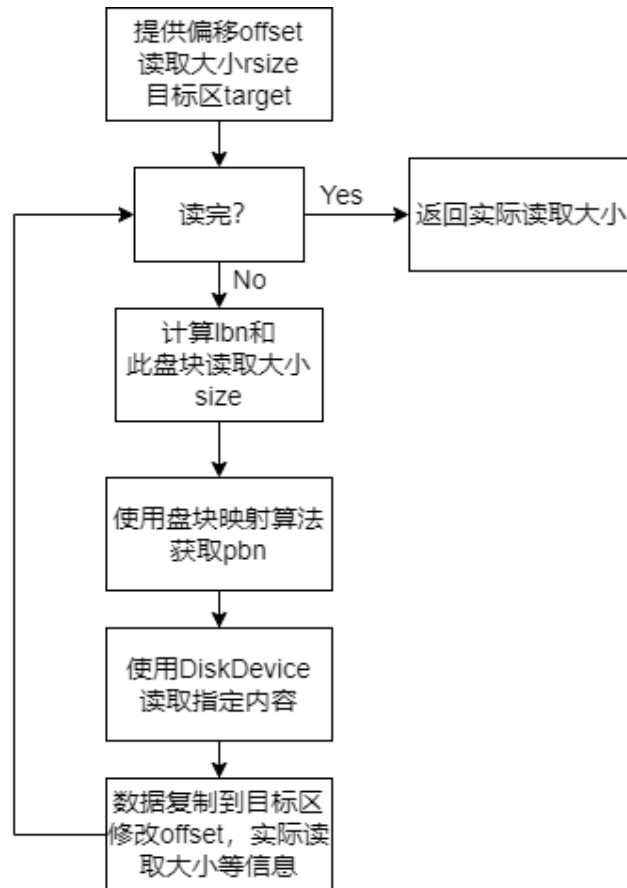


3.7.5 Inode 读

这里主要介绍 Inode 上数据读的方法。需要提供偏移 offset（由 File 结构的 f_offset 提供），读取大小（根据命令参数提供）和目标地址。

读取分为多次。每次最多允许读 1 个盘块。读取前，用位移 offset 计算逻辑盘块 lbn ($lbn = \text{offset} / \text{BLOCK_SIZE}$), 根据盘块映射算法, 计算物理盘块 pbn。

之后，使用 DiskDevice 类提供的读接口从文件卷中读取数据，拷贝到目标缓存块区。修改偏移量、实际读取大小等相关信息。重复直到已经读了待读取大小。最终返回实际读取大小。



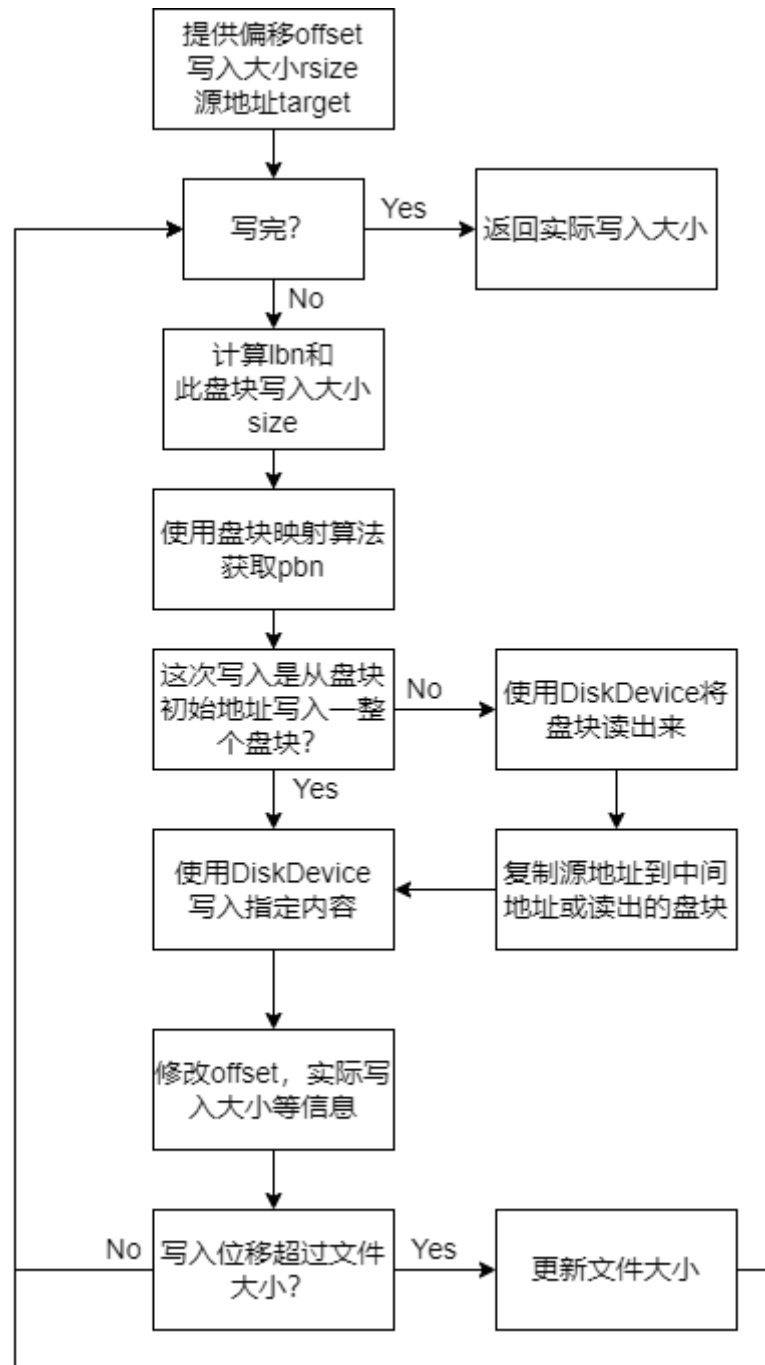
3.7.6 Inode 写

这里主要介绍 Inode 上数据写的方法。需要提供偏移 offset（由 File 结构的 f_offset 提供），写入大小（根据命令参数提供）和源地址。

写入分为多次。每次最多允许写 1 个盘块。写入前，用位移 offset 计算待写入的逻辑盘块 lbn ($lbn = offset / BLOCK_SIZE$)，根据盘块映射算法，计算物理盘块 pbn。

这里需要注意的是，需要判断我们这次写入是否从盘块初始地址开始完整的写入了一个完整的盘块。若不是，则需要先将盘块读出来，以免写入的时候把盘块数据覆盖。

之后，将源地址数据复制到待写入区域，使用 DiskDevice 类提供的写接口向文件卷中写入数据。修改偏移量、实际读取大小等相关信息，若写入位移已超过文件大小，则修改 Inode 的 i_size。重复直到已写了待写入大小。最终返回实际写入大小。



3.8 用户管理

3.8.1 权限检查

权限检查，主要是根据当前用户和文件持有者用户的关系进行比对，再判断文件 Inode 中的权限标志 `i_mode` 中对应位是否为 1。

用户和文件持有者有三种关系：

(1) 用户和文件持有者是用一人（uid 相同）

此时 `i_mode` 用到从右到左第 6-8 位（即 0x400, 0x200, 0x100）。

(2) 用户和文件持有者在同一组（`uid` 不同，但 `gid` 相同）

此时 `i_mode` 用到从右到左第 3-5 位（即 0x040, 0x020, 0x010）。

(3) 用户和文件持有者无特殊关系（`uid`、`gid` 都不同）

此时 `i_mode` 用到从右到左第 2-0 位（即 0x004, 0x002, 0x001）

3.8.2 创建用户

创建用户只允许 `root` 用户执行。

提供用户名和密码后，检查用户名是否重复，若未重复，且用户数量未达到上限，则可以创建。创建时，在 `UserTable` 中申请一个 `User` 结构，填入用户名、密码。`uid` 和 `gid` 在分配 `User` 结构的时候就已经设定，不需要改动。

3.8.3 删除用户

删除用户同样允许 `root` 用户执行，不允许删除 `root` 用户。

提供待删除用户，且确实存在后，直接在 `UserTable` 表中回收这一 `User` 结构即可。

（注：删除用户的时候保留其创建的文件，后续新用户会继承他的 `id` 号，则也会继承他的文件）。

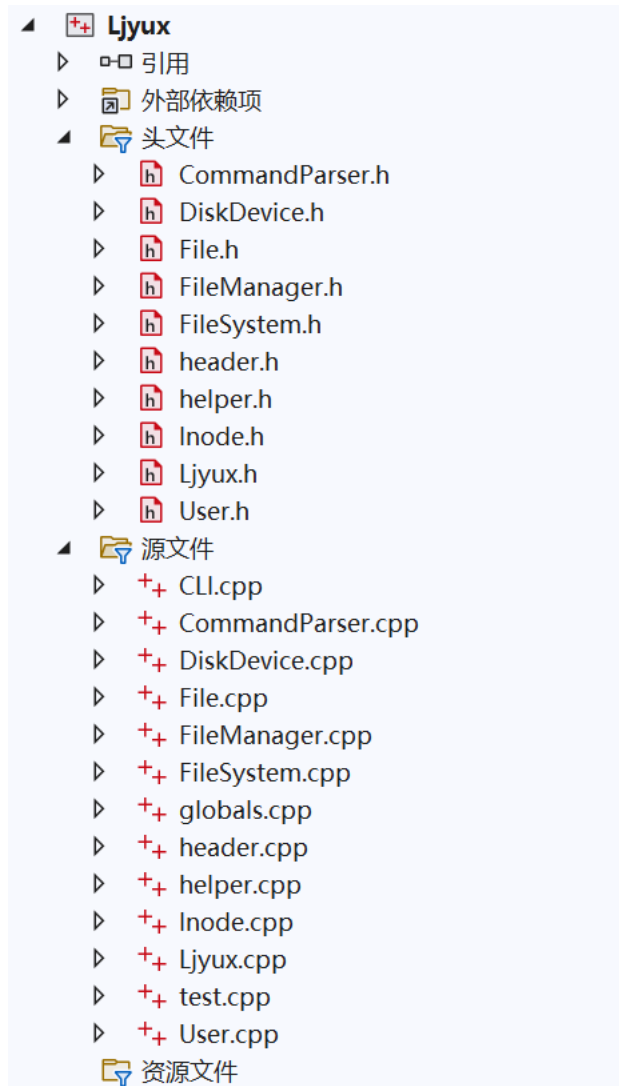
3.8.4 修改分组

修改分组同样只允许 `root` 用户执行。

用用户名取出 `User` 结构后，修改 `gid` 即可。

3.9 项目结构

下图是 `Ljyux` 工程项目结构



下面将介绍各文件的功能。

头文件：

文件	功能
CommandParser.h	命令解析模块函数定义
DiskDevice.h	文件卷读写模块、高速缓存模块定义 (文件读写接口以模板函数的形式实现，根据 C++ 特性这两个函数实现在此头文件中)
File.h	文件打开结构、文件打开结构表、用户打开文件表的定义
FileManager.h	FileManager 结构的定义
FileSystem.h	FileSystem 以及若干磁盘数据结构的定义

header.h	所有其他头文件共同使用部分，包括标准 C/C++库，宏定义, 共用辅助函数等
helper.h	帮助命令部分用到的函数定义
Inode.h	内存 Inode, Inode 表定义
Ljyux.h	内核模块 Ljyux 的定义
User.h	用户结构 User 和用户表 UserTable 的定义

源文件：

CLI.cpp	命令行接口，包含 main 函数
CommandParser.cpp	CommandParser.h 声明的实现
DiskDevice.cpp	DiskDevice.h 声明的实现
File.cpp	File.h 声明的实现
FileManager.cpp	FileManager.h 声明的实现
FileSystem.cpp	FileSystem.h 声明的实现
globals.cpp	全局变量/对象指针的声明
header.cpp	header.h 声明的实现
helper.cpp	helper.h 声明的实现
Inode.cpp	Inode.h 声明的实现
Ljyux.cpp	Ljyux.h 声明的实现
test.cpp	测试用
User.cpp	User.h 声明的实现

四、运行结果分析

4.1 逐条命令简单测试

以下对每条命令进行简单测试。

打开程序，输入用户名 root 和密码 root123（系统默认）。

先不进行默认初始化。

```
D:\data\C++\Ljyux\x64\Debug\Ljyux.exe
Ljyux OS
Version:1.0.0
by: Lin Jiayi, 2019CS , Tongji University

请输入用户名: root
请输入密码: root123
是否要对文件系统进行默认初始化(对应课设主程序部分)? [y/n]
n

输入help命令以获得提示
/ > _
```

使用 help 命令，查看有哪些可用命令。

```
/ > help
可用指令如下:
mkfs : 格式化文件卷
mkdir : 创建目录
rmdir : 删除目录
ls : 展示目录信息
cd : 更改当前目录
mkfile : 创建文件
rmfile : 删除文件
fopen : 打开文件
fclose : 关闭文件
fread : 读取文件
fwrite : 写入文件
fseek : 移动文件读写指针
cat : 输出文件
load : 从宿主机读取文件到Ljyux中
dump : 从Ljyux写出文件到宿主机中
whoami : 查看当前用户
adduser : 添加用户
deluser : 删除用户
su : 更换用户
setgroup : 为用户设置分组
lsinode : 展示inode信息
lsblk : 展示盘块
help : 展示帮助信息
lsuser : 展示用户
chmod : 更改文件权限
setmod : 设置文件权限
/ >
```

4.1.1 格式化命令

输入命令 mkfs，进行格式化。

```
/ > mkfs  
格式化完毕!  
/ > _
```

4.1.2 目录相关命令

首先，在根目录下创建/bin 目录。

```
/ > mkdir /bin  
目录 /bin 创建成功!  
/ > _
```

使用 ls 命令，展示根目录。

```
/ > ls /  
.  
bin
```

移动到 bin 目录下。

```
/ > cd bin  
/bin > ls .  
.  
..
```

返回。

```
/bin > cd ..  
/ > ls .  
.  
bin
```

删除 bin 目录。

```
/ > rmdir /bin  
删除目录 /bin 成功!  
/ > ls .  
.  
/ > _
```

再创建/home 目录，使用 exit 命令退出，再进入，发现目录保留。

```
/ > mkdir /home  
目录 /home 创建成功!  
/ > exit
```

```
D:\data\C++\Ljyux\x64\Debug\Ljyux.exe (进程 26668) 已退出，代码为 0。
```

```
Ljyux OS
Version:1.0.0
by: Lin Jiayi, 2019CS , Tongji University

请输入用户名: root
请输入密码: root123
是否要对文件系统进行默认初始化(对应课设主程序部分)? [y/n]
n

输入help命令以获得提示
/ > ls .
. home
/ >
```

4.1.3 文件创建相关

在上一问的基础上，在 home 目录下创建文件/home/file.txt。

```
/ > mkfile /home/file.txt
/home/file.txt 创建成功! 权限为777
```

默认权限即为 777，即所有人拥有读写执行权限。

使用 ls 命令，加入-l 以展示细节。可以看到 file.txt 与目录项的 filemode 并不一样，第一位是 f 表示数据文件，d 表示目录文件。

```
/ > ls /home -l
filemode          name          size          owner_uid
drwxrwxrwx        .              3              0
drwxrwxrwx        ..             2              0
frwxrwxrwx        file.txt       0B             0
```

打开文件，发现打开成功

```
/ > fopen /home/file.txt
/home/file.txt 打开成功, 打开模式为:rw
/ >
```

默认打开方式为读写，这符合预期。

关闭文件。

```
/ > fclose /home/file.txt
文件 /home/file.txt关闭成功!
/ >
```

删除文件，发现删除成功。

```
/ > rmfile /home/file.txt
删除文件 /home/file.txt 成功!
/ > ls -l /home
filemode          name          size          owner_uid
drwxrwxrwx        .              2              0
drwxrwxrwx        ..             2              0
/ >
```

相关命令的其他用法会在后续给出。

4.1.4 文件读写相关命令

再创建一个文件/home/test，并打开。

```
/ > mkfile /home/test  
/home/test 创建成功! 权限为777  
/ > fopen /home/test  
/home/test 打开成功, 打开模式为:rw
```

写入 helloworld!字符串（注意-s 命令不能输入空格回车等）。

```
/ > fwrite -s helloworld! /home/test  
成功写入11个字节  
/ > _
```

调整文件读写指针到开头，再去读写。

```
/ > fseek /home/test -b 0  
当前文件指针位置: 0  
/ > fread /home/test  
读取到了 11个字节  
helloworld!
```

读取内容符合预期。

使用 cat 命令也可以读取文件，且读取不会改变读写指针。

```
/ > cat /home/test  
helloworld!  
/ > _
```

更多读写细节性内容将在后文给出。

4.1.5 用户相关命令

先查看当前用户信息。

```
/ > whoami  
当前用户 : root  
用户uid : 0  
用户gid : 0  
/ > _
```

显然是 root 用户，因为初始只有 root 用户。

创建普通用户 lin，密码为 1951444。

```
/ > adduser -u lin -p 1951444  
添加用户lin成功! uid: 1 gid: 0  
/ > _
```

可以看到，用户 lin 被创建。

修改用户的 gid，并使用 lsuser 查看。

```
/ > setgroup -u lin -g 1
修改group id成功!
/ > lsuser
user      uid      gid
root      0        0
lin       1        1
```

登录用户 lin

```
/ > su - lin
请输入密码 : 1951444
修改用户成功! 当前用户为: lin
/ > / > whoami
当前用户 : lin
用户uid : 1
用户gid : 0
```

如果是不指明用户，则登录到 root 用户上

```
/ > su -
请输入密码 : root123
修改用户成功! 当前用户为: root
```

删除用户 lin。再用 lsuser 展示，发现的确删除成功。

```
/ > deluser -u lin
删除用户lin成功!
/ > lsuser
user      uid      gid
root      0        0
```

4.1.6 其他命令

使用 lsblk 展示盘块内容。（主要用于开发阶段的 debug）


```

/ > lsblk 0
=====
offset      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0x00000000  00 00 00 00 72 6F 6F 74 00 00 00 00 00 00 00 00
0x00000010  00 00 72 6F 6F 74 31 32 33 00 00 00 00 00 00 00
0x00000020  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000040  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000060  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000080  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000A0  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000C0  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000E0  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000100  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000120  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000140  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000160  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000180  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000001A0  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000001C0  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000001E0  FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
0x000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=====

```

（本想做成 VSCode 中的 Hexdump 的效果，可惜能力有限，未能完全复现）

使用 `lsinode` 展示 Inode 信息。（同样用于调试）

```

/ > lsinode 0
inode_id : 0
inode_size : 2
inode_mode : drwxrwxrwx
/ > █

```

`help` 命令查看某条命令的使用方法

```

/ > help fseek
命令名称: fseek
命令格式:
fseek fp (fp为文件路径, 默认回到开头)
fseek fp -b size (fp为文件路径, 从文件开始向后移动size字节)
fseek fp -e size (fp为文件路径, 从文件结尾向前移动size字节)
fseek fp -a size (fp为文件路径, 从当前位置移动size个字节)
命令作用: 移动文件读写指针
命令说明: 必须保证文件已打开

```

exit 命令退出 Ljyux。

```

/ > exit
D:\data\C++\Ljyux\x64\Debug\Ljyux.exe (进程 21712) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .

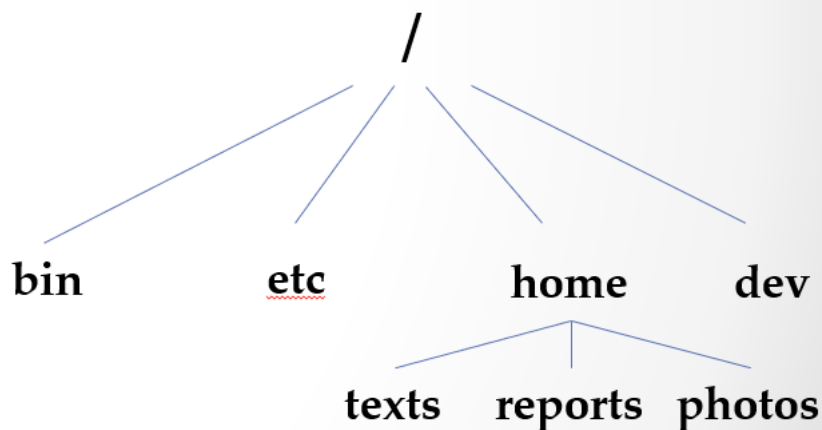
```

(注意: 若不采用 exit 命令退出, 系统不会调用析构函数, 从而导致部分内存中的信息未同步到磁盘中, 下次打开 Ljyux 会产生不可预期的错误。若不慎非正常退出, 请下次打开时格式化或默认初始化)

4.2 主程序测试

4.2.1 主程序要求

- 格式化文件卷;
- 用 mkdir 命令创建子目录, 建立如图所示的目录结构;
- 把你的课设报告, 关于课程设计报告的 ReadMe.txt 和一张图片存进这个文件系统, 分别放在/home/texts , /home/reports 和/home/photos 文件夹;
- 图形界面或者命令行方式, 等待用户输入;
- 根据用户不同的输入, 返回结果。






4.2.2 主程序实现

为了便于测试,我直接将构造如此文件系统用到的命令直接进行硬编程到程序中。使用到的命令如下

```
"mkfs",
"mkdir /bin",
"mkdir /etc",
"mkdir /home",
"mkdir /dev",
"mkdir /home/texts",
"mkdir /home/reports",
"mkdir /home/photos",
"cd /home/texts",
"mkfile report.pdf",
"fopen report.pdf",
"load .\\file\\report.pdf report.pdf",
"fclose report.pdf",
"cd /home/reports",
"mkfile readme.txt",
"fopen readme.txt",
"load .\\file\\readme.txt readme.txt",
"fclose readme.txt",
"cd /home/photos",
"mkfile pic.jpg",
"fopen pic.jpg",
"load .\\file\\pic.jpg pic.jpg",
"fclose pic.jpg",
"cd /"
```

注:测试的时候,我会在 VS 项目同目录下(提交时为可执行文件同目录下)添加一个文件夹 file,内部放有三个文件:report.pdf(pdf 文件,对应课设报告,下面测试的时候我找了个别的 pdf 文件,因为测试的时候报告还没写完),readme.txt(txt 文件,对应 readme.txt,我测试时随便写了几行,因为测试的时候 readme 也没写完),pic.jpg(图片,随便找的)

此电脑 > Data (D:) > data > C++ > Ljyux > file		
名称	类型	大小
 pic.jpg	JPG 文件	291 KB
 readme.txt	文本文档	1 KB
 report.pdf	Microsoft Edge PD...	327 KB

4.2.3 测试情况

登录，选择默认初始化。

```
是否要对文件系统进行默认初始化(对应课设主程序部分)? [y/n]
y
默认初始化开始!
格式化完毕!
目录 /bin 创建成功!
目录 /etc 创建成功!
目录 /home 创建成功!
目录 /dev 创建成功!
目录 /home/texts 创建成功!
目录 /home/reports 创建成功!
目录 /home/photos 创建成功!
report.pdf 创建成功! 权限为777
report.pdf 打开成功, 打开模式为:rw
成功写入 334020 个字节
文件 report.pdf关闭成功!
readme.txt 创建成功! 权限为777
readme.txt 打开成功, 打开模式为:rw
成功写入 66 个字节
文件 readme.txt关闭成功!
pic.jpg 创建成功! 权限为777
pic.jpg 打开成功, 打开模式为:rw
成功写入 297574 个字节
文件 pic.jpg关闭成功!
默认初始化结束!

输入help命令以获得提示
/ >
```

检查:

查看根目录:

```
/ > ls -l .
filemode      name      size      owner_uid
drwxrwxrwx    .         5         0
drwxrwxrwx    bin       2         0
drwxrwxrwx    etc       2         0
drwxrwxrwx    home     5         0
drwxrwxrwx    dev      2         0
```

查看/home:

```
/ > ls -l /home
filemode      name      size      owner_uid
drwxrwxrwx    .         5         0
drwxrwxrwx    ..        5         0
drwxrwxrwx    texts     3         0
drwxrwxrwx    reports   3         0
drwxrwxrwx    photos    3         0
```

查看/home 下三个子目录:

```

/home > ls -l texts
filemode      name      size      owner_uid
drwxrwxrwx    .         3         0
drwxrwxrwx    ..        5         0
frwxrwxrwx    report.pdf 334020B    0

/home > ls -l reports
filemode      name      size      owner_uid
drwxrwxrwx    .         3         0
drwxrwxrwx    ..        5         0
frwxrwxrwx    readme.txt 66B        0

/home > ls -l photos
filemode      name      size      owner_uid
drwxrwxrwx    .         3         0
drwxrwxrwx    ..        5         0
frwxrwxrwx    pic.jpg   297574B    0







```

导出三个文件。

```

/ > fopen /home/reports/readme.txt
/home/reports/readme.txt 打开成功，打开模式为:rw
/ > dump /home/reports/readme.txt .\file\readme2.txt
写出了 66 个字节!
/ > fclose /home/reports/readme.txt
文件 /home/reports/readme.txt关闭成功!
/ > fopen /home/texts/report.pdf
/home/texts/report.pdf 打开成功，打开模式为:rw
/ > dump /home/texts/report.pdf .\file\report2.pdf
写出了 334020 个字节!
/ > fclose /home/texts/report.pdf
文件 /home/texts/report.pdf关闭成功!
/ > fopen /home/photos/pic.jpg
/home/photos/pic.jpg 打开成功，打开模式为:rw
/ > dump /home/photos/pic.jpg .\file\pic2.jpg
写出了 297574 个字节!
/ > fclose /home/photos/pic.jpg
文件 /home/photos/pic.jpg关闭成功!

```

名称	类型	大小
 pic.jpg	JPG 文件	291 KB
 pic2.jpg	JPG 文件	291 KB
 readme.txt	文本文档	1 KB
 readme2.txt	文本文档	1 KB
 report.pdf	Microsoft Edge PD...	327 KB
 report2.pdf	Microsoft Edge PD...	327 KB

进行比对，发现无误，说明存储正确。


```
/ > adduser -u lin -p 1951444
添加用户lin成功! uid: 1 gid: 0
/ > whoami
当前用户 : root
用户uid : 0
用户gid : 0
/ > _
```

之后, 在 root 用户下创建文件 test1, 权限设置为 777, 写入 5 个字符 hello

```
/ > mkfile test1
test1 创建成功! 权限为777
/ > fopen test1
test1 打开成功, 打开模式为:rw
/ > fwrite -s hello test1
成功写入5个字节
/ > _
```

之后, 登录到 lin。打开 test1, 写入三个字符 lin。

```
请输入密码 : 1951444
修改用户成功! 当前用户为: lin
/ > / > fopen test1
test1 打开成功, 打开模式为:rw
/ > fwrite test1 -s lin
成功写入3个字节
/ > _
```

这时候再回到 root 用户, 用 cat 命令输出, 发现字符串变成了 linlo

```
请输入密码 : root123
修改用户成功! 当前用户为: root
/ > / > cat test1
linlo
```

由于 cat 命令并不改变文件读写指针, 因此 root 用户再想 test1 写入字符 whoareyou, 此时文件内容为 linlowhoareyou

```
/ > fwrite test1 -s whoareyou
成功写入9个字节
/ > cat test1
linlowhoareyou
```

切换到用户 lin, 用 fread 读取剩余部分, 发现变成 lowhoareyou。

```
/ > fread test1
读取到了 11个字节
lowhoareyou
/ > _
```

以上操作说明 Ljyux 允许多用户读写。

4.4.2 单用户权限测试

在 root 用户下创建文件 test2, 设置权限为 0x077 (即, 自己无权限, 但其

他人有)

```
/ > mkfile test2 -m 077
test2 创建成功! 权限为77
/ > ls -l .
filemode          name          size          owner_uid
drwxrwxrwx        .              3              0
frwxrwxrwx        test1          14B            0
f---rwxrwx        test2          0B             0
```

尝试打开文件，不行。

```
/ > fopen test2 -rw
无打开权限!
/ > fopen test2 -ro
无打开权限!
/ >
```

修改权限为 0x477 (即仅自己不可写)，发现可以以只读模式打开，不可以以读写模式打开。并且只读模式下不可以写。

```
/ > setmod test2 477
文件权限修改成功!
/ > ls -l .
filemode          name          size          owner_uid
drwxrwxrwx        .              3              0
frwxrwxrwx        test1          14B            0
fr--rwxrwx        test2          0B             0
```

```
/ > fopen test2 -rw
无打开权限!
/ > fopen test2 -ro
test2 打开成功, 打开模式为:ro
/ > fwrite -s hello test2
文件无写权限!
/ > _
```

修改权限为 0x677，可读可写

```
/ > chmod test2 u+w
文件权限修改成功!
/ > fopen test2 -rw
test2 打开成功, 打开模式为:rw
/ > fclose test2
文件 test2关闭成功!
/ > fopen test2 -ro
test2 打开成功, 打开模式为:ro
/ > fclose test2 -rw
文件 test2关闭成功!
```

4.4.3 多用户权限测试

在 4.4.1 的基础上，root 用户下创建文件 test3，权限为 700

```

/ > mkfile test3 -m 700
test3 创建成功! 权限为700
/ > ls -l .
filemode          name          size          owner_uid
drwxrwxrwx        .             4             0
frwxrwxrwx        test1         14B           0
frw-rwxrwx        test2         0B            0
frwx-----        test3         0B            0

```

root 用户下读写模式打开，写入 8 个字符，并关闭。

```

/ > fopen test3 -rw
test3 打开成功, 打开模式为:rw
/ > fwrite test3 -s rootfile
成功写入8个字节
/ > fclose test3
文件 test3关闭成功!
/ >

```

切换成用户 lin，以读写模式打开，无权限。

```

/ > su - lin
请输入密码 : 1951444
修改用户成功! 当前用户为: lin
/ > / >
/ > fopen test3 -rw
无打开权限!
/ >

```

回到 root，更改权限为 0x770

```

/ > setmod test3 770
文件权限修改成功!
/ > ls -l .
filemode          name          size          owner_uid
drwxrwxrwx        .             4             0
frwxrwxrwx        test1         14B           0
frwxrwx---        test2         0B            0
frwxrwx---        test3         15B           0

```

(之前失误把 test2 的权限改成了 770)

再在 lin 下打开，发现成功，并且可读，也可以写，并关闭。

```

/ > / > su - lin
请输入密码 : 1951444
修改用户成功! 当前用户为: lin
/ > / >
/ > fread test3
读取到了 8个字节
rootfile
/ > fwrite -s linfile test3
成功写入7个字节
/ > fclose test3
文件 test3关闭成功!

```

回到 root，将 lin 的 gid 改为 1，

```
/ > setgroup -u lin -g 1
修改group id成功!
/ > lsuser
user      uid      gid
root      0        0
lin       1        1
```

切换到用户 lin，发现又不能打开。

```
/ > su - lin
请输入密码 : 1951444
修改用户成功! 当前用户为: lin
/ > / > fopen test3
无打开权限!
/ >
```

五、 用户使用说明

5.1 操作说明

Ljyux，即 Lin Jiayi's Unix，是 Lin Jiayi 同学的操作系统课设，实现参考了 Unix/Linux 的设计方式，但又有所区别。

打开可执行文件，会提示输出用户名和密码。root 用户的用户名是 root，密码是 root123。

用户名和密码匹配后，会询问是否进行默认初始化，输入 y+回车表示进行默认初始化，其余则不进行。

（这个是课设验收要求，会格式化，请谨慎使用）

输入命令的方式类似于 Linux，即输入一行+回车，鉴于实现方式问题，命令的参数用空格分割，因此所有的参数（包括路径名）不应含有空格回车。

退出程序采用 exit 命令。若未采用 exit 命令退出，会导致系统无法调用析构函数，从而丢失信息。

5.2 可用命令及命令格式说明

5.2.1 可用命令：

可用命令如下：

- mkfs : 格式化文件卷
- mkdir : 创建目录
- rmdir : 删除目录
- ls : 展示目录信息

-
- cd : 更改当前目录
 - mkfile : 创建文件
 - rmfile : 删除文件
 - fopen : 打开文件
 - fclose : 关闭文件
 - fread : 读取文件
 - fwrite : 写入文件
 - fseek : 移动文件读写指针
 - cat : 输出文件
 - load : 从宿主机读取文件到 Ljyux 中
 - dump : 从 Ljyux 写出文件到宿主机中
 - whoami : 查看当前用户
 - adduser : 添加用户
 - deluser : 删除用户
 - su : 更换用户
 - setgroup : 为用户设置分组
 - lsinode : 展示 inode 信息
 - lsblk : 展示盘块
 - help : 展示帮助信息
 - lsuser : 展示用户
 - chmod : 更改文件权限
 - setmod : 设置文件权限
 - exit: 退出 Ljyux

5.2.2 命令说明:

命令名称: adduser

命令格式: adduser [-u name] [-p password] (name 为用户名, password 为密码)

命令作用: 添加用户

命令说明: 用户名和密码最长长度为 13。若不添加 -u -p 项, 则会在后续要

求输入

命令名称: cat

命令格式: cat fp(fp 为文件路径)

命令作用: 输出文件

命令说明: 必须保证文件已打开, 此操作不改变读写指针

命令名称: cd

命令格式: cd fp(fp 为路径名)

命令作用: 更改当前目录

命令说明: 无

命令名称: chmod

命令格式: chmod fp (u|g|v) (+|-) (r|w|x) (fp 为文件路径)

命令作用: 更改文件权限

命令说明: 类似 Linux 的 chmod 命令

命令名称: deluser

命令格式: deluser -u uname (uname 为用户名)

命令作用: 删除用户

命令说明: 必须保证用户存在。删除时保留此用户的文件

命令名称: dump

命令格式: dump src dst (src 为 Ljyux 文件路径, dst 为宿主机文件路径)

命令作用: 从 Ljyux 写出文件到宿主机中

命令说明: 必须保证文件已打开

命令名称: fclose

命令格式: fclose fp(fp 为文件路径)

命令作用：关闭文件

命令说明：必须保证文件已打开

命令名称：fopen

命令格式：fopen fp [-rw|-ro] (fp 为文件路径)

命令作用：打开文件

命令说明：打开文件时需要保证有权限

命令名称：fread

命令格式：fread fp [-n length] (fp 为文件路径，length 为读取长度)

命令作用：读取文件

命令说明：必须保证文件打开从读写指针开始读取 length 个字节到屏幕中，
会修改读写指针

命令名称：fseek

命令格式：

fseek fp (fp 为文件路径，默认回到开头)

fseek fp -b size (fp 为文件路径，从文件开始向后移动 size 字节)

fseek fp -e size (fp 为文件路径，从文件结尾向前移动 size 字节)

fseek fp -a size (fp 为文件路径，从当前位置移动 size 个字节)

命令作用：移动文件读写指针

命令说明：必须保证文件已打开

命令名称：fwrite

命令格式：

fwrite fp -s str [-n size] (fp 为文件路径，str 为写入内容，size 为
写入大小)

fwrite fp -i [-n size] (输入模式，fp 为文件路径，size 为写入大小)

命令作用：写入文件

命令说明:

-s 输入模式不支持字符串含有空格和回车等间隔符

-i 模式下, 以单行#加回车表示输入结束

必须保证文件已打开

命令名称: help

命令格式: help [command] (command 为命令)

命令作用: 展示帮助信息

命令说明: 若不指明命令, 或命令有误, 则会展示所有可用命令

命令名称: load

命令格式: load src dst (src 为宿主机文件路径, dst 为 Ljyux 文件路径)

命令作用: 从宿主机读取文件到 Ljyux 中

命令说明: 必须保证文件已打开, 此操作不改变读写指针, 会覆盖

命令名称: ls

命令格式: ls [-l] fp (fp 是目录路径)

命令作用: 展示目录信息

命令说明: fp 不允许是数据文件

命令名称: lsblk

命令格式: lsblk blk_id (blk_id 是盘块号)

命令作用: 展示盘块

命令说明: 无

命令名称: lsinode

命令格式: lsinode inode_id (inode_id 是 inode 号)

命令作用: 展示 inode 信息

命令说明: 无

命令名称: lsuser

命令格式: lsuser

命令作用: 展示用户

命令说明: 无

命令名称: mkdir

命令格式: mkdir fp (fp 为路径名)

命令作用: 创建目录

命令说明: Ljyux 不允许目录和文件重名

命令名称: mkfile

命令格式: mkfile fp [-m auth] (fp 为文件路径, auth 为文件权限, 16 进制表示)

命令作用: 创建文件

命令说明: 不允许与其他文件重名, 不支持重名覆盖

命令名称: mkfs

命令格式: mkfs

命令作用: 格式化文件卷

命令说明: 格式化后, 当前目录为/, 用户为 root

命令名称: rmdir

命令格式: rmdir fp (fp 为路径名)

命令作用: 删除目录

命令说明: 删除目录时必须保证目录为空

命令名称: rmfile

命令格式: rmfile fp (fp 为文件路径)

命令作用：删除文件

命令说明：删除文件时必须保证文件不被打开

命令名称：setgroup

命令格式：setgroup -u uname -g gid (uname 为用户名，gid 为 id 号，要求在 0-15，若错误则默认为 0)

命令作用：为用户设置分组

命令说明：必须指明用户和 gid

命令名称：setmod

命令格式：setmod fp mode (fp 为文件路径，mode 为文件权限，16 进制)

命令作用：设置文件权限

命令说明：权限设置参考 Unix/Linux 文件权限，若对文件权限不熟悉不建议使用此命令

命令名称：su

命令格式：su [- uname] (uname 为用户名)

命令作用：更换用户

命令说明：若不指明用户，则切换到 root 用户

命令名称：whoami

命令格式：whoami

命令作用：查看当前用户

命令说明：无

六、总结与心得体会

6.1 心得体会

操作系统课设，从开学初我便研究（因为听学长说每届变化不大），于是到了第四周发布任务时，已完成得差不多了。并在第四周结束时完成了报告的编写。

但实际上，操作系统课设我认为难度是比较大的。Unix V6++虽然是一个较为小型的操作系统，但其原理还是很复杂的，在上学期理论课尚且勉强掌握，到这学期其实已有所忘记。因此实际上，我用了将近一个月的时间才完成。

难点主要在于如何上手：直接按照 Unix V6++敲一遍代码，并不可行，因为 Unix V6++的源代码过于复杂，并且有些函数实际上用不到，也看不懂；自己写，又不知道如何组织。再加上遇到了 C++ 的特性问题，很多 Bug 无法解决，以至于我曾两次删库重写。

我认为关键在于要搞明白做什么。文件系统，管理数据文件、目录、用户。层次分为 User 结构 → File 结构 → Inode 结构 → DiskInode 结构 → 磁盘块，每层用指针或索引进行勾连。然后是各种算法，要把 Unix V6++ 的算法搞明白，再自己实现，可以做出修改。最后就是各模块的调用，为了方便，我把常被调用的模块设置为全局对象指针，再由内核模块类 Ljyux 管理全局变量指针，其他模块可以调用之。

总之操作系统课设让我对 Unix 文件系统有了进一步的了解，也提升了我对于 C++ 的掌握程度。

6.2 总结反思

可以进一步优化的地方：

- 优化 Inode 的管理模式，尽量减少内存 Inode 和文件卷的同步次数。
- 优化缓存管理。
- 优化模块调用方式，减少耦合度。

七、参考资料

（由于没有参考重要的文献和论文，仅仅给出参考的来源，就不按照论文参考格式编写了）

- Unix V6++源代码
- 方钰老师在 2021-2022 第一学期操作系统理论课使用的课件和其他资料
- 18 级学姐鞠璇的课设报告（仅仅参考了她的课设的组织方式，实际上实现都不一样）