

课 程 设 计 报 告

设计题目：多用户、多级目录结构文件系统的设计与实现

班 级：计算机2004

组长学号：20202021

组长姓名：张海波

指导教师：冯时

设计时间：2023 年 5 月

设计分工

组长学号及姓名：20202021 张海波

分工：结构体定义 内存块操作 文件操作 内存块初始化 用户
创建删除显示 命令处理

组员学号及姓名：20204807 杜学鸿

分工：结构体定义 目录操作 文件操作 帮助手册 用户登录登
出更改用户组 命令处理

摘 要

本次课设我们设计并实现了一个多用户、多级目录结构文件系统以模拟 Unix 的文件管理系统。该文件系统具有用户管理、文件管理、目录管理等功能，并使用命令行实现输入输出。其中，文件系统的核心功能之一就是文件索引结构的设计和实现。我们采用了内存管理模块、文件索引模块、文件管理模块、用户管理模块、目录管理模块进行功能划分。其中，文件索引模块负责管理文件的索引信息，包括文件的物理结构、目录结构以及空闲盘块的组织。具体来说，我们采用了串联结构表示文件的物理结构，在多级目录结构中采用了类似于 Unix 文件系统的层级目录结构。同时，我们还采用了成组链表法来管理空闲盘块，提高了文件系统对磁盘空间的利用率和性能。通过这些索引结构的设计和实现，我们实现了文件系统支持各种命令操作，包括创建文件、打开文件、关闭文件、删除文件、复制文件、写入文件、重复写入文件、读文件、更改文件指针、更改文件权限、创建目录及文件、删除目录及文件、进入指定目录、回退上一级目录、重命名文件或目录等。

关键词：操作系统，文件系统模拟，Unix，多级索引

目录

摘 要	3
1 概述.....	5
1.1 目的.....	5
1.2 内容简述.....	5
2 课程设计任务及要求.....	7
2.1 设计任务.....	7
2.2 设计要求.....	7
3 算法与数据结构.....	8
3.1 总体设计思想.....	8
3.2 内存块读写模块.....	9
3.2.1 功能.....	9
3.2.2 相关数据结构.....	11
3.2.3 算法流程图.....	11
3.3 目录管理模块.....	12
3.3.1 功能.....	12
3.3.2 数据结构.....	13
3.3.3 算法流程图.....	14
3.4 文件管理模块.....	16
3.4.1 功能.....	16
3.4.2 相关数据结构.....	17
3.4.3 算法流程图.....	18
3.5 用户管理模块.....	21
3.5.1 功能.....	21
3.5.2 相关数据结构.....	21
3.5.3 算法流程图.....	21
3.6 初始化模块.....	22
3.6.1 功能.....	22
3.6.2 程序代码.....	23
3.6.3 算法流程图.....	25
4 程序设计与实现.....	26
4.1 程序总体流程图.....	26
4.2 程序说明.....	27
4.3 实验结果.....	27
4.3.1 初始化.....	27
4.3.2 基本命令进行测试.....	28
4.3.3 用户相关命令.....	31
4.3.4 大文件读写.....	32
5 结论	34
6 参考文献.....	34
7 收获、体会和建议.....	35
张海波	35
杜学鸿	35

1 概述

1.1 目的

在 UNIX 系统中，文件系统是一个至关重要的组成部分。它提供了层次结构的目录和文件，并负责管理系统内文件信息。本次课设旨在编写一个类似于 UNIX 文件系统的程序，以便实现基本的文件操作并掌握文件系统的结构。

1.2 内容简述

使用一个普通的大文件（该课设内使用 myDisk.img）来模拟UNIX 文件系统 一个文件卷（把一个大文件当一张磁盘用）。

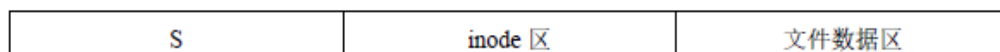


图 1 文件卷结构

磁盘以块为基础单位，每块大小为 512 字节，共设置 100000 块，总计 48.8MB。

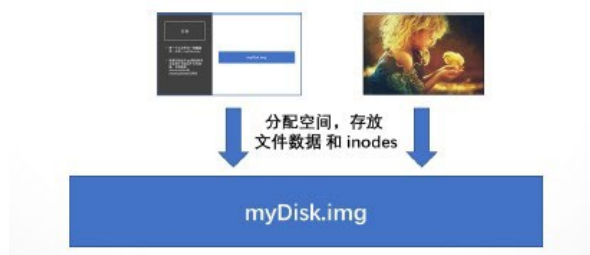


图 2 使用文件卷模拟逻辑磁盘

(1) 磁盘文件结构

- Superblock 块：存储当前磁盘可用块数量和磁盘块总量。并且以成组连接法管理所有空闲 block。
- Inode 块：记录文件或目录的信息，如文件大小、权限、时间戳、索引表等。
- block 块：存储用户、文件、目录信息

(2) 文件目录结构

存储子目录 Inode 号和子目录文件名

(3) 文件结构

记录文件的 inode 编号、文件的读写指针位置、文件打开用户

(4) 功能简介

`format` : 格式化文件卷

`ls` : 列出当前目录

`mkdir` : 创建目录

`creat` : 新建文件

`open` : 打开文件

`close` : 关闭文件

`read` : 读文件

`write` : 写文件

`fseek` : 更改文件读写起始位置

(5) 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

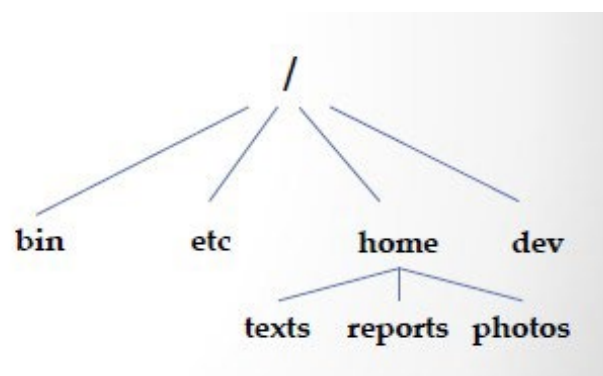


图 3 文件目录结构

2 课程设计任务及要求

2.1 设计任务

在下列内容中任选其一：

- 1、多用户、多级目录结构文件系统的设计与实现；
- 2、WDM 驱动程序开发；
- 3、存储管理系统的实现，主要包括虚拟存储管理调页、缺页统计等；
- 4、进程管理系统的实现，包括进程的创建、调度、通信、撤消等功能；
- 5、自选一个感兴趣的与操作系统有关的问题加以实现，要求难度相当。

2.2 设计要求

- 1、在深入理解操作系统基本原理的基础上，对于选定的题目，以小组为单位，先确定设计方案；
- 2、设计系统的数据结构和程序结构，设计每个模块的处理流程。要求设计合理；
- 3、编程序实现系统，要求实现可视化的运行界面，界面应清楚地反映出系统的运行结果；
- 4、确定测试方案，选择测试用例，对系统进行测试；
- 5、运行系统并要通过验收，讲解运行结果，说明系统的特色和创新之处，并回答指导教师的提问；
- 6、提交课程设计报告。

3 算法与数据结构

3.1 总体设计思想

在本文件系统中，采用数据块的方式来分配内存。内存中使用循环链表等数据结构，而外部存储则采用成组链接法进行具体的分配。该文件系统还提供了登录和登出功能。在登录时，系统会验证用户账户并加载外部存储文件，而登出后则需要通过相关函数来保存文件。此外，为了方便管理目录和文件信息，链表结构被广泛应用。其中链表形式主要是通过数组模拟实现的。总的来说，本系统是基于一个大型普通文件来模拟 UNIX 文件系统的运行机制。

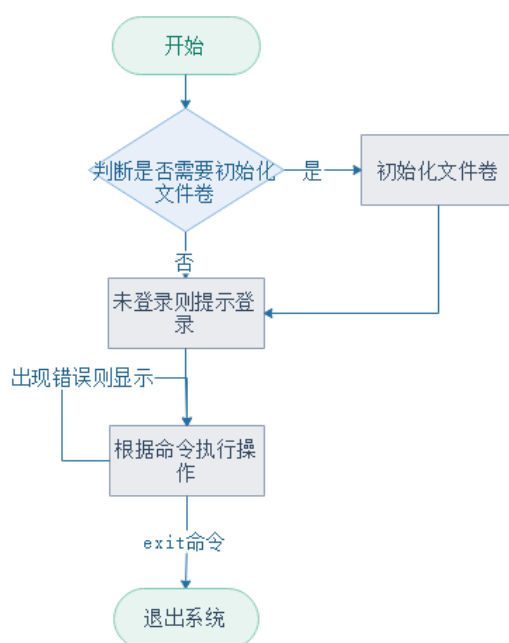


图 4: 整体流程图

整个多用户文件系统可以划分为以下几个模块：

- 顶层管理模块：主要负责处理用户输入信息、将用户输入命令转化为内核操作、反馈信息输出和错误反馈等。
- 内存管理模块：主要负责初始化、申请和释放 SuperBlock、Inode 和 Block。该模块提供读取和写入 SuperBlock、Inode 和 Block 块的接口，并提供申请和释放 Block 块、以及 Inode 中逻辑块号和物理块号映射关系转换的接口。
- 文件管理模块：主要负责文件相关的一系列操作，包括创建文件、删除文件、显示文件列表、打开文件、关闭文件、写文件、读文件、更改文件指针、更改文件权限等操作的相关接口。

- 用户管理模块：主要负责用户相关的一系列操作，包括用户登录、用户登出、创建用户、删除用户、获取当前登录用户信息、更改用户所属组、输出用户列表信息等操作的接口。
- 目录管理模块：主要负责文件目录相关的一系列操作，包括创建目录、删除目录、打开目录、获取当前目录等操作的相关接口。

具体可实现的功能如下：

- 多用户下文件的创建、删除、打开、关闭、读、写、读写指针移动、更改文件权限
- 显示当前打开文件列表
- 多用户下目录的创建、删除、更改及一个目录下文件和子目录的显示
- 用户管理，包括创建、删除、登录、退出、更改所属组、显示当前用户信息和显示所有用户信息
- 格式化文件卷
- 退出文件系统
- 帮助文档

3.2 内存块读写模块

3.2.1 功能

内存块读写，分配一个 Block 块，回收一个 Block 块，Inode 中逻辑块号与物理块号的映射

(1) 内存块读写：在文件操作过程中，需要经常对 SuperBlock 块、Inode 块、User 表等部分进行读写。因此，编写了 Read_SuperBlock、Write_SuperBlock、Read_InodeBitMap、Write_InodeBitMap、Read_User、Write_User、Read_Inode 和 Write_Inode 几个函数对内存进行读写。

(2) 分配一个 Block 块：空闲盘块采用成组链接法进行管理。详细流程如图 6 所示，并在图 7 中给出了流程图。

(3) 回收一个 Block 块：回收 Block 块为分配 Block 块的逆过程，两者流程相似。

```

//如果该空闲块已满，需要使 superbloc 指向一个新的空块
if ( superbloc.s_nfree == FREE_BLOCK_GROUP_NUM) {
    fd.open(DISK_NAME,ios :: out | ios :: in | ios :: binary) ;
    fd.seekg ((BLOCK_POSITION + block_num) BLOCK_SIZE, ios::beg) ;
    fd.write ((char)&superblock.s_free, sizeof (superblock.s_free)) ;
    fd.close () ;
    superbloc.s_free [0] = block_num ;
    superbloc.s_nfree = 0;
    superbloc.s_fblocknum++;
}
else{
    superbloc.s_nfree++;
    superbloc.s_free [superblock.s_nfree] = block_num;
}

```

(4) Inode 中逻辑块号与物理块号的映射

- 小型文件的索引结构：对于小型文件，使用文件索引表中的 `i_addr[0]-i_addr[5]` 共 6 项作为直接索引表。这相当于限制了小型文件的长度范围在 0-6 个数据块之间，每个盘块大小等于 512 字节。其中，文件逻辑块号 `n` 对应的物理块号记录在 `i_addr[n]` 中。例如，`i_addr[2]` 的值为 2058 表示该文件的第 3 块数据（偏移量 1024-1535 字节）占据的物理盘块编号为 2058。
- 大型文件的索引结构：当文件的长度超出小型文件的限制时，就需要采用大型文件的索引结构。除了使用基本索引表中的 `i_addr[0]-i_addr[5]` 记录该文件前 6 个物理块号之外，还要用到 `i_addr[6]`、`i_addr[7]` 这两项。每一项指向一个含有物理块号明细的一次间接索引表块，每个一次间接索引表块可以容纳 $(512/4)-128$ 个物理块号，可为 128 个文件的逻辑块和物理块建立对应关系。因此，大型文件的长度范围是 $7*(6+128*2)$ 个数据块。如果内核需要通过一次间接索引表块访问数据，则需先将一次间接索引表块读入内存，从中找到逻辑块号对应的物理块号，然后再读取该物理块上的文件数据。
- 巨型文件的索引结构：如果文件长度比大型文件还要大，则除了用到基本索引表中的 `i_addr[0]-i_addr[5]` 以及 `i_addr[6]`、`i_addr[7]` 来记录该文件前 1282+6 个物理块号之外，还要使用 `i_addr[8]`、`i_addr[9]` 来记录二次间接索引表块的物理块号。二次间接索引表块的作用类似于一次间接索引表，表中的每一项都记录着它所指向的一个一次间接索引表块的物理块号，然后再由该一次间接索引表块中的各项记录文件数据的物理块号。每张二次间接索引表块最多可记录 128 个一次间接索引表的物理块号，因此巨型文件的长

度范围为 1282+7 至 1281282+128*2+6。

3.2.2 相关数据结构

```
// SuperBlock结构体
struct SuperBlock
{
    unsigned short s_inodenum;           // Inode总数
    unsigned short s_finodenum;          // 空闲Inode数
    unsigned short s_blocknum;           // Block总数
    unsigned short s_fblocknum;          // 空闲Block数
    unsigned int s_nfree;                 // 直接管理的空闲块数
    unsigned int s_free[FREE_BLOCK_GROUP_NUM]; // 空闲块索引表
};

// Directory结构体
struct Directory
{
    unsigned int d_inodenum[SUBDIRECTORY_NUM]; // 子目录Inode号
    char d_filename[SUBDIRECTORY_NUM][FILE_NAME_MAX]; // 子目录文件名
};

// User结构体
struct User
{
    unsigned short u_id[USER_NUM];           // 用户id
    unsigned short u_gid[USER_NUM];          // 用户所在组id
    char u_name[USER_NUM][USER_NAME_MAX];    // 用户名
    char u_password[USER_NUM][USER_PASSWORD_MAX]; // 用户密码
};
```

3.2.3 算法流程图

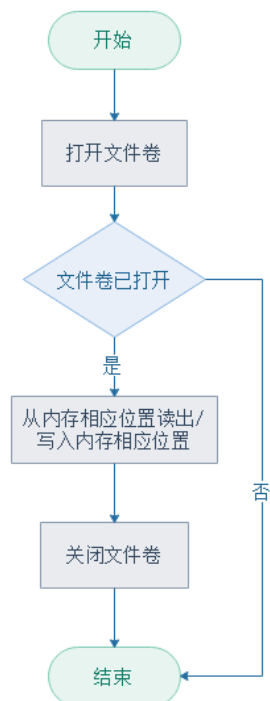


图5：内存块读写

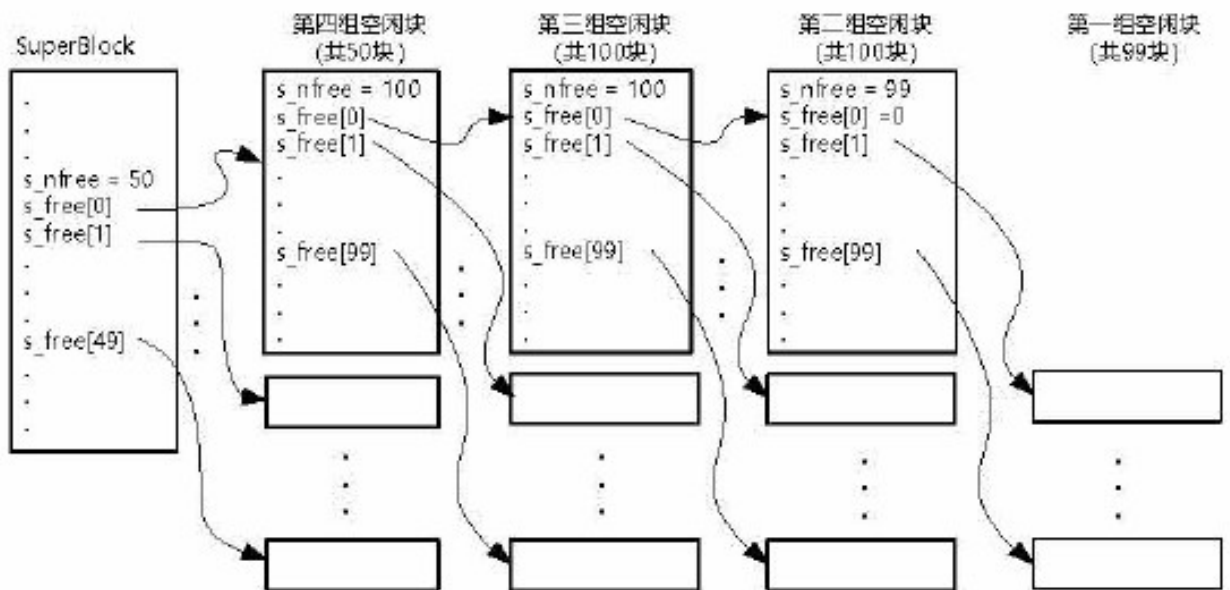


图6：成组链接法

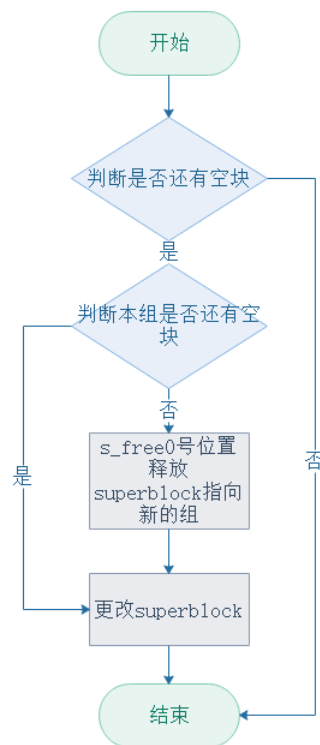


图7:分配一个Block 块

3.3 目录管理模块

3.3.1 功能

实现目录操作需要以下几个函数：

- 创建目录
- 删除目录
- 打开目录

- 获取当前目录

(1)创建目录： 创建目录需要满足以下条件：

目录名合法

存在空闲的 block 和 inode

当前目录不存在同名目录

当前目录的文件数量未达到限制

(2)删除目录： 删除目录需要满足以下条件：

目录名合法

目录为空

(3)打开目录：

打开一个目录需要通过逐级访问目录树中的各个节点来寻找目标目录。目录路径使用“/”进行分割，其中“.”代表当前目录，“..”代表父目录。找到目标目录对应的 inode 号后，即可修改相应的 directory，完成目录的切换。

(4)获取当前目录：

从当前目录向上一层层遍历，使用“/”连接每一层目录的名称，即可获取当前目录。

3.3.2 数据结构

```
//Inode结构体
struct Inode {
    enum INodeMode {
        IFILE = 0x1, //是文件
        IDIRECTORY = 0x2 //是目录
    };
    enum INodePermission { //分为文件主、文件主同组和其他用户
        OWNER_R = 0400,
        OWNER_W = 0200,
        OWNER_E = 0100,
        GROUP_R = 040,
        GROUP_W = 020,
        GROUP_E = 010,
        ELSE_R = 04,
        ELSE_W = 02,
        ELSE_E = 01,
    };
    unsigned int i_addr[NINODE]; //逻辑块号和物理块号转换的索引表
    unsigned int i_size; //文件大小，字节为单位
    unsigned short i_count; //引用计数
    unsigned short i_number; //Inode的编号
    unsigned short i_mode; //文件工作方式信息
    unsigned short i_permission; //文件权限
    unsigned short i_uid; //文件所有者的用户标识
    unsigned short i_gid; //文件所有者的组标识
    time_t i_time; //最后访问时间
};

//Directory结构体
struct Directory {
    unsigned int d_inodenum[SUBDIRECTORY_NUM]; //子目录Inode号
    char d_filename[SUBDIRECTORY_NUM][FILE_NAME_MAX]; //子目录文件名
};
```

3.3.3 算法流程图

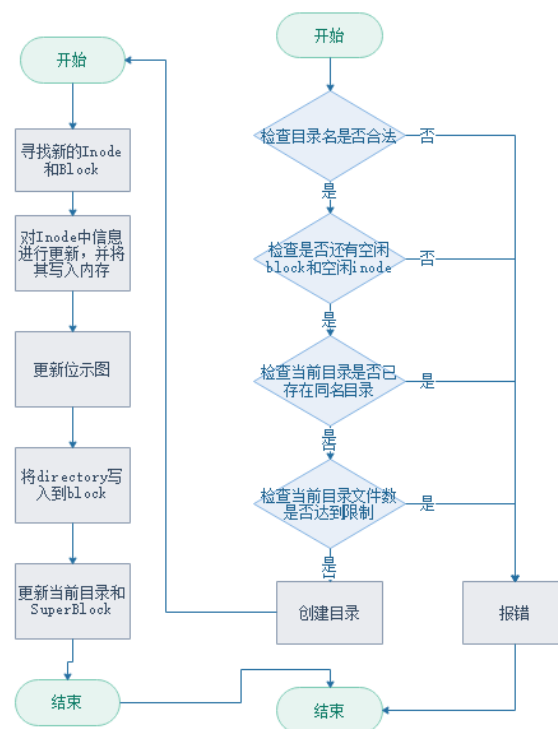


图8: 创建目录流程图

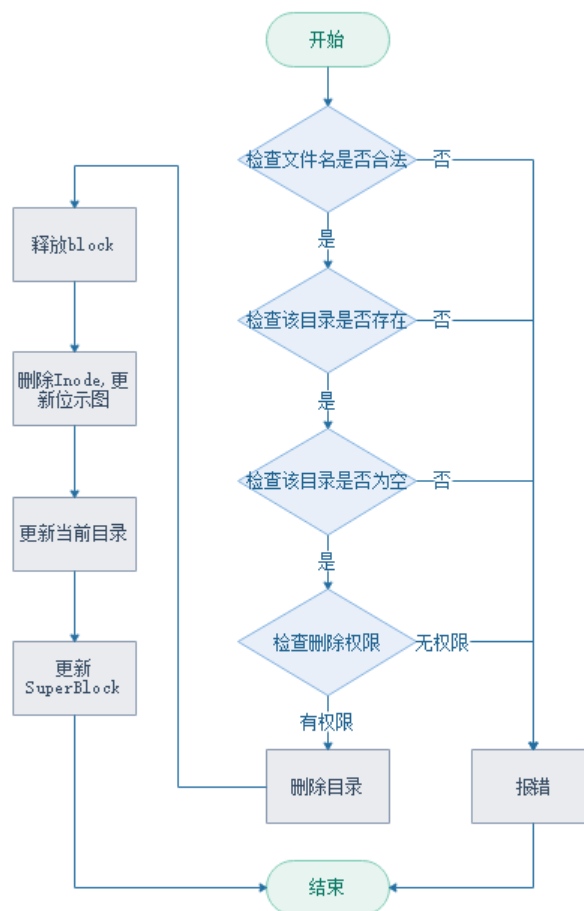


图9: 删除目录流程图

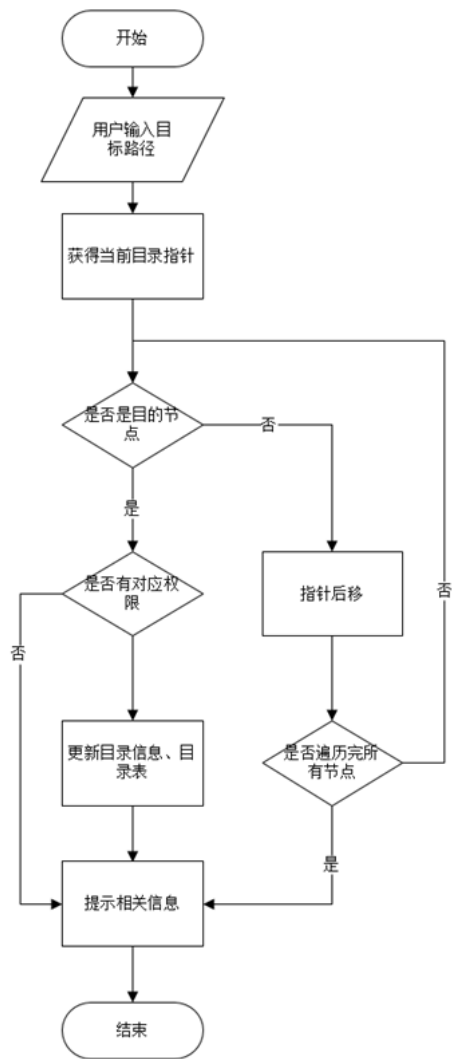


图 10 打开目录

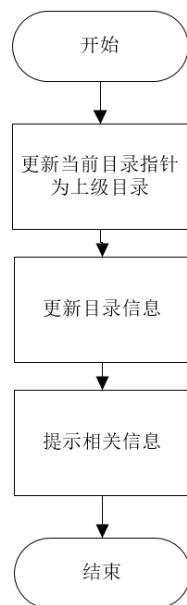


图 11 获取目录信息

3.4 文件管理模块

3.4.1 功能

创建文件, 删除文件, 读文件, 写文件, 更改文件权限, 打开文件, 关闭文件, 更改文件读写位置

(1) 创建文件: 只有在文件名合法、当前目录不存在同名文件、当前目录的文件数量未达到限制并且有空闲的 block 和 inode 时, 才能成功地创建一个新文件。

(2) 删除文件: 要删除一个文件, 必须保证文件名合法, 且该文件未被打开。

(3) 更改文件指针: 只有在当前用户为文件打开用户并具有写权限的情况下, 才能更改 File 结构体中的 `file->f_offset`。

(4) 读文件: 为了读取文件, 必须保证当前用户为文件打开用户并具有写权限。从 `file->f_offset` 指定位置开始, 逐步获取 Block 块, 直到读取文件长度等于 `length` 或读到文件末尾。在字符串末尾添加一个零字节, 并返回读取的字符串长度。

(5) 写文件: 要写入文件, 必须保证当前用户为文件打开用户并具有写权限。

(6) 更改文件权限: 每个文件都有三组权限, 分别为主用户、同组用户和其他用户, 每组权限又分为读、写和执行权限。要修改这些权限, 只需在相应的 Inode 中修改 `i_permission` 属性即可。

(7) 打开文件: 打开文件的前提是文件名合法且存在于当前目录下。如果找到了该文件, 则会创建一个新的 File 结构体, 并将 `f_inodeid` 设置为找到的 `inode_num`, `f_offset` 设置为 0, `f_uid` 设置为打开文件的用户 id, 并返回 file 结构体。

(8) 关闭文件: 关闭文件后, 用户将无法再读取文件内容。此外, 可以释放该文件在内存中占用的空间, 并将其从打开文件表中删除。

3.4.2 相关数据结构

```
//Inode结构体
struct Inode {
    enum InodeMode {
        IFILE = 0x1, //是文件
        IDIRECTORY = 0x2 //是目录
    };
    enum InodePermission { //分为文件主、文件主同组和其他用户
        OWNER_R = 0400,
        OWNER_W = 0200,
        OWNER_E = 0100,
        GROUP_R = 040,
        GROUP_W = 020,
        GROUP_E = 010,
        ELSE_R = 04,
        ELSE_W = 02,
        ELSE_E = 01,
    };
    unsigned int i_addr[NINODE]; //逻辑块号和物理块号转换的索引表
    unsigned int i_size; //文件大小，字节为单位
    unsigned short i_count; //引用计数
    unsigned short i_number; //Inode的编号
    unsigned short i_mode; //文件工作方式信息
    unsigned short i_permission; //文件权限
    unsigned short i_uid; //文件所有者的用户标识
    unsigned short i_gid; //文件所有者的组标识
    time_t i_time; //最后访问时间
};

//Directory结构体
struct Directory {
    unsigned int d_inodenum[SUBDIRECTORY_NUM]; //子目录Inode号
    char d_filename[SUBDIRECTORY_NUM][FILE_NAME_MAX]; //子目录文件名
};
```

```
//User结构体
struct User {
    unsigned short u_id[USER_NUM]; //用户id
    unsigned short u_gid[USER_NUM]; //用户所在组id
    char u_name[USER_NUM][USER_NAME_MAX]; //用户名
    char u_password[USER_NUM][USER_PASSWORD_MAX]; //用户密码
};

//File结构体
struct File {
    unsigned int f_inodeid; //文件的inode编号
    unsigned int f_offset; //文件的读写指针位置
    unsigned int f_uid; //文件打开用户
};
```

3.4.3 算法流程图

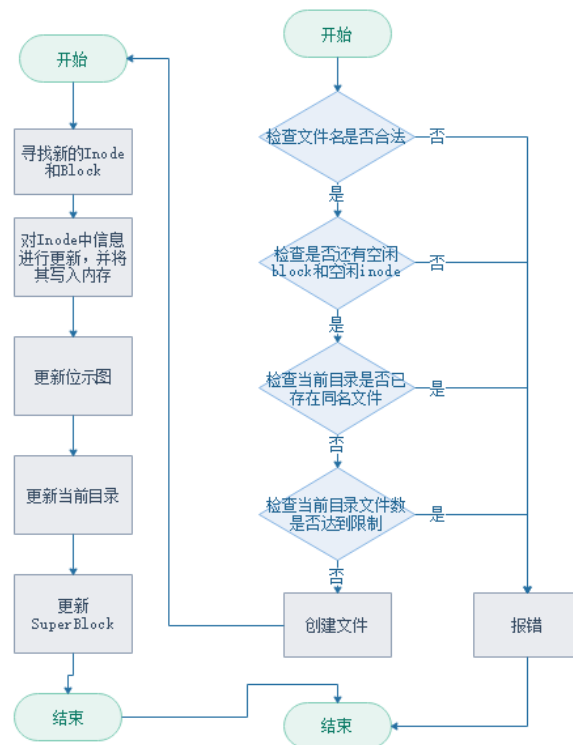


图 12: 创建文件

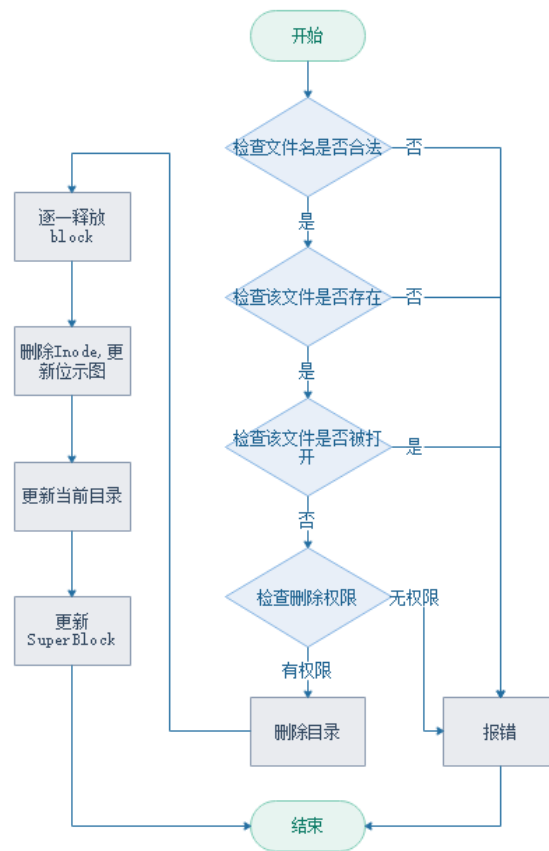


图13: 删除文件

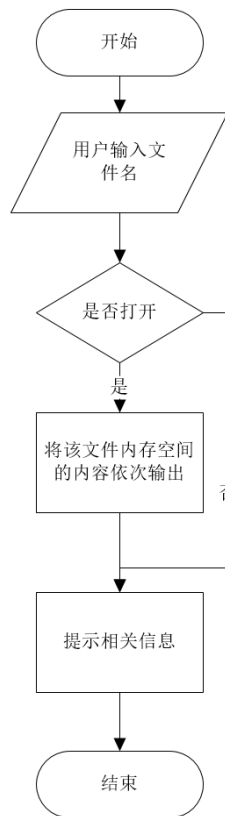


图 14: 读文件

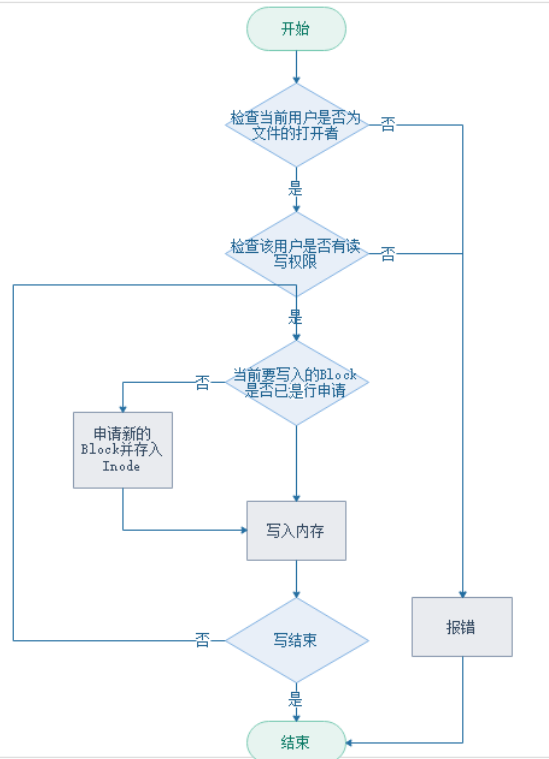


图15: 写文件

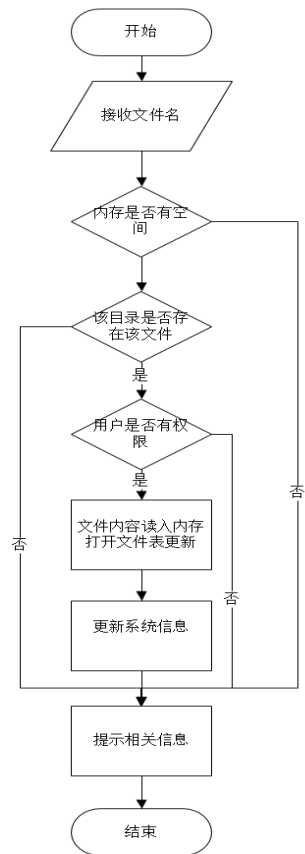


图 16: 打开文件

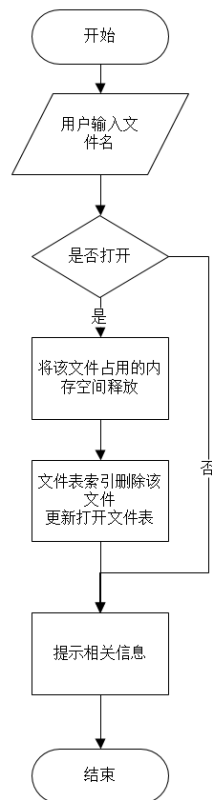


图 17 关闭文件

3.5 用户管理模块

3.5.1 功能

登录用户，退出登录，创建用户，删除用户，改变用户所属组

(1) 登录用户：在检查用户名和密码均正确的情况下，才能完成登录。如果输入的用户名不正确，则提示该用户不存在；如果输入的密码不正确，则提示密码错误。

(2) 退出登录：通过将全局的 `user_id` 改为 `-1`，可以完成用户的退出登录。

(3) 创建用户：只有 `root` 用户可以进行用户的注册。在注册前需要先检查该用户名是否已经存在，若存在则给出报错，否则可以创建一个新用户。

(4) 删除用户：只有 `root` 用户可以进行用户的删除。在删除前需要先检查该用户是否存在，若存在则进行用户的删除，否则给出报错。

(5) 更改用户所属组：只有 `root` 用户可以进行用户所属组的更改。在更改前需要先检查该用户是否存在，若存在则进行用户所属组的更改，否则给出报错。

3.5.2 相关数据结构

```
//User结构体
struct User {
    unsigned short u_id[USER_NUM]; //用户id
    unsigned short u_gid[USER_NUM]; //用户所在组id
    char u_name[USER_NUM][USER_NAME_MAX]; //用户名
    char u_password[USER_NUM][USER_PASSWORD_MAX]; //用户密码
};
```

3.5.3 算法流程图

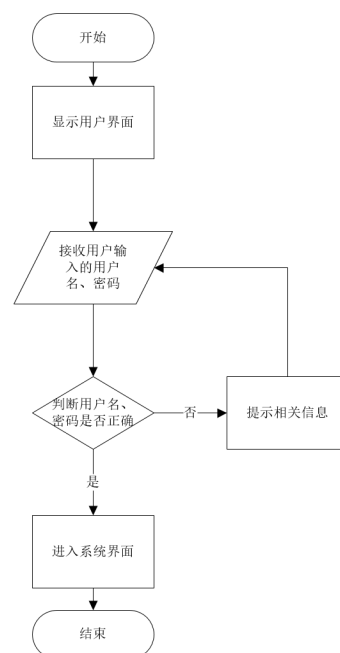


图 18 登录用户

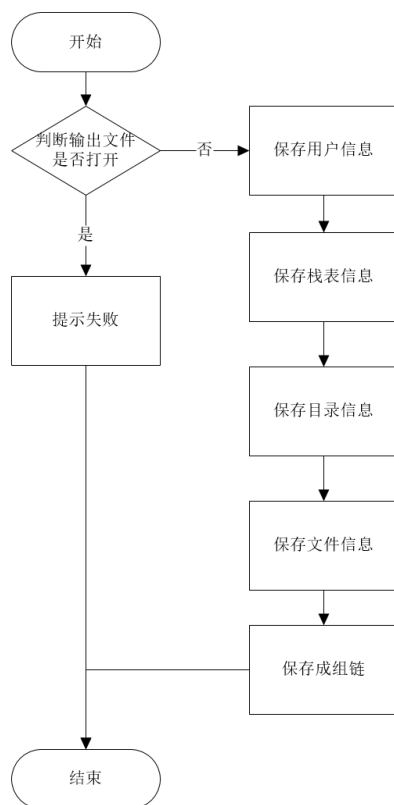


图 19 退出登录

3.6 初始化模块

3.6.1 功能

在未检测到 img 文件情况下会申请在当前目录格式化生成 Img 文件作为磁盘块，具体流程如下：

创建文件卷：通过 fstream 类创建名为 DISK_NAME 的文件卷，并关闭文件。

初始化 SuperBlock：对系统的 SuperBlock 进行初始化，包括总共有多少个 Inode、总块数以及可用的 Block 数，同时也将管理的 Block 空间初始化。最后将 SuperBlock 写入磁盘。

进行成组链接：将所有 Block 按照一定组的规则链接起来，每组包含一定数量的 Block。将成组链接的信息写入磁盘。

初始化位图：将所有 Inode 的位图进行初始化，前两个被设置为 1 以避免误操作。

创建根目录：创建根目录，初始时只有“.”和“..”两个子目录，然后将其写入磁盘。同时在根目录下创建一个管理员账户和一个测试账户用于后续使用。

进入根目录并创建文件和目录：对刚刚创建好的文件系统进行访问，进入根目录

并创建文件和目录，完成初始化的过程。

3.6.2 程序代码

```
// 初始化整个文件系统空间
void Init()
{
    // 在当前目录新建文件作为文件卷
    fstream fd(DISK_NAME, ios::out);
    fd.close();
    fd.open(DISK_NAME, ios::out | ios::in | ios::binary);

    // 如果没有打开文件则输出提示信息并throw错误
    if (!fd.is_open())
    {
        cout << "无法打开文件卷myDisk.img" << endl;
        throw(ERROR_CANT_OPEN_FILE);
    }

    // 对SuperBlock进行初始化
    SuperBlock superblock;
    superblock.s_inodenum = INODE_NUM;
    // 总块数 = Block数 + SuperBlock所占块数 + Inode位图所占块数 + Inode所占块数
    superblock.s_blocknum = BLOCK_NUM + BLOCK_POSITION;
    superblock.s_finodenum = INODE_NUM - 2; // 有两个已经被用
    superblock.s_fblocknum = BLOCK_NUM - 2; // 有两个已经被用
    // 直接管理的Block空间 (0-49)
    for (int i = 0; i < FREE_BLOCK_GROUP_NUM; i++)
        superblock.s_free[i] = FREE_BLOCK_GROUP_NUM - 1 - i;
    superblock.s_nfree = FREE_BLOCK_GROUP_NUM - 1 - 2; // 有两个已经被用
    // 写入内存
    fd.seekg(SUPERBLOCK_POSITION * BLOCK_SIZE, ios::beg);
    fd.write((char *)&superblock, sizeof(superblock));

    // 进行成组链接
    unsigned int stack[FREE_BLOCK_GROUP_NUM];
    for (int i = 2; i <= BLOCK_NUM / FREE_BLOCK_GROUP_NUM; i++)
    {
        for (unsigned j = 0; j < FREE_BLOCK_GROUP_NUM; j++)
        {
            stack[j] = FREE_BLOCK_GROUP_NUM * i - 1 - j;
        }
        if (i == BLOCK_NUM / FREE_BLOCK_GROUP_NUM)
            stack[0] = 0;
        // 写入内存
        if (i != BLOCK_NUM / FREE_BLOCK_GROUP_NUM)
            fd.seekg((BLOCK_POSITION + stack[0] - FREE_BLOCK_GROUP_NUM) * BLOCK_SIZE, ios::beg);
        else
            fd.seekg((BLOCK_POSITION + BLOCK_NUM - FREE_BLOCK_GROUP_NUM - 1) * BLOCK_SIZE, ios::beg);
        fd.write((char *)&stack, sizeof(stack));
    }

    // 初始化位图 (初始化前两个为1, 剩下的在定义时已经被置为0)
    unsigned int inode_bitmap[INODE_NUM] = {0};
    inode_bitmap[0] = 1;
    inode_bitmap[1] = 1;
    // 写入内存
    // 写入内存
    fd.seekg(INODE_BITMAP_POSITION * BLOCK_SIZE, ios::beg);
    fd.write((char *)inode_bitmap, sizeof(unsigned int) * INODE_NUM);

    // 创建根目录
    Inode Inode_root;
    Inode_root.i_number = 0; // Inode的编号
    Inode_root.i_addr[0] = 0; // 对应0号Block
    Inode_root.i_mode = Inode::IDIRECTORY; // 目录
    Inode_root.i_count = 0; // 引用计数
    Inode_root.i_uid = 0; // 管理员
    Inode_root.i_gid = 1; // 文件所有者的组标识
    Inode_root.i_size = 0; // 目录大小为0
    Inode_root.i_time = time(NULL); // 最后访问时间
    Inode_root.i_permission = 0777;
```



```

// 写入内存
fd.seekg(INODE_POSITION * BLOCK_SIZE, ios::beg);
fd.write((char *)&Inode_root, sizeof(Inode_root));
// 0号Block写入Directory
Directory root_directory;
strcpy(root_directory.d_filename[0], "."); // 0是自己
root_directory.d_inodenum[0] = 0;
strcpy(root_directory.d_filename[1], ".."); // 1是父亲（此处还是自己）
root_directory.d_inodenum[1] = 0;
for (int i = 2; i < SUBDIRECTORY_NUM; i++)
{
    root_directory.d_filename[i][0] = '\0';
}
for (int i = 2; i < SUBDIRECTORY_NUM; i++)
{
    root_directory.d_inodenum[i] = -1;
}
fd.seekg(BLOCK_POSITION * BLOCK_SIZE, ios::beg);
fd.write((char *)&root_directory, sizeof(root_directory));

// 创建用户文件
Inode Inode_accounting;
Inode_accounting.i_number = 1; // Inode的编号
Inode_accounting.i_addr[0] = 1; // 对应1号Block
Inode_accounting.i_mode = Inode::IFILE; // 文件
Inode_accounting.i_count = 0; // 引用计数
Inode_accounting.i_permission = 0700; // 管理员可读写
Inode_accounting.i_uid = 0; // 管理员
Inode_accounting.i_gid = 1; // 文件所有者的组标识
Inode_accounting.i_size = 0; // 目录大小为1
Inode_accounting.i_time = time(NULL); // 最后访问时间
// 写入内存
fd.seekg(INODE_POSITION * BLOCK_SIZE + INODE_SIZE, ios::beg);
fd.write((char *)&Inode_accounting, sizeof(Inode_accounting));

```

```

// 创建两个账户
User user;
strcpy(user.u_name[0], "root");
strcpy(user.u_password[0], "root");
strcpy(user.u_name[1], "testUser");
strcpy(user.u_password[1], "testUser");
user.u_id[0] = 0;
user.u_id[1] = 1;
for (int i = 2; i < USER_NUM; i++)
{
    user.u_id[i] = -1;
}
for (int i = 2; i < USER_NUM; i++)
{
    user.u_name[i][0] = '\0';
}
for (int i = 2; i < USER_NUM; i++)
{
    user.u_password[i][0] = '\0';
}
user.u_gid[0] = 1;
user.u_gid[1] = 2;
// 写入内存
fd.seekg(BLOCK_POSITION * BLOCK_SIZE + Inode_accounting.i_addr[0] * BLOCK_SIZE, ios::beg);
fd.write((char *)&user, sizeof(user));

```



```

// 进入根目录
fd.seekg(BLOCK_POSITION * BLOCK_SIZE, ios::beg);
fd.read((char *)&directory, sizeof(directory));

fd.close();

// 以root用户创建文件目录
user_id = 0;
Create_Directory("bin");
Create_Directory("etc");
Create_Directory("home");
Create_Directory("dev");
Open_Directory("./home");
Create_Directory("texts");
Create_Directory("reports");
Create_Directory("photos");
Open_Directory("./texts");
Create_File("test.txt");
}

```

3.6.3 算法流程图

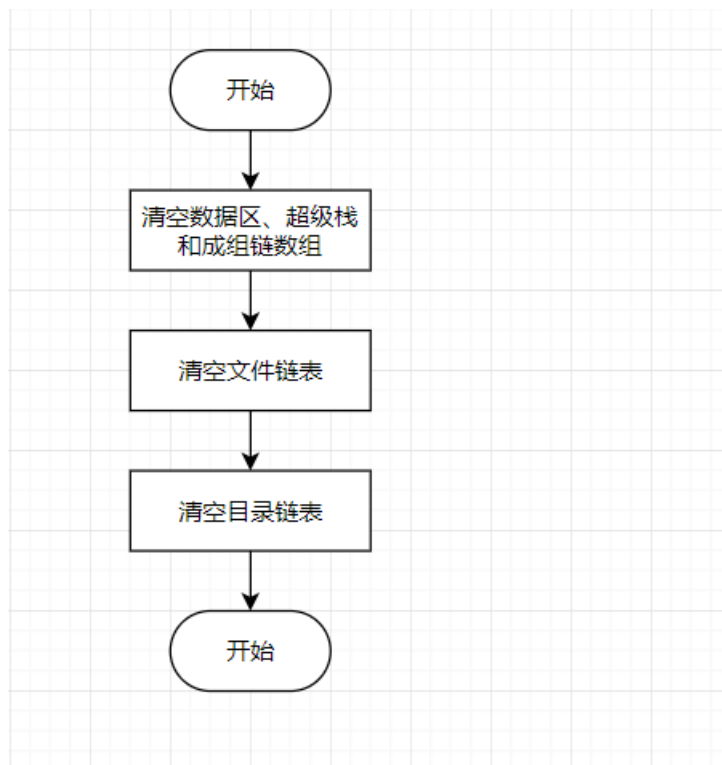


图 20 格式化

4 程序设计与实现

4.1 程序总体流程图

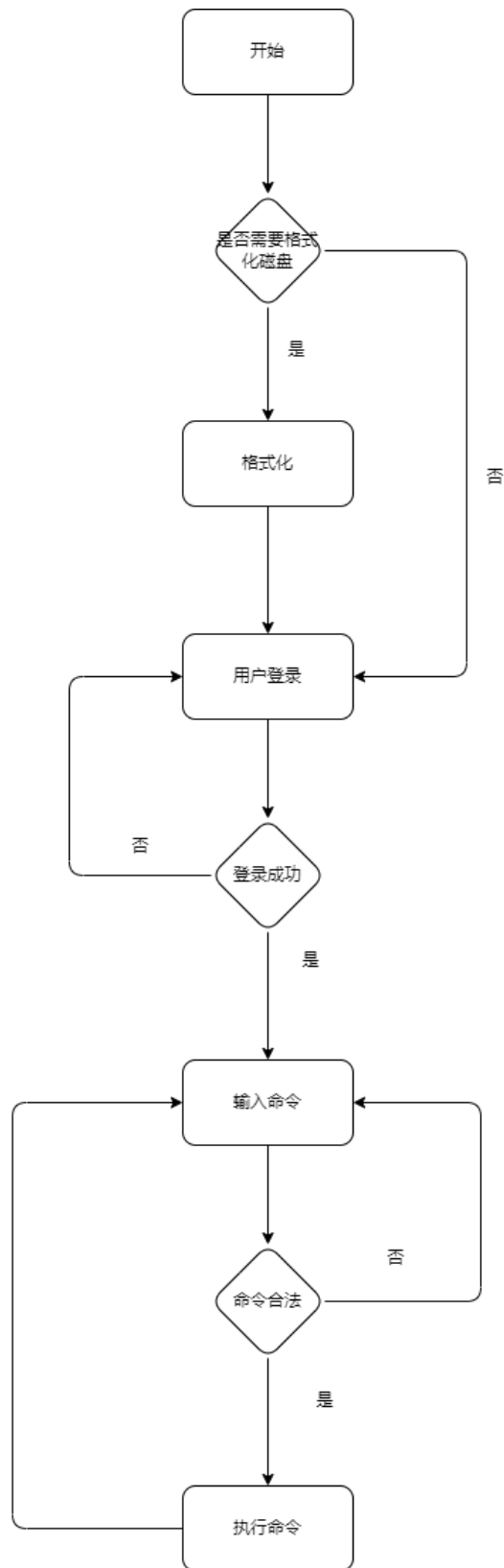


图 21 总体流程图

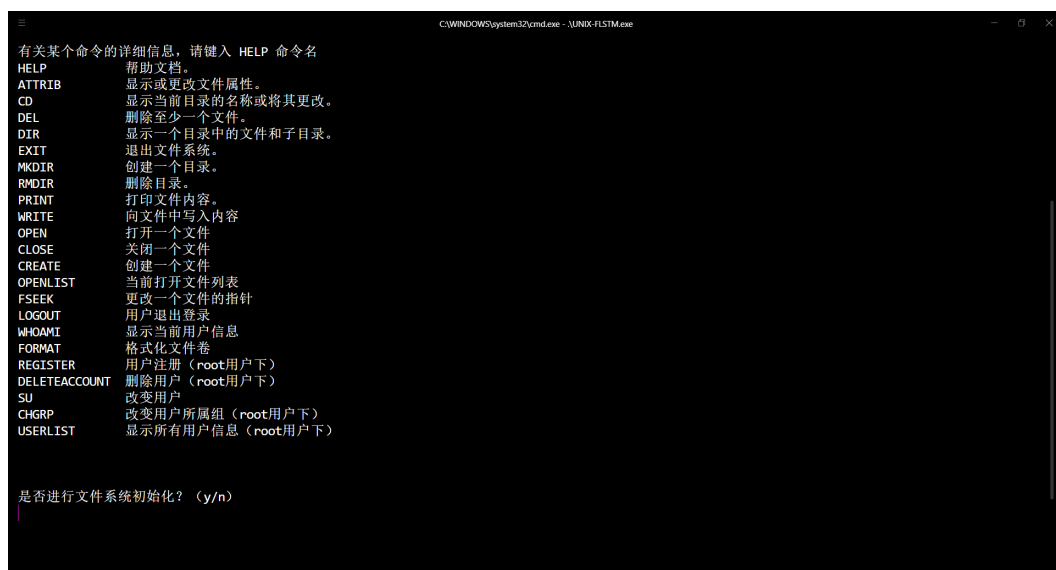
4.2 程序说明

本程序基于 C++语言进行编译开发，设计并实现了一个多用户、多级目录结构文件系统以模拟 Unix 的文件管理系统。该文件系统具有用户管理、文件管理、目录管理等功能，并使用命令行实现输入输出。其中，文件系统的核心功能之一就是文件索引结构的设计和实现。我们采用了内存管理模块、文件索引模块、文件管理模块、用户管理模块、目录管理模块进行功能划分。其中，文件索引模块负责管理文件的索引信息，包括文件的物理结构、目录结构以及空闲盘块的组织。具体来说，我们采用了串联结构表示文件的物理结构，在多级目录结构中采用了类似于 Unix 文件系统的层级目录结构。同时，我们还采用了成组链表法来管理空闲盘块，提高了文件系统对磁盘空间的利用率和性能。通过这些索引结构的设计和实现，我们实现了文件系统支持各种命令操作，包括创建文件、打开文件、关闭文件、删除文件、复制文件、写入文件、重复写入文件、读文件、更改文件指针、更改文件权限、创建目录及文件、删除目录及文件、进入指定目录、回退上一级目录、重命名文件或目录等。

4.3 实验结果

4.3.1 初始化

在初次打开exe，文件系统不存在的情况下，会提示用户进行文件系统的初始化



```
有关某个命令的详细信息，请键入 HELP 命令名
HELP      帮助文档。
ATTRIB    显示或更改文件属性。
CD         显示当前目录的名称或将其更改。
DEL        删除至少一个文件。
DIR        显示一个目录中的文件和子目录。
EXIT       退出文件系统。
MKDIR      创建一个目录。
RMDIR      删除目录。
PRINT      打印文件内容。
WRITE      向文件中写入内容。
OPEN       打开一个文件。
CLOSE      关闭一个文件。
CREATE     创建一个文件。
OPENLIST   当前打开文件列表。
FSEEK      更改一个文件的指针。
LOGOUT     用户退出登录。
WHOAMI     显示当前用户信息。
FORMAT     格式化文件卷。
REGISTER   用户注册（root用户下）。
DELETEACCOUNT 删除用户（root用户下）。
SU         改变用户。
CHGRP      改变用户所属组（root用户下）。
USERLIST   显示所有用户信息（root用户下）。

是否进行文件系统初始化？（y/n）
```

图 22:初始界面

随后提示需要进行登录，并输入用户名和密码，默认情况下存在两个用户：
root 用户（模仿 linux 系统下的 root 用户），testUser 用户（普通用户）
此时使用 root 用户登录

```
用户未登录，请输入用户名和密码
用户名: root
密码: root
root@root>
```

图 23:成功登录

4.3.2 基本命令进行测试

```
用户未登录，请输入用户名和密码
用户名: root
密码: root
root@root>userlist
      UserName      UserId  UserGroupId
      root          0         1
      testUser      1         2

root@root>register test1 test1
成功创建用户

root@root>register test1 test1
用户名已存在，无法注册
【错误码】6

root@root>userlist
      UserName      UserId  UserGroupId
      root          0         1
      testUser      1         2
      test1          2         2

root@root>
```

图 24:用户相关命令

```
root@root>help
有关某个命令的详细信息，请键入 HELP 命令名
HELP          帮助文档。
ATTRIB        显示或更改文件属性。
CD            显示当前目录的名称或将其更改。
DEL           删除至少一个文件。
DIR           显示一个目录中的文件和子目录。
EXIT          退出文件系统。
MKDIR         创建一个目录。
RMDIR         删除目录。
PRINT         打印文件内容。
WRITE         向文件中写入内容
OPEN          打开一个文件
CLOSE         关闭一个文件
CREATE        创建一个文件
OPENLIST      当前打开文件列表
FSEEK         更改一个文件的指针
LOGOUT        用户退出登录
WHOAMI        显示当前用户信息
FORMAT        格式化文件卷
REGISTER      用户注册（root用户下）
DELETEACCOUNT 删除用户（root用户下）
SU            改变用户
CHGRP         改变用户所属组（root用户下）
USERLIST      显示所有用户信息（root用户下）
```

图 25: help 命令

```

root@root>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:05:52 <DIR>              bin
2023-06-11 17:05:52 <DIR>              etc
2023-06-11 17:05:52 <DIR>              home
2023-06-11 17:05:52 <DIR>              dev

root@root>cd home

root@root/home>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:05:52 <DIR>              texts
2023-06-11 17:05:52 <DIR>              reports
2023-06-11 17:05:52 <DIR>              photos

root@root/home>cd texts

root@root/home/texts>create test.txt
文件名不合法, 当前目录已存在名为test.txt的文件
【错误码】2

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:05:52              0              test.txt

```

图26: 文件及目录相关测试

用 `mkdir` 命令创建子目录:

```

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:05:52              0              test.txt

root@root/home/texts>mkdir newdir
成功创建目录newdir

```

图27: 创建命令

```

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:05:52              0              test.txt
2023-06-11 17:16:07 <DIR>              newdir

```

图28: 当前目录结构

打开并从命令行写入文件:

```

root@root/home/texts>open test.txt
已成功打开文件test.txt

root@root/home/texts>write test.txt -s test
成功写入文件

```

图29: 打开并写入文件

阅读文件(因为每次写入之后文件指针默认在文件末尾, 所以必须手动将文件指针调回文件开头或打开再关闭, 此时文件指针自动移回开头):

```

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:18:44              4      test.txt
2023-06-11 17:16:07 <DIR>              newdir

root@root/home/texts>close test.txt
已成功关闭文件test.txt

root@root/home/texts>open test.txt
已成功打开文件test.txt

root@root/home/texts>print test.txt
test

```

图 30：读取文件内容

删除文件：

```

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:21:13              4      test.txt
2023-06-11 17:16:07 <DIR>              newdir

root@root/home/texts>del test.txt
成功删除文件test.txt

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:16:07 <DIR>              newdir

```

图 31：删除文件

删除目录：

```

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:16:07 <DIR>              newdir

root@root/home/texts>rmdir newdir
成功删除目录newdir

root@root/home/texts>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..

```

图 32：删除目录

删除目录失败（要删除的目录不为空）：

```

root@root>dir
      Edit Time      Type      Size(Byte)      File/Directory Name
2023-06-11 17:05:52 <DIR>              .
2023-06-11 17:05:52 <DIR>              ..
2023-06-11 17:05:52 <DIR>              bin
2023-06-11 17:05:52 <DIR>              etc
2023-06-11 17:05:52 <DIR>              home
2023-06-11 17:05:52 <DIR>              dev

root@root>rmdir home
当前目录不为空，不可以删除
【错误码】3

```

图 33：删除目录失败

4.3.3 用户相关命令

```
root@root>userlist
      UserName      UserId      UserGroupId
      root          0          1
      testUser      1          2
      test1         2          2

root@root>su test1 test1
成功更换用户

test1@root>whoami
userid: 2
username: test1

test1@root>logout
用户成功登出

用户未登录，请输入用户名和密码
用户名: root
密码: root
root@root>chgrp test1 3
成功更改用户test1到3组

root@root>userlist
      UserName      UserId      UserGroupId
      root          0          1
      testUser      1          2
      test1         2          3
```

图 34：用户相关基础命令

更改用户权限命令：

```
root@root/home/texts>dir /Q
Edit Time      Type      Size(Byte)  File/Directory Name  Owner Id  Owner Group  Inode Id  O[RWE]G[RWE]E[RWE]
2023-06-11 17:05:52 <DIR>      .              0          1          6          111 111 111
2023-06-11 17:05:52 <DIR>      ..             0          1          4          111 111 111
2023-06-11 17:26:46      4          test.txt      0          1          9          111 111 111

root@root/home/texts>openlist
当前打开的文件有：

root@root/home/texts>dir /Q
Edit Time      Type      Size(Byte)  File/Directory Name  Owner Id  Owner Group  Inode Id  O[RWE]G[RWE]E[RWE]
2023-06-11 17:05:52 <DIR>      .              0          1          6          111 111 111
2023-06-11 17:05:52 <DIR>      ..             0          1          4          111 111 111
2023-06-11 17:26:46      4          test.txt      0          1          9          111 111 111

root@root/home/texts>su test1 test1
成功更换用户

test1@root/home/texts>open test.txt
已成功打开文件test.txt

test1@root/home/texts>print test.txt
test
```

图 35：当前 test1 用户可读 test.txt 文件

```
root@root/home/texts>attrib -r e test.txt
已成功更改文件读写属性

root@root/home/texts>dir /Q
Edit Time      Type      Size(Byte)  File/Directory Name  Owner Id  Owner Group  Inode Id  O[RWE]G[RWE]E[RWE]
2023-06-11 17:05:52 <DIR>      .              0          1          6          111 111 111
2023-06-11 17:05:52 <DIR>      ..             0          1          4          111 111 111
2023-06-11 17:29:34      4          test.txt      0          1          9          111 111 011

root@root/home/texts>
```

图 36：更改 test.txt 文件属性使其他用户不可读

```
test1@root/home/texts>open test.txt
已成功打开文件test.txt

test1@root/home/texts>print test.txt
当前用户没有权限读文件
【错误码】3
```

图 37: test1 用户不可读 test.txt 文件

4.3.4 大文件读写

本部分中将尝试把当前可执行文件所在目录下的 makefile 文件写入该磁盘并在文件中读取

```
UNIX.FileSystem> build > Makefile
1  # CMAKE generated file: DO NOT EDIT!
2  # Generated by "MinGW Makefiles" Generator, CMake Version 3.26
3
4  # Default target executed when no arguments are given to make.
5  default_target: all
6  .PHONY : default_target
7
8  # Allow only one "make -f Makefile2" at a time, but pass parallelism.
9  .NOTPARALLEL:
10
11  #-----
12  # Special targets provided by cmake.
13
14  # Disable implicit rules so canonical targets will work.
15  .SUFFIXES:
16
17  # Disable VCS-based implicit rules.
18  % : %,v
19
20  # Disable VCS-based implicit rules.
21  % : RCS/%
22
23  # Disable VCS-based implicit rules.
24  % : RCS/%,v
25
26  # Disable VCS-based implicit rules.
27  % : SCCS/s.%
28
29  # Disable VCS-based implicit rules.
30  % : s.%
31
32  .SUFFIXES: .hpux_make_needs_suffix_list
```

图 38: makefile 文件内容

文件写入:

```
root@root/home/texts>create makefile.txt
成功创建文件makefile.txt

root@root/home/texts>open makefile.txt
已成功打开文件makefile.txt

root@root/home/texts>write makefile.txt -f Makefile
已将makefile文件内容写入makefile.txt

root@root/home/texts>dir
```

Edit Time	Type	Size(Byte)	File/Directory Name
2023-06-11 17:05:52	<DIR>		.
2023-06-11 17:05:52	<DIR>		..
2023-06-11 17:32:46		4	test.txt
2023-06-11 17:37:56		9221	makefile.txt

图 39: Makefile 文件写入磁盘

可以看到该文件大小为 9221 字节

文件读取(使用更改文件指针方法, 将文件指针调回开头):

```
@echo ... edit_cache
@echo ... rebuild_cache
@echo ... UNIX-FLSTM
@echo ... src/Block.obj
@echo ... src/Block.i
@echo ... src/Block.s
@echo ... src/Directory.obj
@echo ... src/Directory.i
@echo ... src/Directory.s
@echo ... src/File.obj
@echo ... src/File.i
@echo ... src/File.s
@echo ... src/User.obj
@echo ... src/User.i
@echo ... src/User.s
@echo ... src/main.obj
@echo ... src/main.i
@echo ... src/main.s
@echo ... src/tools.obj
@echo ... src/tools.i
@echo ... src/tools.s
.PHONY : help

#=====
# Special targets to cleanup operation of make.

# Special rule to run CMake to check the build system integrity.
# No rule that depends on this can have commands that come from listfiles
# because they might be regenerated.
cmake_check_build_system:
    $(CMAKE_COMMAND) -S$(CMAKE_SOURCE_DIR) -B$(CMAKE_BINARY_DIR) --check-build-system CMakeFiles\Makefile.cmake 0
.PHONY : cmake_check_build_system

root@root/home/texts>
```

图 40: 更改文件指针并读取

```
@echo ... edit_cache
@echo ... rebuild_cache
@echo ... UNIX-FLSTM
@echo ... src/Block.obj
@echo ... src/Block.i
@echo ... src/Block.s
@echo ... src/Directory.obj
@echo ... src/Directory.i
@echo ... src/Directory.s
@echo ... src/File.obj
@echo ... src/File.i
@echo ... src/File.s
@echo ... src/User.obj
@echo ... src/User.i
@echo ... src/User.s
@echo ... src/main.obj
@echo ... src/main.i
@echo ... src/main.s
@echo ... src/tools.obj
@echo ... src/tools.i
@echo ... src/tools.s
.PHONY : help

#=====
# Special targets to cleanup operation of make.

# Special rule to run CMake to check the build system integrity.
# No rule that depends on this can have commands that come from listfiles
# because they might be regenerated.
cmake_check_build_system:
    $(CMAKE_COMMAND) -S$(CMAKE_SOURCE_DIR) -B$(CMAKE_BINARY_DIR) --check-build-system CMakeFiles\Makefile.cmake 0
.PHONY : cmake_check_build_system

root@root/home/texts>
```

图 41: 读取的文件内容

5 结论

我们的课程项目中实现的文件系统较好地遵循了操作系统的设计原则，并且成功地完成了此次课程设计所要求的基本功能。同时，我们还在功能上做了一些扩展，将文件系统的运行过程较好地呈现出来，这使得文件系统的使用更加直观和方便。

在设计和实现文件系统的过程中，我们注重了模块化设计和代码规范，采用了合理的数据结构和算法来提高文件系统的效率和稳定性。通过实验结果可以看出，我们实现的文件系统能够较好地支持各种命令操作，包括创建、修改、删除文件和目录，读写文件等。同时，我们还实现了权限控制、多用户管理等扩展功能，增强了文件系统的灵活性和易用性。

综上所述，我们的文件系统较好地完成了此次课程设计的基本要求，并且在功能和易用性上做出了一定的扩展。通过这次实践，我们对操作系统中文件系统的设计和实现有了更深刻的认识，并且提升了自己的编程能力和团队协作能力。

6 参考文献

1. 徐虹等编著. 操作系统实验指导——基于 Linux 内核. 北京：清华大学出版社. 2004.
2. 陈向群等编著. Windows 内核实验教程. 北京：机械工业出版社. 2002.
3. 周苏等编著. 操作系统原理实验. 北京：科学出版社. 2003.
4. 张尧学编著. 计算机操作系统教程习题解答与实验指导. 北京：清华大学出版社. 2000.

7 收获、体会和建议

张海波：

在我们的课程项目中，我们学习并实践了操作系统中文件系统的设计和实现。通过这次实践，我们获得了很多有价值的心得和收获。

首先，我们深入了解了文件系统的组成部分和实现原理。我们学习了文件索引结构的设计和实现，包括物理结构、目录结构和空闲盘块管理等方面，在此基础上实现了多级目录结构和文件操作功能。我们还学习了不同用户之间的权限控制问题，实现了用户管理模块和文件权限机制。

其次，我们意识到了代码规范和注释对于项目实现的重要性。在项目实践过程中，我们注重代码的可读性和可维护性，采用注释和命名规范等方式提高了代码质量，并且实践了团队协作开发。

最后，我们也认识到了文件系统的效率和稳定性对于计算机系统的重要性。在实践过程中，我们通过多种技术手段优化了文件系统的性能，并保证了系统的安全性和稳定性。

总的来说，通过这次课程项目的实践，我们掌握了操作系统中文件系统的设计和实现方法，并且提升了我们的编程能力和团队协作能力。在未来的学习和工作中，我们将继续加强对操作系统中文件系统的学习，并且不断提升自己的技能水平，为实现更高效、更稳定的文件系统做出贡献。

杜学鸿：

在本次的课程设计中，我们选择了多用户、多级目录结构文件系统的设计与实现。完成了项目后，我们不仅巩固了课堂上学到的与文件系统相关的知识，还深入理解了多用户环境下文件系统的组织方式、目录结构的设计原则以及各项功能的实现方法。同时，我们对文件系统的多用户管理、权限控制以及数据存储与检索等方面有了更深入的认识。

在课程设计初期，我们进行了广泛的资料搜集和阅读，研究了现有文件系统的设计理念和实现方式。随后，我们进行了团队内部的讨论，确定了系统的整体架构、用户管理模块、目录结构、文件操作等关键要素。这个过程不仅增强了我们的分析和设计能力，还提升了团队合作和协调的能力，对我们日后的工作有着重要的意义。

在编写代码的过程中，我们遇到了一些挑战和困难，包括文件存储管理、用户权限控制、目录结构维护等方面的实现。然而，通过团队成员之间的积极交流和合作，我们克服了这些问题，并通过持续的调试和测试确保了系统的稳定性和功能完整性。这个过程不仅锻炼了我们的问题解决能力，还培养了我们的耐心和持续学习的精神。

我深刻认识到这次课程设计对我们的成长和提升具有重要意义。它使我们能够将课堂理论与实际项目相结合，培养了我们的系统设计和开发能力。同时，我要特别感谢冯时老师一直以来的指导和支持，他的专业知识和经验对我们的项目取得成功起到了关键作用，我们由衷地感谢他的悉心指导。