同济大学

TONGJI UNIVERSITY

《操作系统系统原理》

大作业报告

作业名称	基于 Unix V6++ 的文件系统设计
姓　　名	吴达鹏 1751118
学院（系）	电子与信息工程学院
专　　业	计算机科学与技术
任课教师	邓　蓉
日　　期	2023 年　3 月　31 日

目录

一、 实验概述

1.1 实验简介

识别、研读 UNIX V6++ 系统中文件系统和缓存管理模块。按需摘取其中的代码和数据结构，用以管理二级文件系统。

1.2 实验目的

学习 UNIX V6++ 如何分解文件系统功能，依托开源软件进行二次开发。

1.3 实验任务

1. 把一个大文件当一张磁盘用。命名为：myDisk.img。磁盘文件卷结构如下：

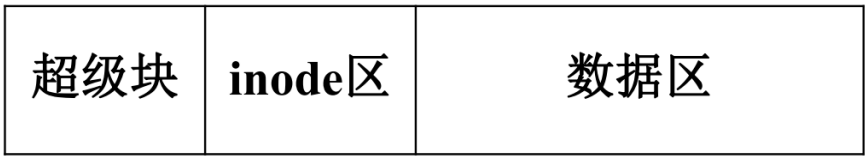

图 **1**　磁盘文件卷格式

实际上一个文件卷就是一个逻辑磁盘，磁盘以块为单位存储信息，块大小为 512 字节。

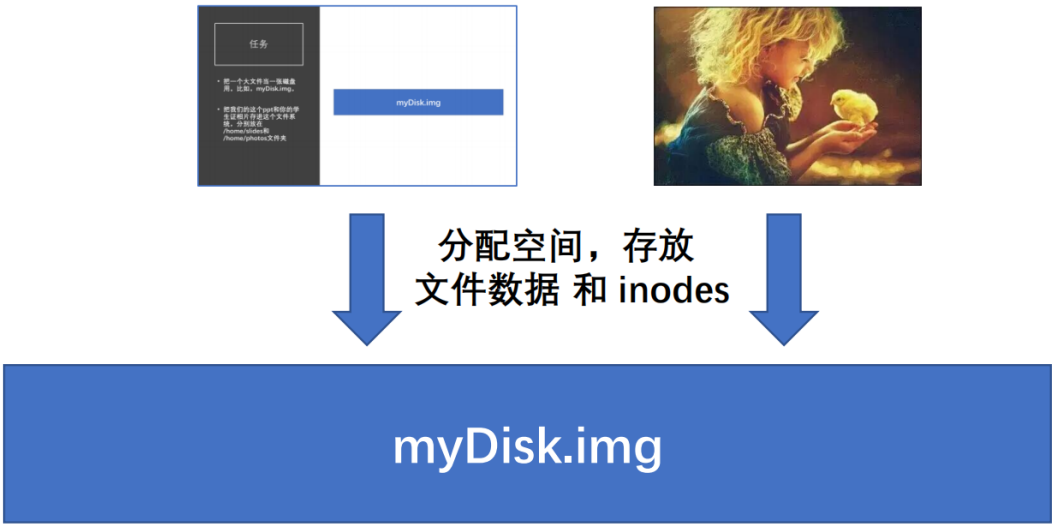

图 **2**　使用文件卷模拟逻辑磁盘

2. 把随意的一个纯文本文件，你的课设报告和你的头像，存进这个文件系统，分别放在/home/texts，/home/reports 和/home/photos 文件夹下。

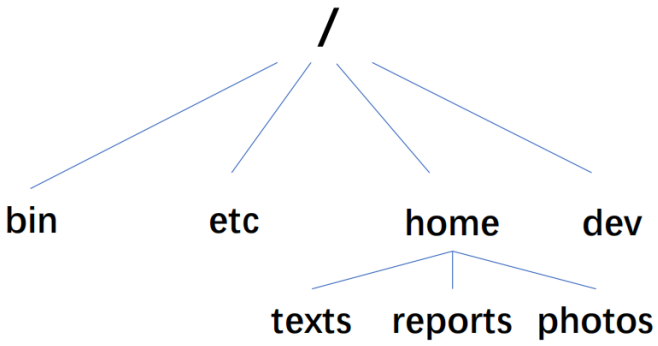

图 **3**　磁盘目录结构

3. 实现如下系统调用：

(a) creat
(b) open
(c) read
(d) write
(e) lseek
(f) close
(g) mkdir

(h) 其他可能会用到的

4. 图形化界面或命令行窗口，等待用户输入指令，使用我们的文件管理系统。要求系统具有较高的鲁棒性、健壮性、易操作性等等。
5. 要求通过测试：将宿主机中的文件存入虚拟磁盘并可以将虚拟磁盘中存放的文件取出，保存在宿主机文件系统中。要求.txt 能够被记事本打开，可执行程序能够跑起来。
6. 书写用户手册，介绍如何使用你所编写的二级文件系统 SecondFileSystem。手册应该包含足够的细节，使 UNIX 的初学者可以使用 SecondFileSystem，并可从中领会到 UNIX 文件操作的过程。按惯例，手册是纯文本，文件名应该是 readme。
7. 源代码必须层次清晰且经过充分注释，这样合作者才能够理解系统，对源代码进行维护。

1.4 实验环境

- 操作系统：Windows 10 家庭版开发版
- 语言：C++
- IDE：Visual Studio Code 2019
- gcc：9.2.0

二、 概要设计

2.1 模块设计

整个文件系统可分解为如下模块：

1. 顶层管理模块：主要负责用户输入信息处理、将用户输入命令转化为内核操作、反馈信息输出、错误反馈等。
2. 用户管理模块：主要负责用户相关的一系列操作。提供用户登录、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相关接口。
3. 内存管理模块：主要负责 SuperBlock、Inode、Block 的初始化、申请和释放。提供读入读出 SuperBlock、Inode、Block 块以及申请释放 Block 块的接口，同时提供每个 Inode 中逻辑 Block 块号和物理 Block 块号映射关系转换接口。
4. 文件管理模块：主要负责文件相关的一系列操作。提供创建文件、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。

5. 目录管理模块：主要负责文件目录相关的一系列操作。提供创建目录、删除目录、打开目录、获取当前目录等操作的相关接口

2.2 数据结构设计

主要参考 Unix V6++ 系统文件结构，几个主要的数据结构定义如下：

- Inode

  一个 Inode 大小为 64 字节，8 个 Inode 占满一个 Block（一个 Block 为 512 字节），内部定义了两个枚举类型，InodeType 枚举了该 Inode 结点的类型（File 为文件、DIRECTORY 为目录），InodePermission 枚举了该 Inode 的访问权限，分别为用户、用户同组人员、其他人员的可读、可写、可执行权限。详细定义如下：

```
/*Inode*/
struct Inode{

    /*Inode 结点类型，分为文件和目录 */
    enum InodeType {
        FILE = 0x1,
        DIRECTORY = 0x2
    };

    /* 访问权限控制，分为用户、用户同组人员和其他三类 */
    enum InodePermission {
        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
    };

    uint i_addr[NINODE];     /* 逻辑块号到物理块号的映射 */
    uint i_size;             /* 文件大小，单位为字节 */
    ushort i_count;          /* 引用计数 */
    ushort i_number;         /*Inode 的编号 */
    ushort i_mode;           /* 文件工作方式信息 */
    ushort i_permission;     /* 文件权限 */
    ushort i_uid;            /* 文件所有者的用户 id*/
    ushort i_gid;            /* 文件所有者的组 id*/
    time_t i_time;           /* 最后访问时间 */
    };
```

- SuperBlock
  SuperBlock 为整个文件系统的超级块，用于管理文件系统的 Inode 和 Block，包括 Inode 和 Block 总数，空闲 Inode 和 Block 数，以及采取成组链接法管理空闲 Inodek（编号存储在 s_free 中），每一组最大 Inode 容量为 50，其中 s_free[0] 指示下一组空闲表的位置，若已经是最后一组，s_free[0] = 0。详细定义如下：

```
/*SuperBlock*/
struct SuperBlock
{
    ushort inodeNum;         /*Inode 总数 */
    ushort freeInodeNum;     /* 空闲 Inode 数量 */
    uint blockNum;           /*Block 总数 */
    uint freeBlockNum;       /* 空闲 Block 数 */
    uint nfree;              /* 直接管理的空闲块数 */
    uint free[FREE_BLOCK_GROUP_NUM];                /* 空闲块索
 ↪  引表，每个盘块最多 50 个空闲块 */
};
```

- Directory
  Directory 存储目录列表，对于一个 Inode，如果它的 i_mode 属性为 DIRECTORY，说明该 Inode 是一个目录，其 i_add[0] 所指向的 Block 存储一个 Directory，该结构体中存储信息即为当前的目录信息。包含 InodeNumber 和 fileName 两个属性：

```
/*Directory: 目录 */
struct  Directory
{
    uint d_inodeNumber[SUB_DIRECTORY_NUM];
 ↪  /* 子目录 inode 号 */
    char d_fileName[SUB_DIRECTORY_NUM][STRING_LENGTH];
 ↪  /* 子目录文件名 */
};
```

- User
  User 结构体存储系统中所有用户的信息，从 1 号 Block 开始存放，详细信息定义如下：

```
/*User*/
struct User
{
    uint u_uid[USER_NUM];        /* 用户 id*/
    uint u_gid[USER_NUM];        /* 用户组 id*/
    char u_name[USER_NUM][STRING_LENGTH];     /* 用户名 */
    char u_password[USER_NUM][STRING_LENGTH];/* 用户密码 */
};
```

- File（暂时未用）
  File 存储打开文件信息，包括 Inode 编号、读写指针的位置和打开用户的 id：

```
    /*User*/
    struct User
    {
```

```
        uint u_uid[USER_NUM];          /* 用户 id*/
        uint u_gid[USER_NUM];          /* 用户组 id*/
        char u_name[USER_NUM][STRING_LENGTH];     /* 用户名
↪   */
        char u_password[USER_NUM][STRING_LENGTH];/* 用户密
↪   码 */
    };
```

2.3 模块间调用关系

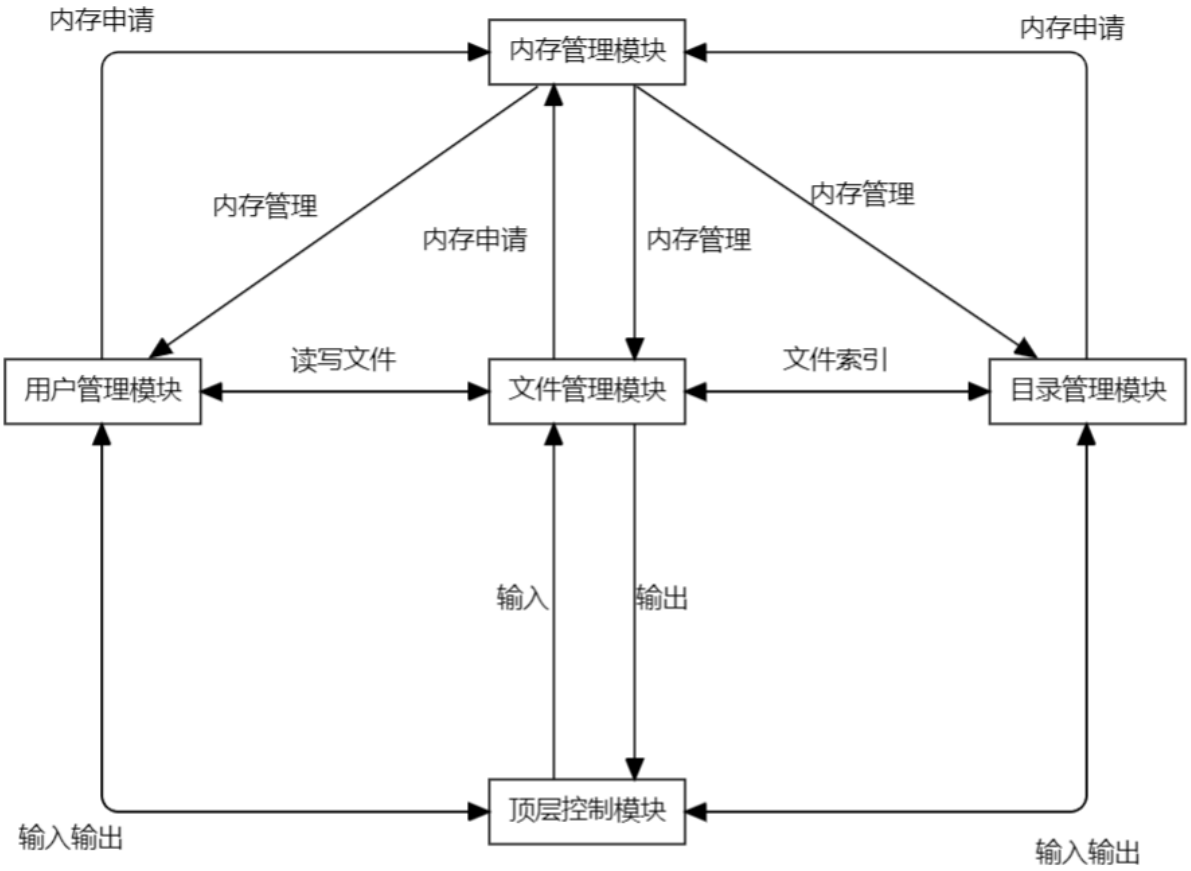

图 4　模块间调用关系

2.4 系统执行流程

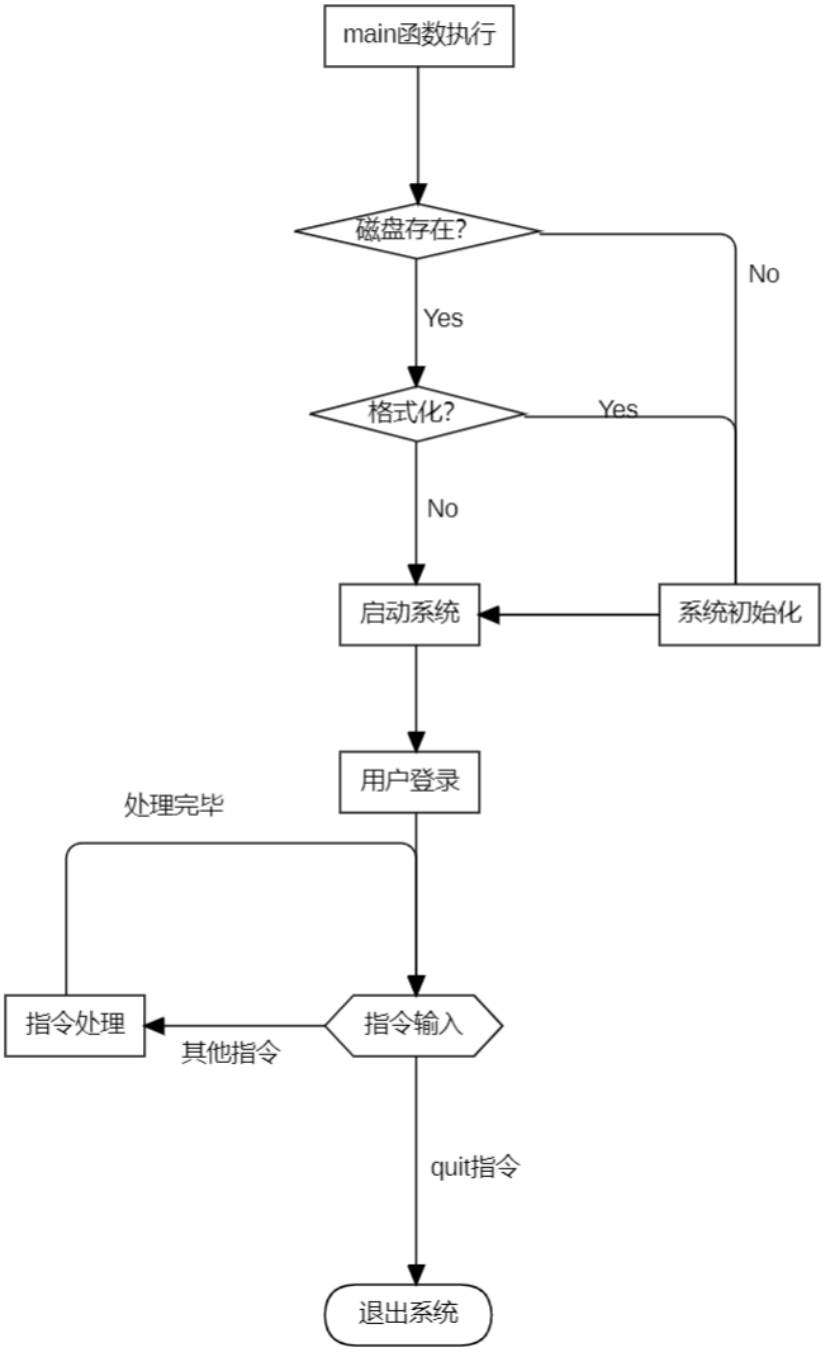

图 **5**　程序执行流程

三、 详细设计

3.1 文件卷详细结构

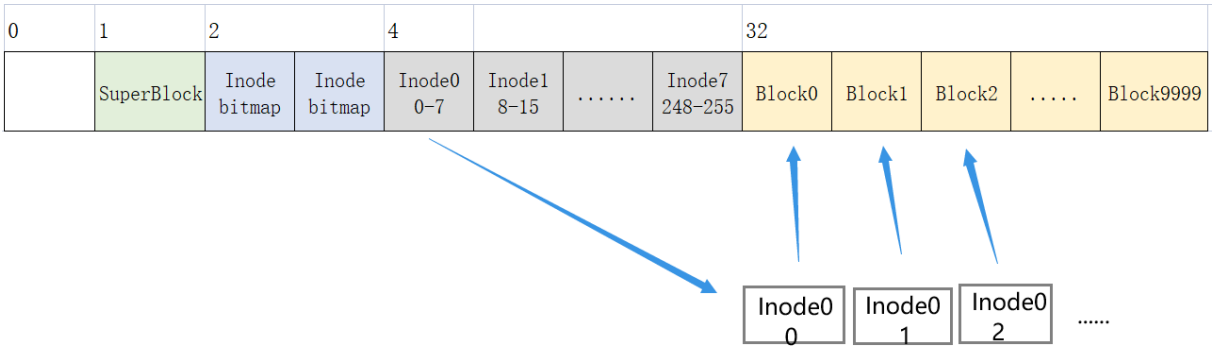

图 6　文件卷详细结构

3.2 用户管理模块

3.2.1 用户登录

由 User_Login() 函数完成，根据用户输入用户名和密码，到 User 结构中查询所有保存的用户信息，如果能匹配上账号密码，登录成功，否则登录失败。

```
/* 用户登录 */
void User_Login(char* username, char* password){

    /* 读取系统中的用户信息 */
    User user;
    Read_User(user);

    bool flag = false;
    for(int i = 0; i < USER_NUM; i++){
        if(!strcmp(user.u_name[i], username)){
            if(!strcmp(user.u_password[i], password)){
                /* 登录成功，更改当前用户的 id*/
                user_id = user.u_uid[i];
                flag = true;
```

```
                cout << " 登录成功！" << endl;
            }
            else{
                cout << " 密码错误！" << endl;
                throw(PASSWORD_ERROR);
            }
        }
    }

    /* 如果不存在该用户，给出提示 */
    if(!flag){
        cout << USER_NOT_EXIST << endl;
        throw(USER_NOT_EXIST_ERROR);
    }
}
```

3.2.2 创建用户

输出新的用户名和密码，查询 User 结构中是否存在相同用户名，如果不存在相同的用户名创建成功，否则创建失败。

3.3 内存管理模块

3.3.1 文件卷的初始化

包括 SuperBlock、Inode、Block 的初始化，封装在 initialize 函数中，Inode 的管理直接用一个长度为 InodeNum 的数组，每一个元素值取 1 或 0，表示该编号的 Inode 是否被分配（类似于位示图管理），Block 采用成组链接法管理。整个初始化流程大致如下：

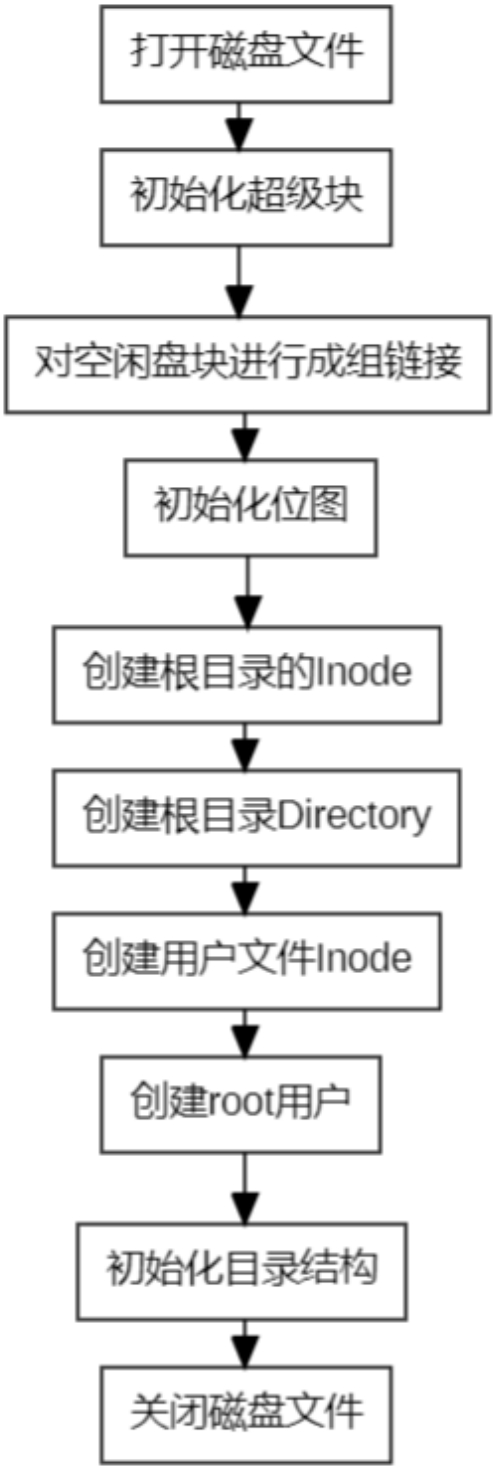

图 **7**　文件卷初始化流程图

部分代码如下：

```
/* 内存初始化 */
```

```
void Initialize(){

    ......

    /* 初始化超级块 */
    SuperBlock superBlock;
    superBlock.inodeNum = INODE_NUM;

    /* 总块数 = Block 数 + SuperBlock 所占块数 + Inode 位图所
↪   占块数 */
    superBlock.blockNum = BLOCK_NUM + BLOCK_START_POS;
    superBlock.freeInodeNum = INODE_NUM - 2;
↪   /* 两个位置被占用 */
    superBlock.freeBlockNum = BLOCK_NUM - 2;
↪   /* 两个位置被占用 */

    /* 直接管理的 Block 空间（0-49），这里编号 49~0*/
    for(int i = 0; i < FREE_BLOCK_GROUP_NUM; i++)
        superBlock.free[i] = FREE_BLOCK_GROUP_NUM - 1 - i;
    superBlock.nfree = FREE_BLOCK_GROUP_NUM - 1 - 2;

    /* 超级块写入内存 */
    fd.seekg(SUPER_START_POS * BLOCK_SIZE, ios::beg);
    fd.write((char*)&superBlock, sizeof(superBlock));

    /* 进行成组链接,stack[i] 表示的是某一组的空闲盘块号，若
↪   stack[0]，表示空闲盘块链结束 */
    uint stack[FREE_BLOCK_GROUP_NUM];

    for(uint i = 2; i <= BLOCK_NUM FREE_BLOCK_GROUP_NUM;
↪   i++){

        /* 为每一组的空闲盘块编号 */
        /* 第一组：99~50，第二组：149~100，第三组：
↪   199~150......*/
```

```
        for(uint j = 0; j < FREE_BLOCK_GROUP_NUM; j++){
            stack[j] = FREE_BLOCK_GROUP_NUM * i - 1 - j;
        }

        /* 写入内存：1、调整写入位置 */
        if(i == BLOCK_NUM / FREE_BLOCK_GROUP_NUM){
            stack[0] = 0;
            fd.seekg((BLOCK_START_POS + BLOCK_NUM - 1 -
↪   FREE_BLOCK_GROUP_NUM) * BLOCK_SIZE, ios::beg);
        }
        else{
            /* 每一组第一个空闲块的编号做偏移量 */
            fd.seekg((BLOCK_START_POS + stack[0] -
↪   FREE_BLOCK_GROUP_NUM) * BLOCK_SIZE, ios::beg);
        }

        /* 写入内存：2、文件内写入该组空闲盘块编号 */
        fd.write((char*)stack,sizeof(stack));
    }

    /* 初始化位图 */
    uint ibitmap[INODE_NUM] = { 0 };
    ibitmap[0] = ibitmap[1] = 1;

    ......

    /* 创建根目录 */
    Inode inodeRoot;
    inodeRoot.i_number = 0;                 /* 根目录的 Inode
↪   的编号为 0*/
    inodeRoot.i_addr[0] = 0;                /* 对应 0 号
↪   Block*/
    inodeRoot.i_mode = Inode::DIRECTORY;    /* 表示为目录 */
    inodeRoot.i_count = 0;                  /* 引用计数 */
```

```
    inodeRoot.i_uid = 0;                    /*uid 为 0，表示属
↪   于管理员 */
    inodeRoot.i_gid = 1;                    /* 管理员组号为
↪   1*/
    inodeRoot.i_size = 0;                   /* 目录大小为 0*/
    inodeRoot.i_time = time(NULL);          /* 当前时间 */
    inodeRoot.i_permission = 0777;          /* 对所有人开放访
↪   问权限 */

    ......

    /*0 号 block 中写入根 Directory*/
    Directory rootDirectory;
    strcpy(rootDirectory.d_fileName[0], ".");      /* 第一个目
↪   录项为.，表示当前目录 */
    strcpy(rootDirectory.d_fileName[1], "..");     /* 第二个目
↪   录项为..，表示父亲，根的父亲是自己 */
    rootDirectory.d_inodeNumber[0] = 0;     /* 当前目录 inode
↪   编号为 0*/
    rootDirectory.d_inodeNumber[1] = 0;     /* 根目录父亲为自
↪   己，编号也为 0*/

    ......

    /* 创建用户文件 */
    Inode inodeAccounting;
    inodeAccounting.i_number = 1;           /*inode 编号 */
    inodeAccounting.i_addr[0] = 1;          /* 对应 1 号
↪   block*/
    inodeAccounting.i_mode = Inode::FILE;   /* 用户文件 */
    inodeAccounting.i_count = 0;            /* 引用计数为 0*/
    inodeAccounting.i_permission = 0700;    /* 仅管理员访问 */
    inodeAccounting.i_uid = 0;              /* 所属用户为管理
↪   员 */
    inodeAccounting.i_gid = 1;              /* 管理员组号 */
```

```
    inodeAccounting.i_size = 0;                 /* 目录大小为 0*/
    inodeAccounting.i_time = time(NULL);        /* 当前时间 */

    ......

    /* 创建一个 root 用户 */
    User user;
    user.u_uid[0] = 0;
    user.u_gid[0] = 1;
    strcpy(user.u_name[0],"root");
    strcpy(user.u_password[0],"root");

    ......

    /* 初始化根目录 */
    user_id = 0;
    Create_Directory((char*)"bin");
    Create_Directory((char*)"etc");
    Create_Directory((char*)"home");
    Create_Directory((char*)"dev");
}
```

3.4 目录管理模块

3.4.1 创建目录

Create_Directory(char* dir) 函数负责创建目录一个名为 dir 的目录，创建目录的过程，主要是为新目录分配 Inode、Block 和 Directory，并对文件卷存储的内容进行维护。创建目录的主要步骤如下：

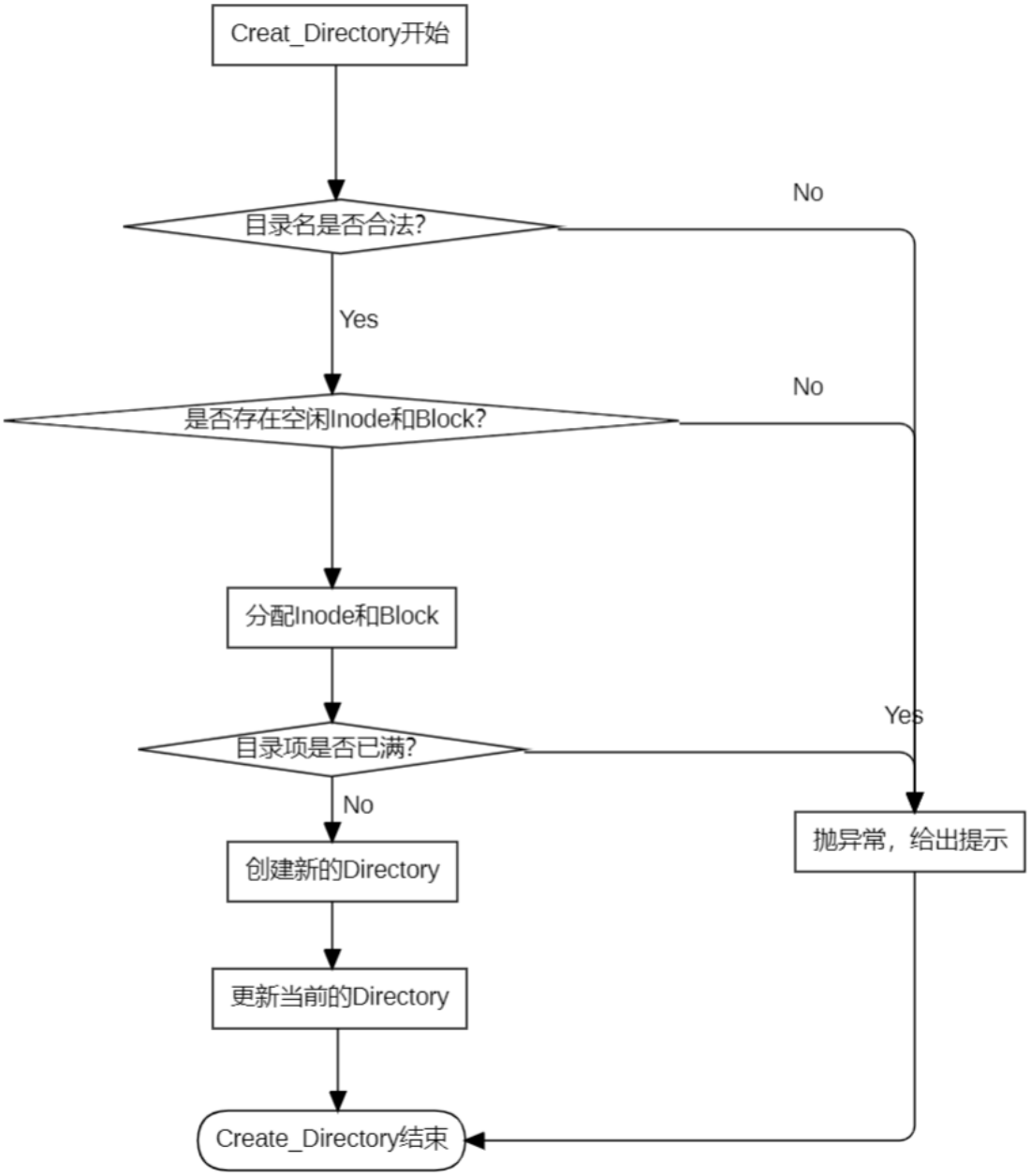

图 **8**　创建目录流程图

部分代码如下：

```
/* 创建名为 dirName 的目录 */
void Create_Directory(char *dirName)
{

    /* 检查目录名是否为空 */
```

```
    if (dirName == "")
    {
        cout << " 目录名不能为空" << endl;
        throw(OPEN_ERROR);
    }

    /* 检查当前目录是否有同名目录 */
    for (int i = 0; i < SUB_DIRECTORY_NUM; i++)
    {

        // 如果当前目录下有同名项，判断类型是否相同（是否同时为目录或文件）
        if (!strcmp(curDirectory.d_fileName[i], dirName))
        {

            Inode temp_inode;
            int inumber = curDirectory.d_inodeNumber[i];

            /* 打开磁盘，查询这个 Inode*/
            Read_Inode(temp_inode, inumber);

            /* 如果这个 Inode 是文件，就无所谓，因为当前是创建目录，允许目录和文件同名 */
            if (temp_inode.i_mode & Inode::FILE)
                continue;

            /* 如果这个 Inode 是目录，就冲突，抛出异常 */
            cout << FILE_EXISTS << endl;
            throw(FILENAME_ERROR);
        }
    }

    ......

    /* 检查是否有空闲 Block 和 Inode 可以分配 */
```

```
    if (superBlock.freeBlockNum == 0 ||
↪   superBlock.freeInodeNum == 0)
    {
        cout << "blockNum = " << superBlock.freeBlockNum << "
↪   inodeNum = " << superBlock.inodeNum << endl;
        cout << NOT_ENOUGH_MEMORY << endl;
        throw(SPACE_ERROR);
    }

    ......

    /* 匹配找到的第一个空闲 Inode*/
    uint newInodeIdx = 0;
    while (newInodeIdx < INODE_NUM && ibitmap[newInodeIdx] ==
↪   1)
        newInodeIdx++;

    ......

    /* 分配新的 block，将新目录存储在编号为 blockNumer 的
↪   block 中 */
    uint blockNumer;
    Allocate_Block(blockNumer);

    ......
    /* 将 directory 写入到 block 中 */
    Directory newDirectory;
    strcpy(newDirectory.d_fileName[0], ".");
    strcpy(newDirectory.d_fileName[1], "..");

    /* 将未使用的 inodeNumber 置为-1*/
    for (int i = 2; i < SUB_DIRECTORY_NUM; i++)
    {
        newDirectory.d_inodeNumber[i] = -1;
        newDirectory.d_fileName[i][0] = '\0';
```

```
    }

    /* 新路径的当前 inode 索引为 newInodeIdx*/
    newDirectory.d_inodeNumber[0] = newInodeIdx;

    /* 新路径的父亲 inode 索引为当前路径 */
    newDirectory.d_inodeNumber[1] =
↪   curDirectory.d_inodeNumber[0];

    /* 写入 */
    Write_Directory(newDirectory, blockNumer);

    ......

    /* 更新当前目录:2、修改当前目录项，itemCount 为找到的一个可
↪   用的目录项的下标，如果文件/目录数量已达到最大限制，创建目录
↪   失败 */
    int itemCount = 0;
    for (int i = 0; i < SUB_DIRECTORY_NUM; i++)
        if (strlen(curDirectory.d_fileName[i]) > 0)
            itemCount++;
    if (itemCount == SUB_DIRECTORY_NUM)
    {
        cout << ITEM_NUM_EXCESS << endl;
        throw(COUNT_ERROR);
    }
    strcpy(curDirectory.d_fileName[itemCount], dirName);
    curDirectory.d_inodeNumber[itemCount] =
↪   new_iNode.i_number;
    ......
}
```

四、 测试结果

完成了简单的磁盘初始化工作，程序初次运行，找不到 myDisk.img 文件，需要初始化，初始化完成后系统创建了 root 用户，并初始化好了根目录的目录结构，具体结果如下：

```
PS D:\totalStudy\sixth\oslab\myFileSystem> ./main
================================================
              二级文件管理系统 1.0.0
                1751118 吴达鹏
================================================

磁盘不存在，按回车键初始化......

用户未登录，请先登录！
用户名:
root
密  码:
root
登录成功！
root>ls
               name                 update time      type      size
                  .          2023-04-02 16:02:07       dir
                 ..          2023-04-02 16:02:07       dir
                bin          2023-04-02 16:02:07       dir
                etc          2023-04-02 16:02:07       dir
               home          2023-04-02 16:02:07       dir
                dev          2023-04-02 16:02:07       dir
root>
```

图 **9**　初始化测试结果

五、 未完待续.....