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：物联网 1501 班
组长学号：20152021
组长姓名：吉晓琪
指导教师：乔建忠

设计时间：2018 年 7 月

设计分工

组长学号及姓名：20152021 吉晓琪

分工：文件系统数据结构设计、目录创建和删除、文件及目录的复制、粘贴、剪切、查找、重命名及快捷方式的创建和删除、系统初始化、路径切换、系统退出保存、密码修改、密码显示、格式化、报告撰写

组员 1 学号及姓名：20153510 刘英杰

分工：文件系统数据结构设计、文件读取及写入、文件的多级索引、文件创建和删除、空闲磁盘块的分配和回收（成组链接法）、磁盘格式化、文件创建和删除、i 结点的分配和释放、报告撰写

组员 2 学号及姓名：20157546 李灵灵 （宁夏）

分工：文件系统数据结构设计、指令及文件目录名的自动补全、用户权限管理、用户登入登出、显示位示图、显示目录、系统界面和系统交互用户引导的设计、指令识别、报告撰写

摘要

文件系统又是操作系统中不可或缺的一部分，通过相应的算法和数据结构，对计算机系统的存储结构进行组织、分配，负责文件的存储，并对存入的文件实施保护、检索的一组软件集合。

在这次课程设计中，我们组选择的题目是多用户、多级目录结构文件系统，我们结合老师课上所讲有关文件系统的内容，自主设计了数据结构与相关算法，模拟了一个简易 UNIX 文件系统。能够实现多用户的登入、登出及权限管理，系统初始化，系统的退出保存，文件及目录的创建、修改和删除，多级目录，目录切换等基本功能，通过成组链接法对空闲磁盘块进行分配与回收。此外，该文件系统还提供指令及文件目录名的自动补全，文件及目录的复制、粘贴、剪切、重命名、查找和创建快捷方式，格式化等扩展功能。

界面设计友好，交互体验较好，使用过程中会有明确的交互性语句提示，提供不同颜色的提示强化效果，指令帮助说明清晰，程序健壮性和容错性较好。

关键词：Unix 文件系统、成组链接、多级索引、指令自动补全

目　　录

1 概述

本次课程设计编程实现了一个以“多用户、多级目录结构文件系统的设计与实现”为主题的模拟 UNIX 文件系统。该系统能实现 UNIX 文件系统的大部分管理功能，如多用户的登入、登出及权限管理，系统初始化，系统的退出保存，文件及目录的创建、修改和删除，多级目录，目录切换等基本功能，通过成组链接法对空闲磁盘块进行分配与回收。此外，该文件系统还提供指令及文件目录名的自动补全，文件及目录的复制、粘贴、剪切、重命名、查找和创建快捷方式，格式化等扩展功能，能对用户输入的错误命令进行错误提示，还能高效管理磁盘的 i 节点、SFD、数据区等。且提供友好的用户界面，交互体验较好，使用过程中会有明确的交互性语句提示，提供不同颜色的提示强化效果，指令帮助说明清晰， 程序健壮性和容错性较好。

经过两周的课程设计，小组完成了对文件系统的实现，并进行了一定的功能扩展。此外，本文件系统使用合理的数据结构表示超级块、i 结点区、目录及数据区等，使用多级索引方式存储文件内容，使用成组链接法对空闲磁盘法进行分配和回收。使用位势图的数据结构来统一分配回收空闲目录和 i 结点，充分模拟了 Unix 系统的各个功能模块。

2 课程设计任务及要求

2.1 设计目的

操作系统课程设计是本课程重要的实践教学环节。课程设计的目的，一方面使学生更透彻地理解操作系统的基本概念和原理，使之由抽象到具体；另一方面， 通过课程设计加强学生的实验手段与实践技能，培养学生独立分析问题、解决问题、应用知识的能力和创新精神。与本课程的实验教学相比，课程设计独立设课， 具有更多的学时，给学生更多自行设计、自主实验的机会，充分放手让学生真正培养学生的实践动手能力，全面提高学生的综合素质。

2.2 设计内容

多用户、多级目录结构文件系统的设计与实现

2.3 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。

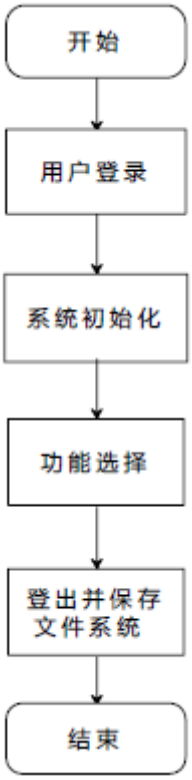
3 算法及数据结构

3.1 算法的总体思想（流程）

文件系统程序总体分为初始化模块、用户登入登出及权限管理管理模块、界面及指令识别模块、指令及文件目录名的自动补全模块、空闲磁盘块的分配回收模块、文件的创建删除模块、文件的读取写入模块、目录的创建删除模块、文件及目录的复制剪切链接模块、文件及目录的重命名和查找模块、格式化模块、系统的退出保存模块等十三个模块。

当程序运行时，首先进入登录模块，登录成功后文件系统进行初始化，将上次使用文件系统操作后的结果重新载入到模拟文件系统各个数据结构中，初始化完毕之后，当前登陆的用户可以选择各种功能，对文件系统进行各种操作，操作过程中有正确并恰当的操作提示以及指令和文件目录名的自动补全，帮助用户方便快捷地完成想要进行的操作，其中操作包括进入下一级或返回上一级目录，创建和级联删除当前目录，创建删除文件，读写文件，复制、粘贴、剪切、重命名、查找文件和目录，退出系统，格式化当前用户的文件系统等基本功能。当选择退出系统后，文件系统会调用文件系统的退出保存模块，将用户操作的结果放入到磁盘中进行保存，以便下次启动程序时的初始化。

整体执行的流程图如下（其中功能选择部分为整个文件系统执行的过程，该部分的各个功能的相应的操作流程和流程图在各个模块中有相应展示）：



3.2 整体数据结构

在本文件系统中包括的数据结构有超级块（记录文件系统的整体信息）、目录和 i 结点结构（对目录信息进行统一管理）、磁盘块结构（存储文件具体内容的信息）、成组链接的数据结构（统一分配回收空闲磁盘块号）、位势图的数据结构（统一分配回收空闲目录和 i 结点）、多级索引的数据结构（存储文件的内容），充分模拟了 Unix 系统的各个功能模块。

课程设计起初，小组成员共同设计好要使用的数据结构，方便之后函数和算法调用，同时使结构清晰，提高程序可读性，简化其他成员继续开发的难度。

本文件系统使用的数据结构及相应注释如下：

```
1  struct SingleSFD{//单个目录项
2      string name; //文件名
3      int id; //文件对应的i结点的id号
4  };
5
6  struct SFD{                              //目录结构
7      int sfdNum; // 动 态 数 组 的 大 小
8      vector<SingleSFD> sfdVec; //目录下的sfd数组
9  };
10
11 struct DiskBlock{//磁盘块结构
12     string file; //文件内容
13     int strNum;       //磁盘块大小
14 };
15
16 struct INode{              //i结点结构
17     int id;                 //i结点所属的用户
18     int type;               //文件类型，0-文件，1-目录
19     int sfd_id;             //i结点对应的目录id
20     int filelen;            //文件长度
21     int auth[8];            //8个user的访问权限
22     int diskBlockNum; //文件占用磁盘块个数
23     int diskBlockId; //所占磁盘块的id号的索引块
24     int qcount; // 文 件 的 引 用 数
25 };
26
27 struct SuperBlock{            //超级块结构
28     int i_node; //i 结 点 总 数
29     int freei_node; //空闲i结点总数
30     vector<int> freeiid; //空闲i结点id数组
31
32     int sfd; // 目 录 数 目
33     int free_SFD; // 空 闲 目 录 数
34     vector<int> freeSFD; //空闲目录下表数组
35
36     int disk;               //磁盘块总块数
37     int freeDisk; //空闲磁盘块块数
38     int freeDiskSta[51]; //成组链接空闲磁盘块栈
39 };
```

```
40
41  struct FileSystem{                    //文件系统结构
42      int boot; // 引 导 区
43      SuperBlock superBlock; //超级块
44      INode iNode[128]; //i 结 点
45      DiskBlock diskBlock[512];//磁盘块
46      SFD sfd[512]; // 目 录 块
47  };
48
49
50  string user; //当前正在访问文件系统的用户
51  FileSystem fileSystem;//文件系统
52
53  int iNode[128]; //i结点位势图，0表示占据，1表示空闲
54  int diskBlock[512]; //磁盘块位势图，0表示占据，1表示空闲
55  int SFDBlock[512]; //目录位势图，0表示占据，1表示空闲
56  int cur_SFD;              //当前所在sfd，当前所在目录
57  int copy_flag;            //粘贴板标志
58  SingleSFD copySFD;    //粘贴板
59  stack<int> staSFD; //记录目录栈
```

3.3 初始化模块

3.3.1 功能

该模块在文件系统程序打开时执行，登录成功之后，对模拟文件系统的初始化，包括超级块、i 节点、磁盘块和 SFD 进行使用前的初始化，重新载入上次程序运行退出时所保存的内容及状态。

3.3.2 数据结构

```
1  string user;   //全局变量，标志当前正在访问文件系统的用户
2  FileSystem fileSystem; //文件系统
3
4  int iNode[128]; //i结点位势图，0表示占据，1表示空闲
5  int SFDBlock[512]; //目录位势图，0 表示占据，1 表示空闲
```

3.3.3 算法

在初始化模块中用到了以下几个函数：

```
5    void initSuperBlock();      //初始化超级块
6    void initINode();           //初始化i结点
7    void initDiskBlock();       //初始化磁盘块
8    void initSFD();             //初始化SFD
9    void init();                //初始化
```

其中各个函数分别对超级块、I 结点、磁盘块和目录结构的各个数据域进行初始化的过程，每个功能函数中首先打开存储相应数据的磁盘文件，之后从文件中读出相应数据域的内容，写入到模拟文件系统的数据结构中，实现对整个文件系统的初始化功能。

3.3.4 流程图

流程图如下：

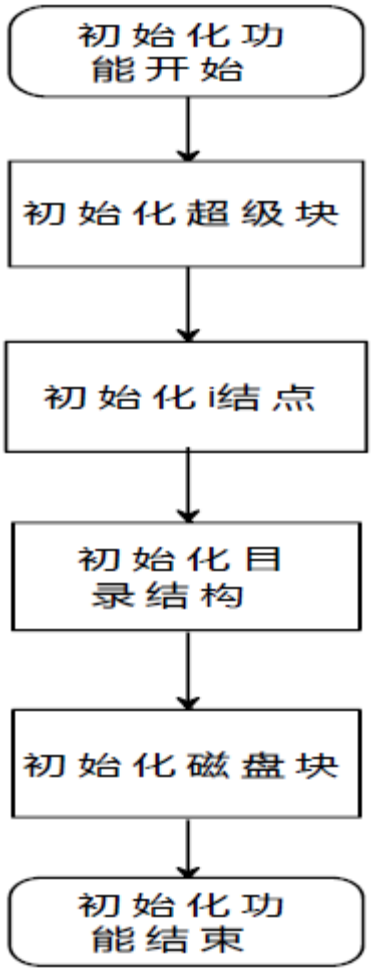


图 3.3.1 初始化流程

3.4 用户登录登出及权限模块

3.4.1 功能

该模块用于实现程序对多用户的控制，程序只允许登陆一个用户，用户拥有读写自己目录下文件的权限以及只读其它用户目录下文件的权限。程序通过全局变量控制当前登陆的用户类别和状态，并可以检测用户对文件的读取权限。

3.4.2 数据结构

```
10 string user;   //全局变量，标志当前正在访问文件系统的用户
11 FileSystem fileSystem; //文件系统
```

3.4.3 算法

```
12 void login();                          //登录用户名
13 void logout();                         //登出
14
15 int checkUser(string user);            //判断此时用户
16 int getInodeNum();                     //获得当前目录的i结点号
6  int checkFileAuth(string filename); //检查当前用户的读写权限
7  int checkDirAuth(string filename); //检查当前用户对目录的读写权限
```

login()函数给全局的 user 赋值，把当前输入的用户名记录下来。

logout()函数调用系统退出函数，保存当前文件系统的内容和状态，并给全局的user 赋空值，退出当前用户。

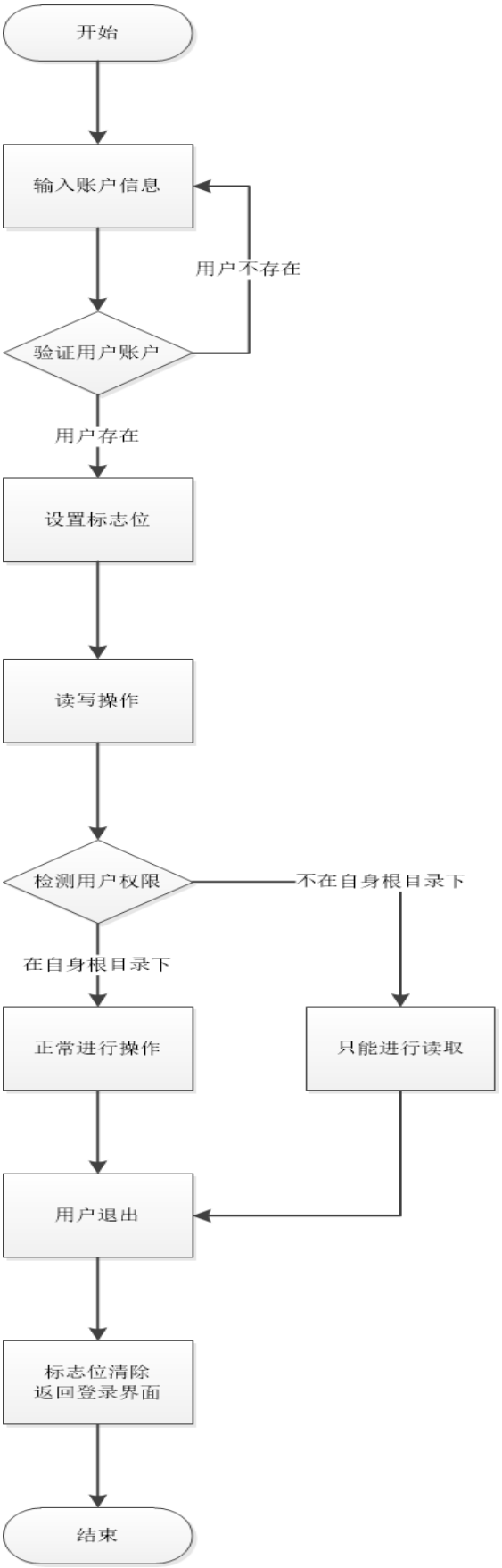
checkUser(string user)函数接收当前的 user 字符串，把它转换成用户的数字下标，方便判断用户权限。

checkFileAuth(string filename)函数通过判断当前SFD 与当前i 节点的内容，返回当前文件 i 节点的用户权限。

checkDirAuth(string filename)函数通过判断当前 SFD 与当前 i 节点的内容，返回当前文件 i 节点的用户权限。

getInodeNum()函数在所有文件 i 节点下 sfd_id 寻找当前的 SFD 的id，最后返回 SFD 的id 号。

3.4.4 流程图


开始
输入账户信息
用户不存在
验证用户账户
用户存在
设置标志位
读写操作
检测用户权限
不在自身根目录下
在自身根目录下
正常进行操作
只能进行读取
用户退出
标志位清除
返回登录界面
结束

3.5 界面及指令识别模块

3.5.1 功能

该模块为用户提供一个界面友好，简单易用的操作界面，类似于 MS DOS 中的命令行输入方式。用户输入命令进入，并且对用户输入的命令进行准确的识别并调用相应的功能模块，响应用户的操作。

使用 C 和 C++里的输入输出语句，程序通过接收命令、分析命令、执行命令来完成用户所需要的功能。程序中主要使用 display()这个函数，使用户利用程序的输入输出语句与文件系统进行交互，连接用户输入的指令和程序相应的函数， 执行函数，并反馈输出结果。这个模块整合了文件系统所有的关键函数，实现了人机交互，是程序中的主干部分。

3.5.2 算法

```
1  void dis_help();            //显示帮助
2  void display(); //界面主函数，用来实现大部分输入输出功能
3  int  checkIn(string in);  //用来检测命令是否存在以及指令的种类
4  void textcolor(int color);//设置字体颜色
```

display()函数中定义了一个 String 型变量 str，用来显示文件系统的提示符号（信息），in1 用来存放用户输入字符串，filename 用来存放双命令（如：create、type、mkdir 等）的第二个命令字符串。指令和文件名均通过函数 cin_command(string &in)和 cin_name(string &name)来获取用户的输入，可以实现自动补全的功能，通过 cout 语句在控制台显示当前操作的结果和信息。函数主要采取 if...else if...else...的条件结构对不同命令进行分析，双命令采用两次输入、两次判断的分析方式。

dis_help()函数提供系统的帮助提示界面显示。 checkIn()函数分析当前输入的命令是否为文件系统的关键字,并判断指令的种类，是单目指令，还是多目指令。

textcolor()用来设置字体的颜色。 display()函数中还设计了将所有用户输入命令的字符串转化为小写字母的函数，方便之后的命令分析。

此外，系统中包括使登陆用户注销（登出）的专门函数，在 display()中，如果判断当前进入了注销的命令下的语句，程序就会将 display()函数转入到 login()登陆函数，用户重新登陆，这样也能实现注销的功能，而不用设置全局的登陆状态变量，浪费空间。

3.5.3 流程图

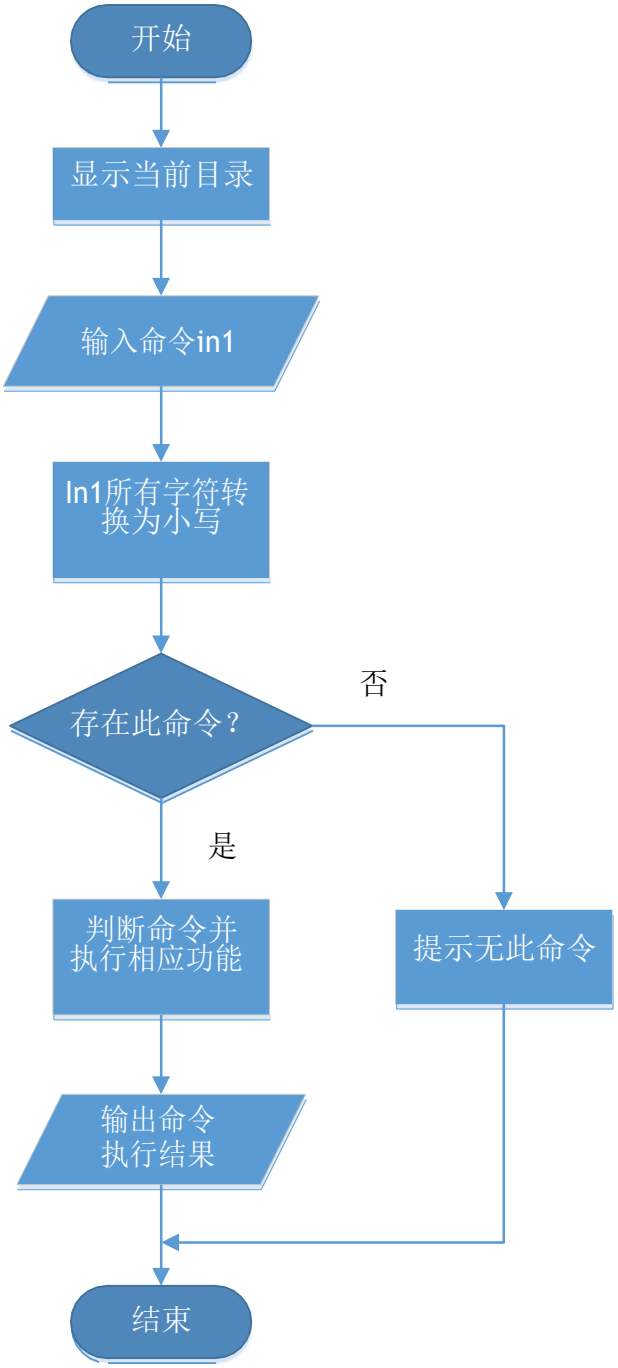


图 3.5.1 display()函数流程

3.6 指令及目录文件名自动补全模块

3.6.1 功能

该模块用于实现用户输入指令、文件或目录名时的自动补全，极大地方便了用户的使用，交互更加简单，系统更加友好。

3.6.1 数据结构

```
string input;     //用户输入的指令或文件/目录名
```

3.6.2 算法

```
5  void cin_name(string &name);//文件/目录名输入
6  void cin_command(string &in);//指令输入
```

cin_command()函数用来接收用户输入的指令，并提供自动补全功能。

当用户输入指令的部分内容，并按下 Tab 键时，系统会去搜索文件系统的指令集，若能唯一确定用户所想输入的指令，则自动补全。

cin_name ()函数用来接收用户输入的文件或目录名，并提供自动补全功能。

当用户输入文件或目录名的部分内容，并按下 Tab 键时，系统会去搜索当前目录下的文件和目录名，若能唯一确定用户所想输入的文件或目录名，则自动补全。

对用户输入的回车、删除、空格等特殊字符均有一定的处理功能。

3.6.3 流程图

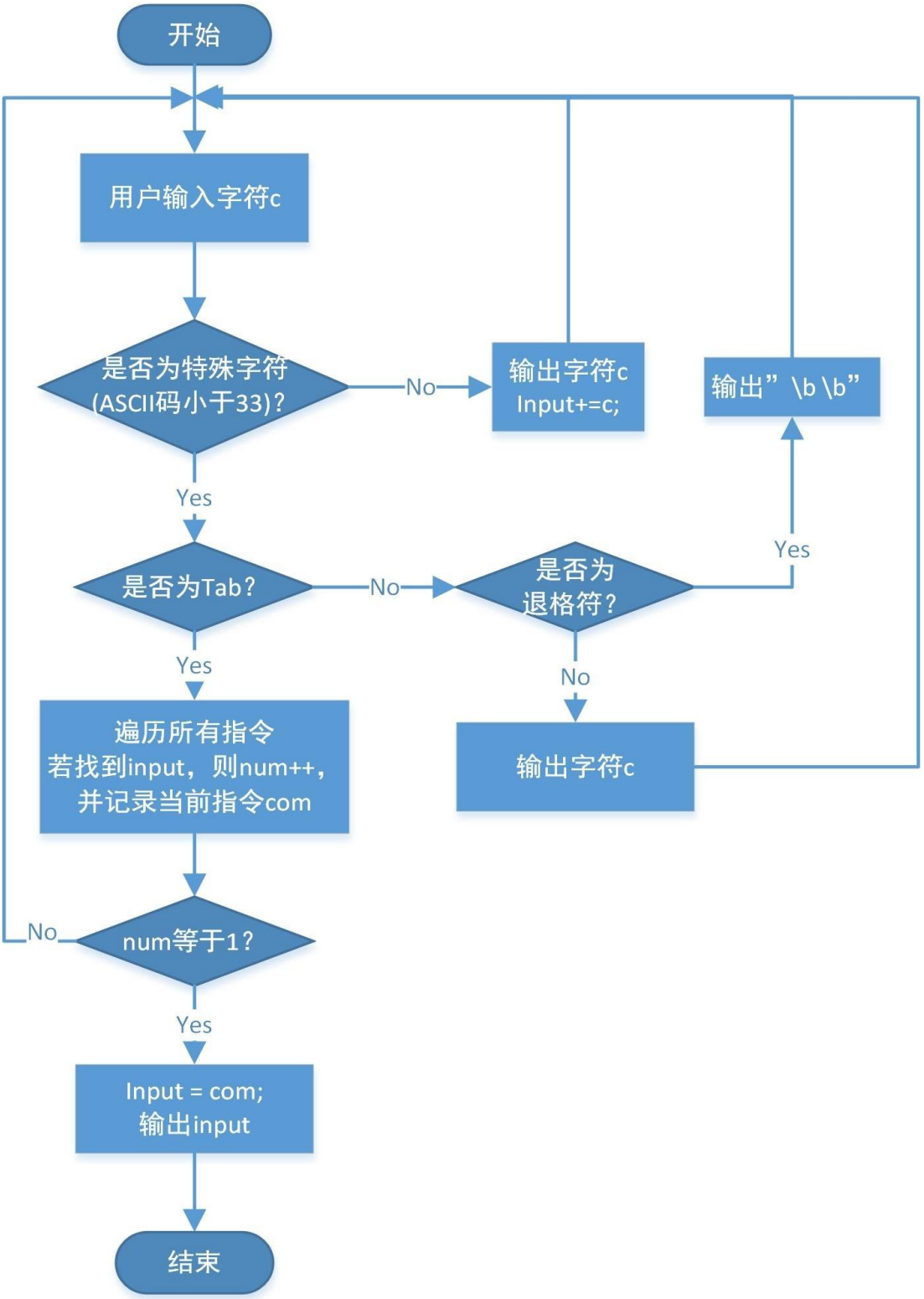


图 3.6.1 指令自动补全流程

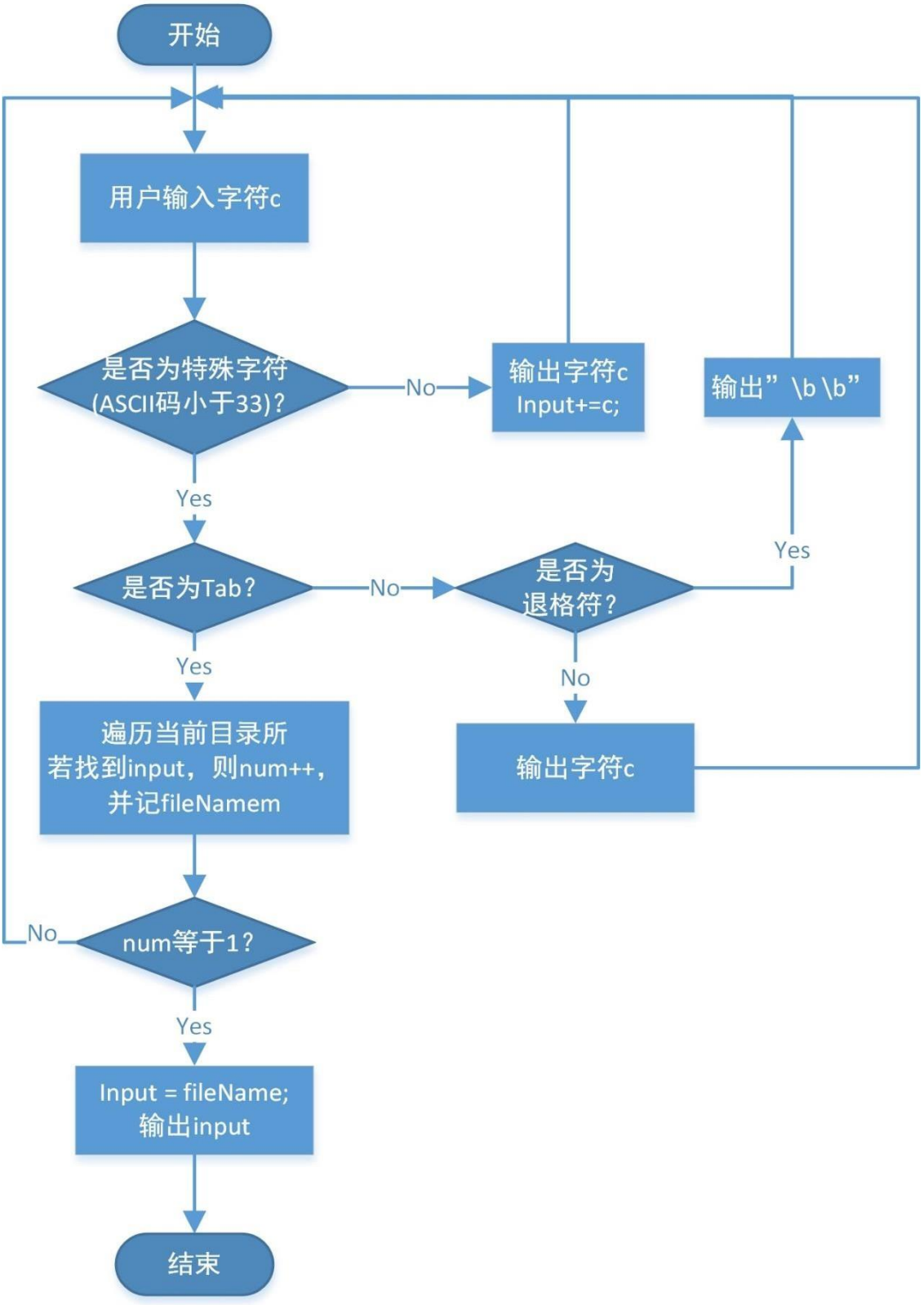


图 3.6.2 文件或目录名自动补全流程

3.7 空闲磁盘块的分配回收模块

3.7.1 功能

该模块在文件和目录创建删除过程中，作为磁盘空间管理方法被调用，在答辩过程中没有解释清楚，在此说明该模块运行方式，当删除文件和目录时，会从已经被占用的磁盘块中释放磁盘空间，当创建文件和目录时，会从空闲的磁盘块中申请磁盘空间，实现磁盘空间管理的方法选的是成组链接法，将未分配的空闲磁盘块，按照成组链接的方法组织起来。

3.7.2 数据结构

```
7  FileSystem fileSystem; //文件系统
8  int iNode[128]; //i结点位势图，0表示占据，1表示空闲
9  int SFDBlock[512]; //目录位势图，0 表示占据，1 表示空闲
```

3.7.3 算法

```
10 void writeABlock(int r); //将装满的栈内容写入一个特定磁盘块
11 void readABlock(int r); //将一个写着空闲磁盘块好的磁盘块内容写入
   栈
12 void freeABlock(int BlockNo); //在成组链接中回收一个空闲磁盘块
13 int allocateOneBlock(); //在成组链接中分配一个空闲磁盘块
```

1.空闲块的组织：把所有空闲盘块按固定数量分组（设为 51），组中第 1 块为“组长”块，第1 组的 51 个空闲块块号放在第 2 组的“组长”块中，第 2 组的51 个空闲块块号放在第 3 组的“组长”块中，以此类推，组、组之间形成链接。最后 1 组的块号（可能不足 51 块）通常放在内存的一个专用栈结构中，对盘块的分配和释放在栈中进行。

2.空闲块的分配：查看超级块（空闲块号栈）中是否 count == 1；若不是，则弹出栈顶元素 r，--count；若是，则弹出栈顶元素 r，把空闲块 r 中的栈（包括栈计数）读入到内存空闲块栈中；返回空闲块编号 r

3.空闲块的分配：被释放空闲块为编号 N。查看超级块中是否栈已满（如 count == 50）；若不是，则 N 入栈，++count；若是，则将超级块中的栈（包括栈计数）写入到空闲块 N，然后把 N 放入内存空闲块栈中的栈顶并置 count 为1。

3.7.4 流程图

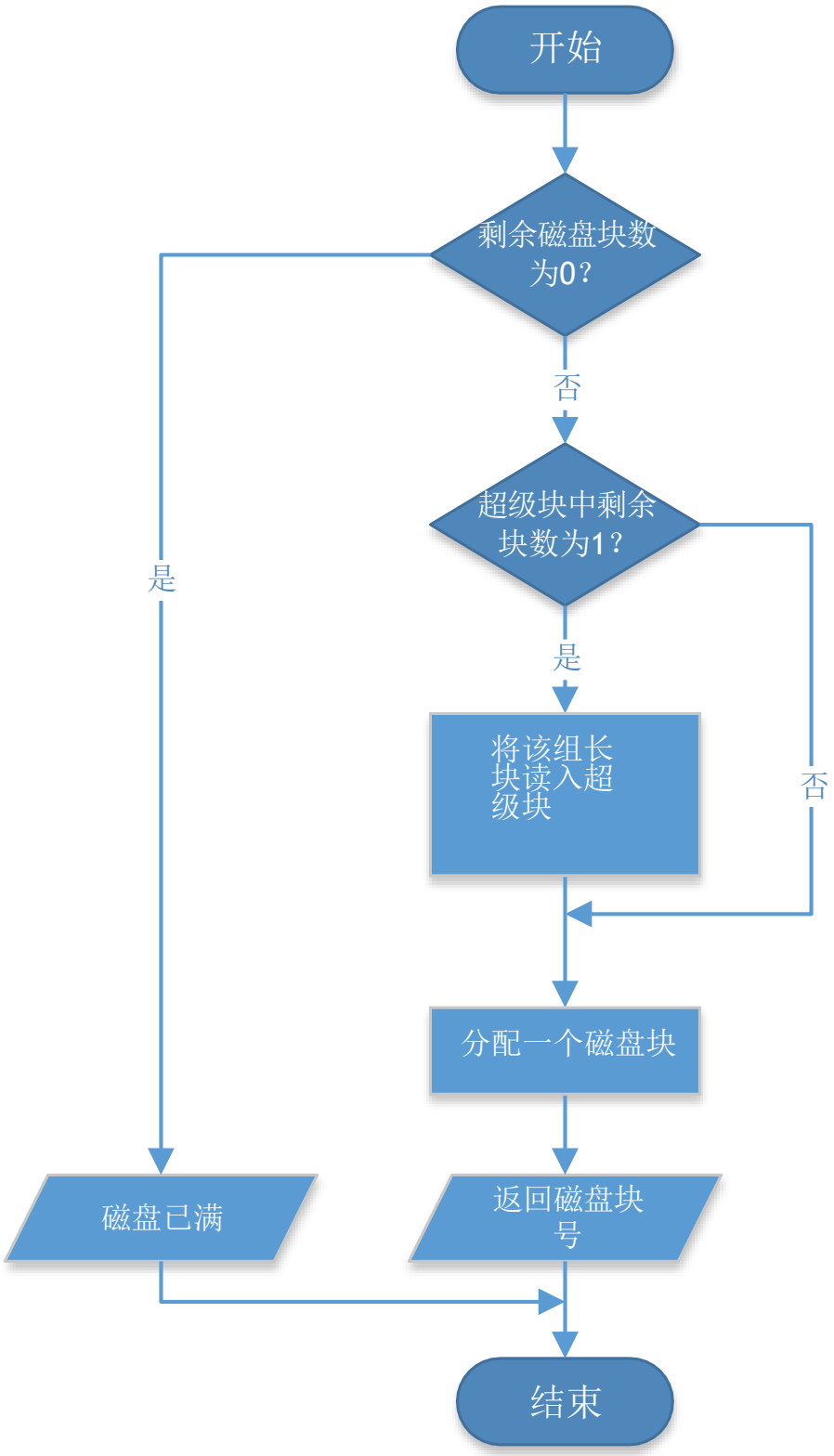


图 3.7.1 空闲磁盘块的分配流程

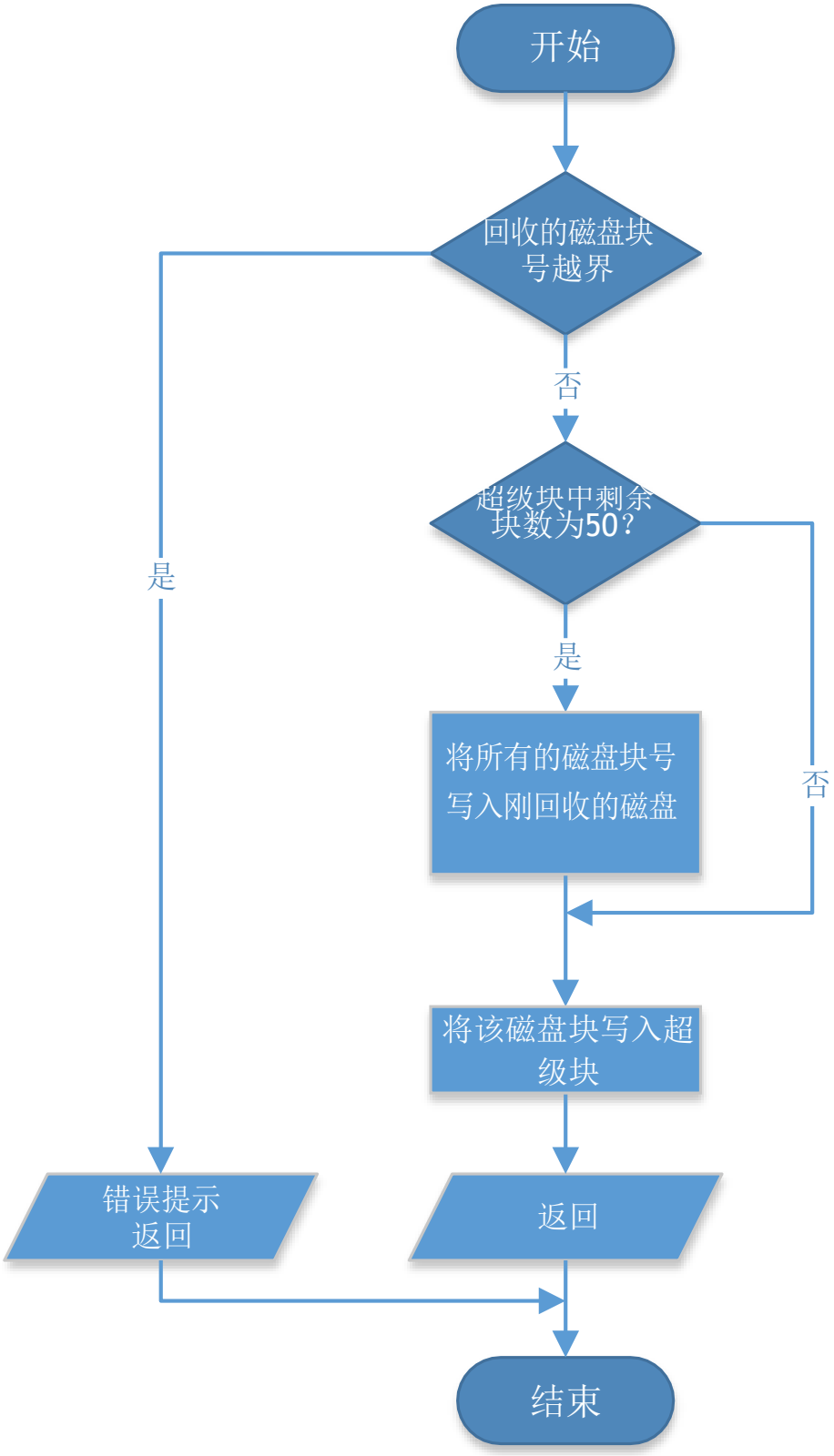


图 3.7.2 空闲磁盘块的释放流程

3.8 创建文件模块

3.8.1 功能

命令：create filename

创建文件模块实现了登陆用户在所在目录下创建一个空文件的功能，并由用户选择是否立即写入内容到文件中。如果输入的 filename 已经存在，则返回创建失败，否则，申请空闲 i 节点和索引块（空闲磁盘块）。然后将文件目录的信息（除文件名外）填入对应的 i 节点，将 i 节点 id 与文件名回填到 SFD 中。

该模块通过调用 allocateOneBlock()和 freeABlock()，实现了空闲块的分配和回收。

3.8.2 数据结构

本模块需要使用的数据结构包括只有文件名和 id 的SFD（结构体）、存储目录中除了文件名以外信息的 i 节点（结构体）、具有512 字节的磁盘块（结构体）。

3.8.3 算法

a.函数

```
//查询当前目录是否存在 name 文件
int checkExitsfd(string name)
//为新创建的文件分配一个 i 结点
int createiNode()
//创建文件的创建索引块
int createFirstIndexB()
//创建文件
int createFile(string name)
//在成组链接中分配一个空闲磁盘块
int allocateOneBlock()
//在成组链接中回收一个空闲磁盘块
void freeABlock(int BlockNo)
```

b.函数间的调用关系

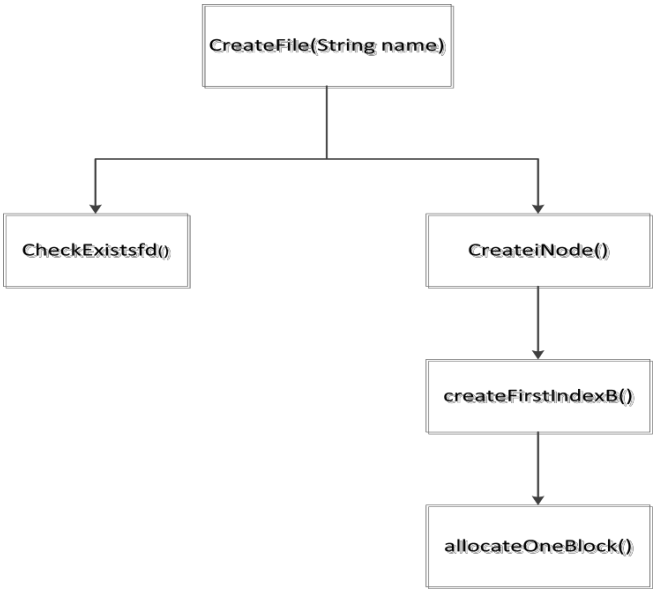


图 3.8.1 createFile()函数流程示意图

c.磁盘资源分派策略

分配方式	优点	缺点
一次性分配	简单	文件不能动态增长
预先分配，不足申请	可以实现文件动态增长	预先分配磁盘块数难以估计
预先分配索引块	可以实现文件动态增长	复杂

表 3.8.1 磁盘资源分配策略

我们采用最后一种分配方式，即只预先分配一个磁盘块作为索引磁盘块，当需要输入文件时，可以再去申请磁盘块，将块号写入具体的索引块。

d.伪码

```
//创建文件
int createFile(string name){

SingleSFD tempSFD;     //临时存放文件信息
 pos = checkExitsfd(name);
if (pos != -1){
    cout << "文件名冲突！\n";
    return 0;
```

```
    }
else
    {   inodeNo = createiNode();//申请i 节点，并初始化
        if inodeNo == -1
        输出没有空闲资源
         return 0;
         else
         tempSFD.id = inodeNo; //将id 与文件名放入临时SFD 项
        tempSFD.name = name;
        将 tempSFD 添加进当前 SFD 的动态数组
        return 1;
    }
//查询当前目录下一固定名的文件下标
int checkExitsfd(string name){
        int pos = -1;
        For i：0 to fileSystem.sfd[cur_SFD].sfdVec.size()//检查当前 SFD 项
          if (fileSystem.sfd[cur_SFD].sfdVec[i].name == name)//如果找到文件名
                pos = i;
                break;
          return pos;   //返回文件在目录中的位置
    }
     //为新创建的文件分配一个 i 结点
     int createiNode(){
        检查是否至少有一个 i 节点和磁盘块；
        if 没有 return -1;
        else
              申请一个磁盘块作为索引块，并将其写入 13 个-1（即初始化为空）；
              申请i 节点，填写其信息（包括存取权限表等）；
             return 申请的i 节点i
```

3.8.4 流程图

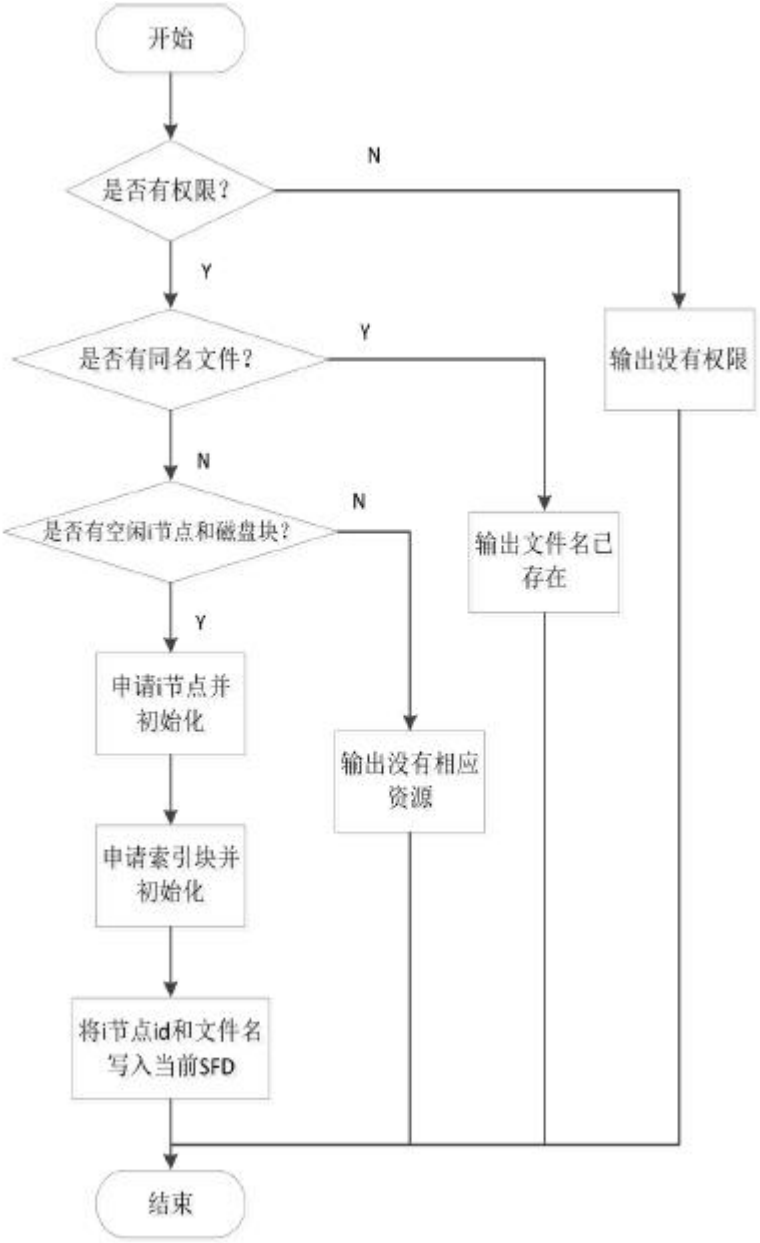


图 3.8.2 创建文件程序流程图

3.9 删除文件模块

3.9.1 功能

命令：delf filename

删除文件模块实现了用户可在当前目录下删除文件的功能。如果输入的文件名不存在，则返回删除失败，否则，检查当前用户是否有删除权限，如果有，删除引用该文件的所有目录下的 SFD 项，并释放文件占用的所有磁盘块和 i 节点。

3.9.2 数据结构

本模块需要使用的数据结构包括只有文件名和 id 的SFD（结构体）、存储目录中除了文件名以外信息的 i 节点（结构体）、具有512 字节的磁盘块（结构体）。

3.9.3 算法

a.函数

```
//删除待删除文件对应的 i 结点及其指向的磁盘块
void deletelNode(int pos)
//遍历删除与待删除文件共享的文件目录
void findSinglesfd(int inodeNo)
//删除指定名字的文件
int freeFile(string name)
```

b.伪码

```
//删除指定名字的文件
    int freeFile(string
    name){ int pos = -1;

        For i：0 to fileSystem.sfd[cur_SFD].sfdVec.size()//检查当前 SFD 项
        if (fileSystem.sfd[cur_SFD].sfdVec[i].name == name)//当前目录下查找文件
                pos = i; //文件在目录中的位置
                break;

       if (pos == -1)

             删除失败文件不存在;
              return 0;

       else
             if   文件引用计数大于 1
```

```
                findSinglesfd(inodeNo);   //找到引用SFD，删除对应项
                deletelNode(pos);//删除i 节点及其下级所有磁盘块
```
删除当前 i 节点所在 SFD 项;
```
                return 1;
        }
```

```
    //遍历删除与待删除文件共享的文件目录
    void findSinglesfd(int inodeNo){
```
遍历 SFD 项;
```
            if (fileSystem.sfd[k].sfdVec[i].id == inodeNo)
```
从动态数组中将其删除
```
    }
```

```
//删除待删除文件对应的 i 结点及其指向的磁盘块
    void deletelNode(int pos){
```
从索引磁盘块中得到索引;

For　i：0 to 13

　　if　索引项不为-1

　　　　if　　当前索引为最顶层索引里的直接索引

　　　　　　删除索引指向的磁盘块的内容并释放磁盘块;

　　　　else　　读出第二级索引

　　　　　　删除索引指向的磁盘块内容并回收;

释放最顶层索引块;
释放 i 节点;
```
}
```

3.9.4 流程图

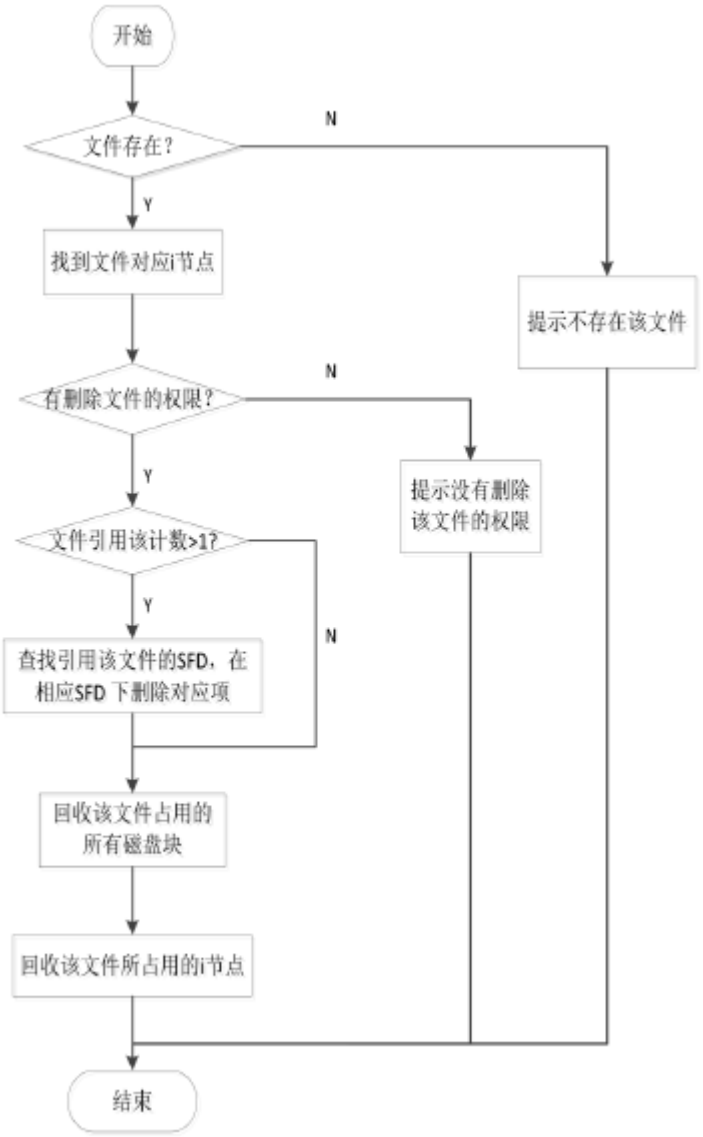


图 3.9.1 删除文件模块流程

3.10 读文件模块

3.10.1 功能

命令：read filename

读文件模块实现了用户可在当前目录下读取文件内容的功能。如果输入的文件名不存在，则返回读文件失败，否则，按根据索引，将文件所占磁盘块中内容依次读出。

3.10.2 数据结构

本模块需要使用的数据结构包括只有文件名和 id 的SFD（结构体）、存储目录中除了文件名以外信息的 i 节点（结构体）、具有512 字节的磁盘块（结构体）。

3.10.2 算法

a.函数

```
//读文件内容函数
void readFile(string name)
//输出文件磁盘块内容
void outputBlock(int blockNO)
```

b.伪码

```
void readFile(string name){

    int pos = checkExitsfd(name);  //查看是当前目录下的第几个文件
    if (pos == -1){
        textcolor(12);
        cout << "文件不存在\n";
    }
    else{
          找到文件 i 节点；
          从i 节点中读取文件索引块号；
          从索引块中读取文件所占磁盘块号；
           outputBlock(int blockNO)；//输出文件所占磁盘块内容
 }
```

3.10.3 流程图

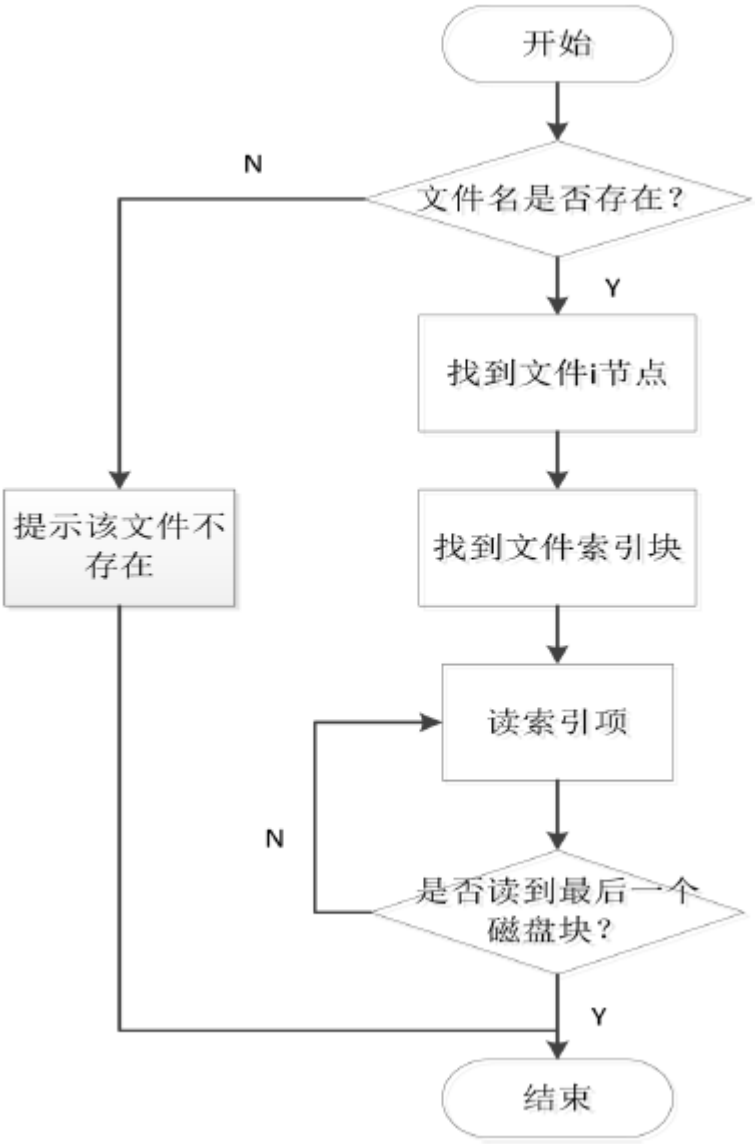


图 3.10.1 读文件程序流程图

3.11写文件模块

3.11.1 功能

命令：write filename

写文件模块实现了可以在任何目录下向指定文件中以手动输入或文件读入的方式写入内容的功能。如果输入的文件名不存在，则返回写文件失败，否则，将文件以单个字符逐一写入磁盘块。其中，要写入的文件不需要预先知道大小，可以多次写文件，增加写文件的灵活性。

3.11.2 数据结构

本模块需要使用的数据结构包括只有文件名和 id 的SFD（结构体）、存储目录中除了文件名以外信息的 i 节点（结构体）、具有512 字节的磁盘块（结构体）。

3.11.2 算法

a. 函 数

```
//写指定文件名的文件
int writeFile(string name)
//返回当前文件所占用的最后磁盘块
int getCur_blockNo(int inodeNo)
//读取索引块中内容
int *getIaddr(int indexnum)
//将数组中的索引内容写回到相应的索引块中，磁盘文件中
void writeiaddr(int BlockNo, int *iaddr)
//将文件占用的磁盘块号写入索引块中
int writeIndexBlock(int indexnum, int BlockNo)
```

b.该模块函数调用关系

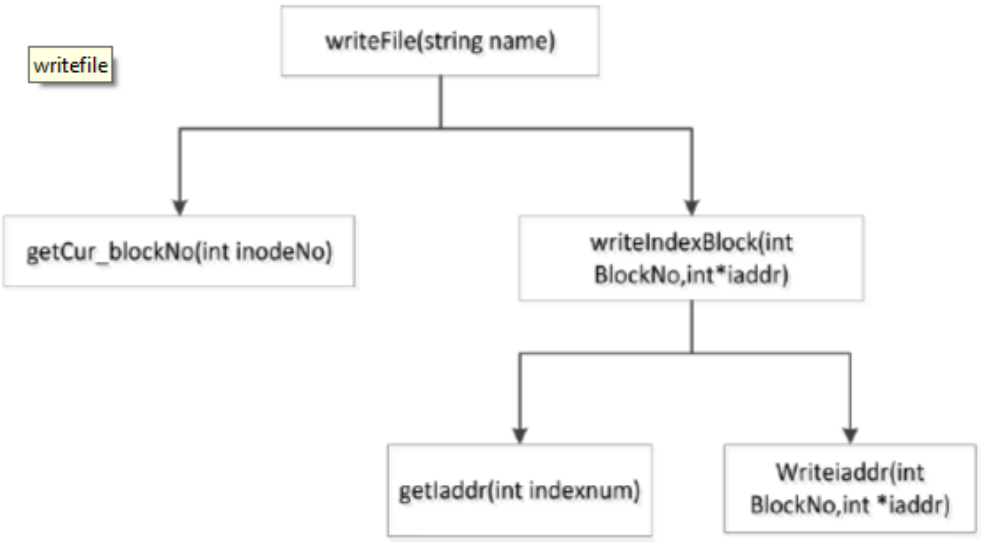


图 3.11.1 写文件函数调用关系

c.外存分配方式比较

分配方式	优点	缺点	是否需要索引
连续分配方式	访问速度快	需要连续磁盘块	不需要
链接分配方式	不需要连续空间	不支持文件随机存取	需要，第一块存放
单级索引分配方式	不需要连续空间 支持随机存取	存放有限磁盘块号，限制文件长度	需要
混合索引分配方式	可存放大文件	复杂	需要

表 3.11.1 外存分配方式比较

综合考虑上述分配方式，我们采用混合索引分配方式，文件的一级索引块中存放13 个磁盘快号，其中 0-9 为直接索引，直接存放文件所占磁盘块号。10-12 为间接索引，存放二级索引块号，每个二级索引块可以存放 128 个磁盘块号。使用这种混合索引方式基本上可以满足足够大的文件的存放问题。

d.伪码

```
//写指定文件名的文件
int writeFile(string name){          //应该先输入文件名字再去找文件所对应的i 结点，这
int pos = checkExitsfd(name);       //检查当前目录下有没有该文件
用户选择
文件输入 -------1
```

```
手动输入 -------2
If 手动输入 从键盘读取存到字符串 s
Else 从文件中读取，存入字符串s
 getCur_blockNo(inodeNo);                    //返回当前文件所对占的最后一块磁盘块
 填满最后一块磁盘块
If 文件全部存入，则写过程完成
Else  cur_blockNo = allocateOneBlock();  //分配磁盘块
       writeIndexBlock(indexnum, cur_blockNo);//写入索引块中
     If  该块填满，文件依然未读完，继续分配磁盘块，并写入索引块中
}



//返回当前文件所占用的最后磁盘块
int getCur_blockNo(int inodeNo){
 从文件i 节点中找到文件索引块号 indexnum
int *iaddr = getIaddr(indexnum);    //得到索引块中存放的内容
For i=0:10
      if (iaddr[i] == -1)
      return iaddr[i - 1];     //最后一块以-1 结束,直接返回文件所占最后一个磁盘块
      For i=11:13
      根据二级索引返回文件所占最后一个磁盘块
      }


//将文件占用的磁盘块号写入索引块中
int writeIndexBlock(int indexnum, int BlockNo){
    int *iaddr = getIaddr(indexnum);          //读出索引块内容
    For i=0:9
      if (iaddr[i] == -1)
      iaddr[i] = BlockNo; //将块号放入临时变量iaddr[]中
```

```
      writeiaddr(indexnum, iaddr); //将iaddr 内容写入索引块中
    For j=10:13
if (iaddr[j] == -1){      //前一索引已满，只能从当前新建的二级索引
     iaddr[j] = cur_B;                //BlockNo 是要放入的索引中的块号
    writeiaddr(indexnum, iaddr);
     int cur_B = allocateOneBlock();  //分配磁盘块存放二级索引
     fileSystem.diskBlock[BlockNo].strNum = 128;//最多存放128 个块号
     int iaddr_all1[128];  //临时存放128 个块号
     For i=0:128
      iaddr_all1[i] = -1;  //初始化索引块
    iaddr_all1[0] = BlockNum;              //iaddre_all1[0]中存放的是二级索引块块
号
    writeiaddr(cur_B, iaddr_all1);
     }
else{                                 //否则直接在二级索引中找相应的空位写入当前块号
     int *iaddr_1 = getIaddr(iaddr[j]);
     for (int i = 0; i < 128;
        i++){ if (iaddr_1[i] == -
        1){
            iaddr_1[i] = BlockNo;
            writeiaddr(iaddr[j], iaddr_1);
}

//读取索引块中内容
int *getIaddr(int indexnum){
将索引块中该内容存放在临时变量iaddr
return iaddr;
}
```

3.11.3 流程图

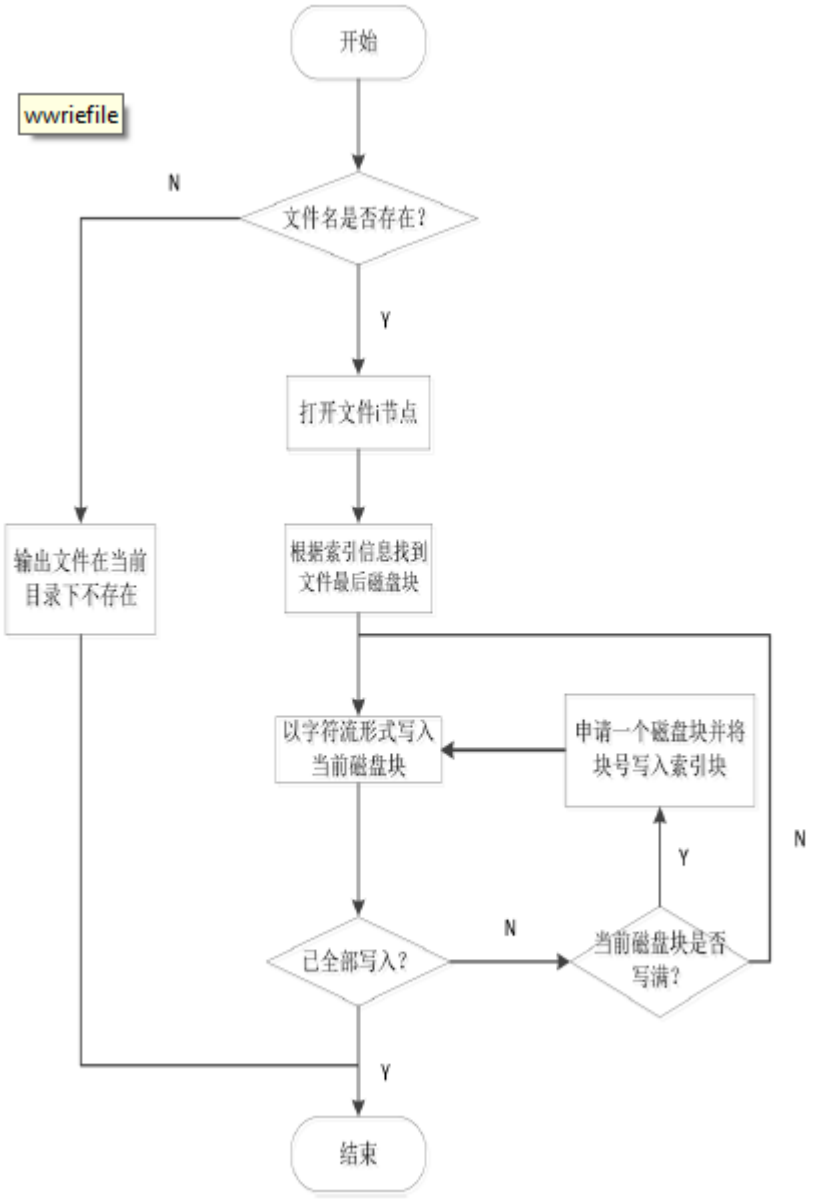


图3.12.2 写文件程序流程图

3.12 创建删除目录模块

3.12.1 功能

该模块实现了目录创建及删除，目录跳转的功能。

创建目录时会检测判断当前目录下是否有重名目录，若已存在则提示错误信息“当前目录下已存在该目录”，若不存在重名，则成功创建。对目录的删除操作是级联的，即删除当前目录及当前目录下的所有目录结构以及文件。在整个创建和级联删除目录的过程中，需要对用户访问权限的控制检查，如果不具有权限，则提示错误信息“无权限，创建和删除失败”，当具有权限时还需要对目录结构和磁盘结构进行分配和回收，以便其他文件的使用，保证整个模拟文件系统的正常运行。

3.12.2 数据结构

```
17  string user; //全局变量，标志当前正在访问文件系统的用户
18  FileSystem fileSystem; //文件系统
19
20  int iNode[128]; //i结点位势图，0表示占据，1表示空闲
21  int diskBlock[512]; //磁盘块位势图，0表示占据，1表示空闲
 6  int SFDBlock[512]; //目录位势图，0 表示占据，1 表示空闲
```

3.12.3 算法

在该模块中主要用到了以下几个函数：

```
22  void showSFD(); //展示SFD
23  void createInitINode(int useINode, int type, int filelen);
24  //为创建文件或目录初始化i结点
25  int createDir(string filename); //创建一个目录
26  void deleteINodeOne(int useINode);//删除一个i结点
 6  int deleteDir(string name);//级联删除一个目录及其子目录和子文件
 7  int goNextDir(string filename); //进入下一级目录
```

当选择创建目录或是删除目录时同样要进行权限检查，当不具有权限时直接

返回失败，当具有权限时在执行相应操作。

当创建目录时，首先判断空闲的 i 结点数和空闲的磁盘快数是否满足需求，不满足时直接返回失败，创建目录主要是对所分配出的空闲目录结构和 i 结点结构的一个初始化的过程。

当删除目录时，必须进行级联删除，将所删除目录的所有下级目录及文件全部删除掉。在这个过程中，采用递归删除的方法，当遇到文件时调用删除文件模块的相应函数，当遇到子目录时，调用删除目录的函数，一级一级直到删除彻底为止，在整个删除过程中，需要对目录结构的 SFD 和i 结点，以及文件删除之后的磁盘块的回收作相应处理，并且调用模块的函数来更新成组链接和位势图中的空闲磁盘块以及i 结点目录信息。

3.12.4 流程图

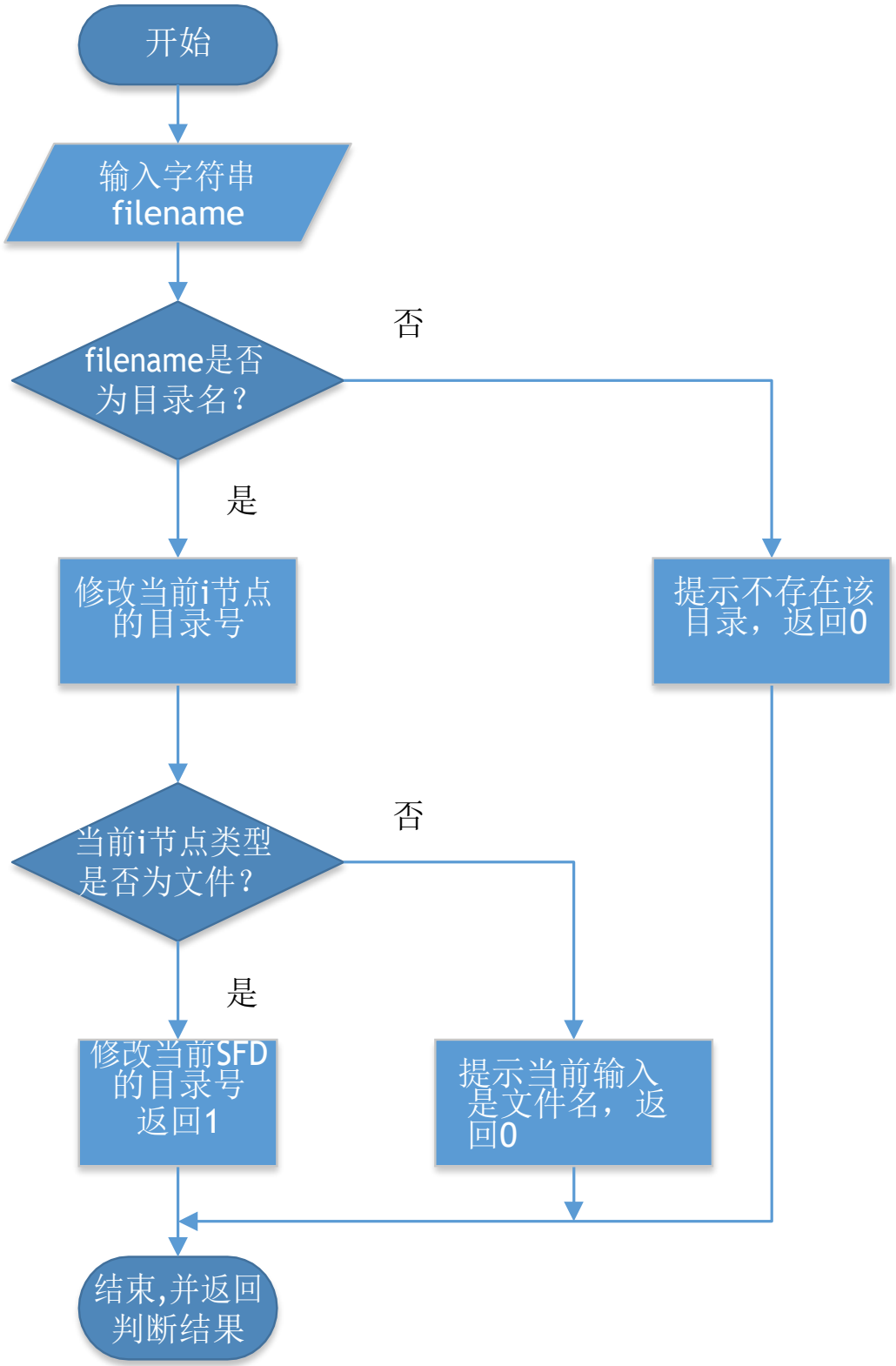


图 4.3.1 goNextDir()函数算法流程

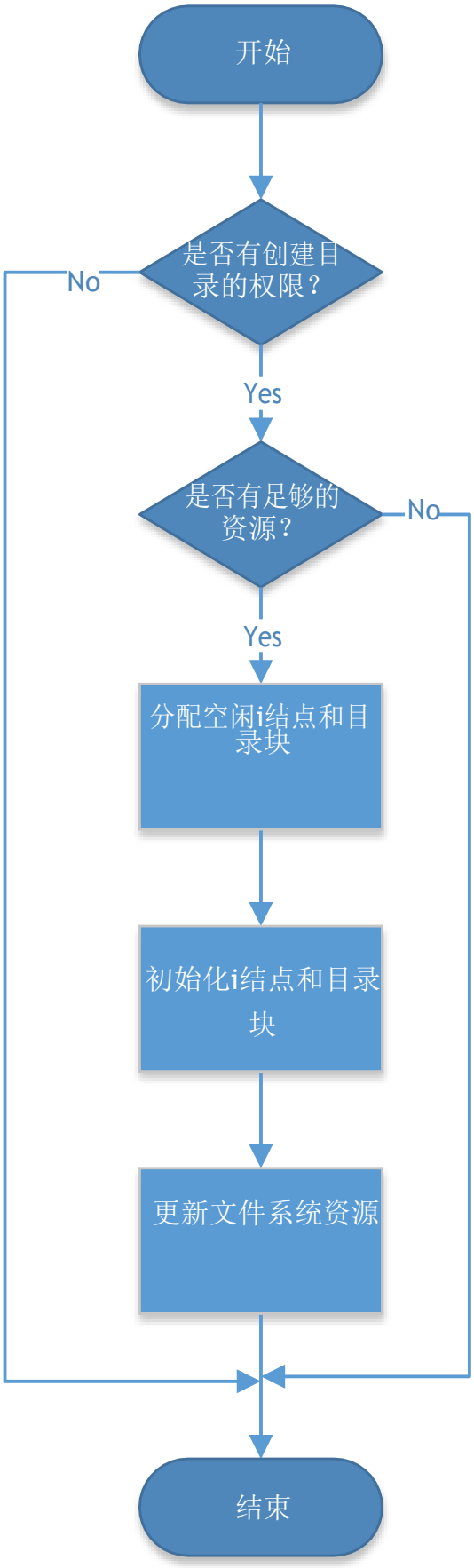


图 3.12.1 创建目录函数流程

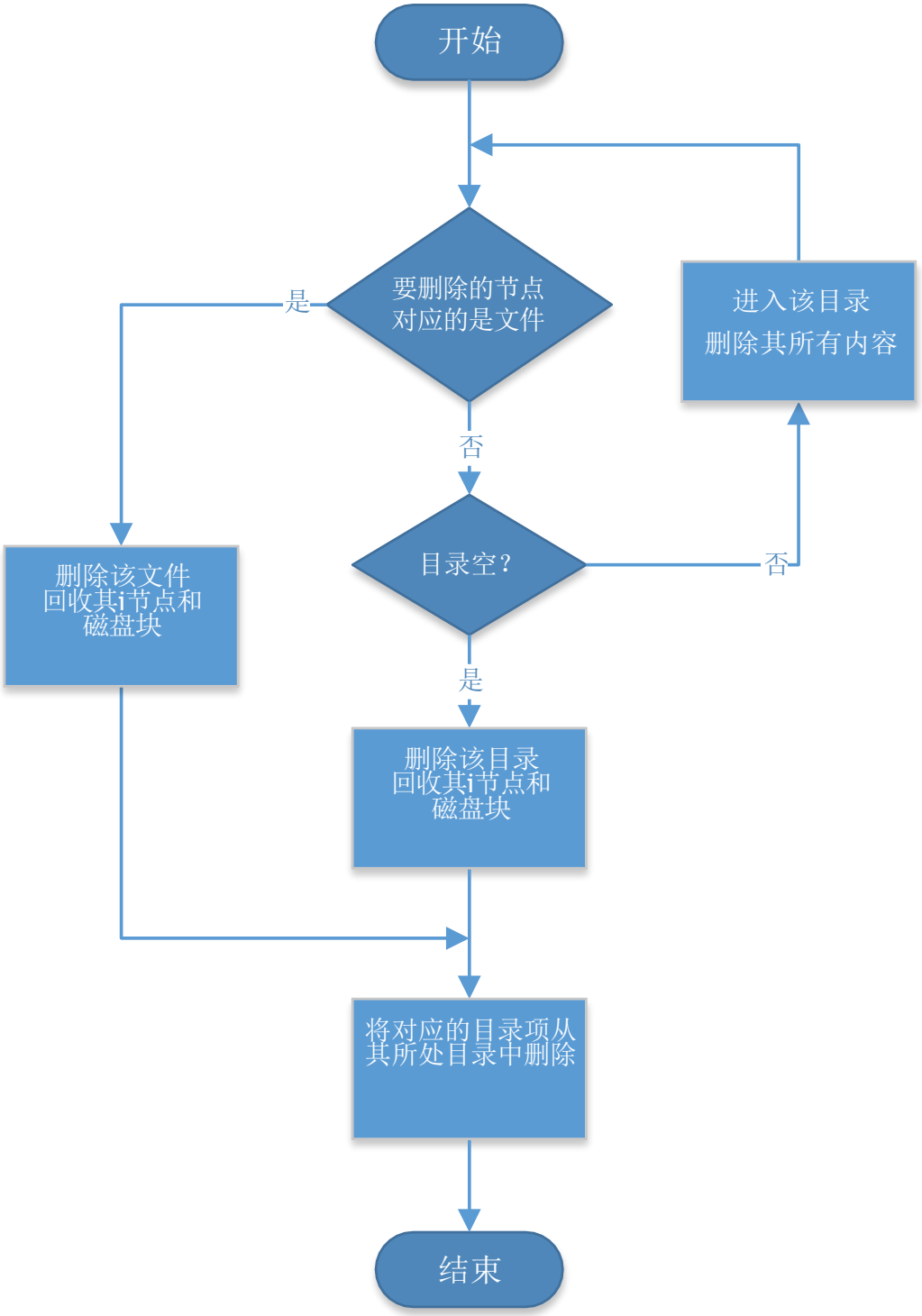


图 3.12.2 删除目录函数流程

3.13 复制剪切链接模块

3.13.1 功能

复制、剪切和链接（快捷方式）是文件系统操作中经常用到的功能。复制会完全拷贝一份文件或目录存入指定的目录下；剪切时会把源文件删除；链接后会有多个SFD 指向同一个i 结点。

如果文件或目录在复制之后被用户删除，则粘贴板会清空；如果要粘贴的文件或目录和粘贴板中的文件或目录有重名，则不允许粘贴；如果当前粘贴板无文件或目录，则不允许粘贴操作并提示错误警告。

删除链接时会有提示信息，询问仅删除当前链接，还是将整个文件及其所有链接删除。

3.13.2 数据结构

```
27 struct SingleSFD{//单个目录项
28     string name;    //文件名
29     int id; //文件对应的i结点的id号
30 };
31
 6 int copy_flag;              //粘贴板标志
 7 SingleSFD copySFD;    //粘贴板
```

3.13.3 算法

```
1 int copyContext(string filename); //复制指定名字的目录或文件
2 int linkContext(string filename); //链接指定名字的目录或文件
3 int cutContext(string filename);          //剪切指定名字的目录或文件
4 int pasteContext(int Create); //粘贴粘贴板上的内容到当前目录下
```

三个函数均使用一个全局的 SingleSFD 结构变量 copySFD 作为文件系统的粘贴板，对这个公有的变量进行赋值、修改和删除。

copyContext(string filename)从指定的源文件或源目录获得它们SFD 的id 和name，把它们存入粘贴板 copySFD。当跳转到指定目录下，pasteContext()会根据粘贴板内的 name 创建一个新的文件或文件夹，然后根据粘贴板内的 id 信息将文件或目录内容完全拷贝过来。

linkContext(string filename)从指定的源文件或源目录获得它们SFD 的id

和 name，把它们存入粘贴板 copySFD，当跳转到指定目录下，pasteContext()把粘贴板内的 id 和name 信息存入目标目录下，当删除链接时会显示提示信息，选择仅删除该链接，还是彻底删除该文件及所有链接。

剪切操作时会先保存源文件的 SFD，并把源文件的 SFD 删除，粘贴时直接在相应的目录下添加进已保存的 SFD 即可完成剪切操作，即 i 结点和磁盘还是源文件的原始 i 结点和磁盘。

3.13.4 流程图

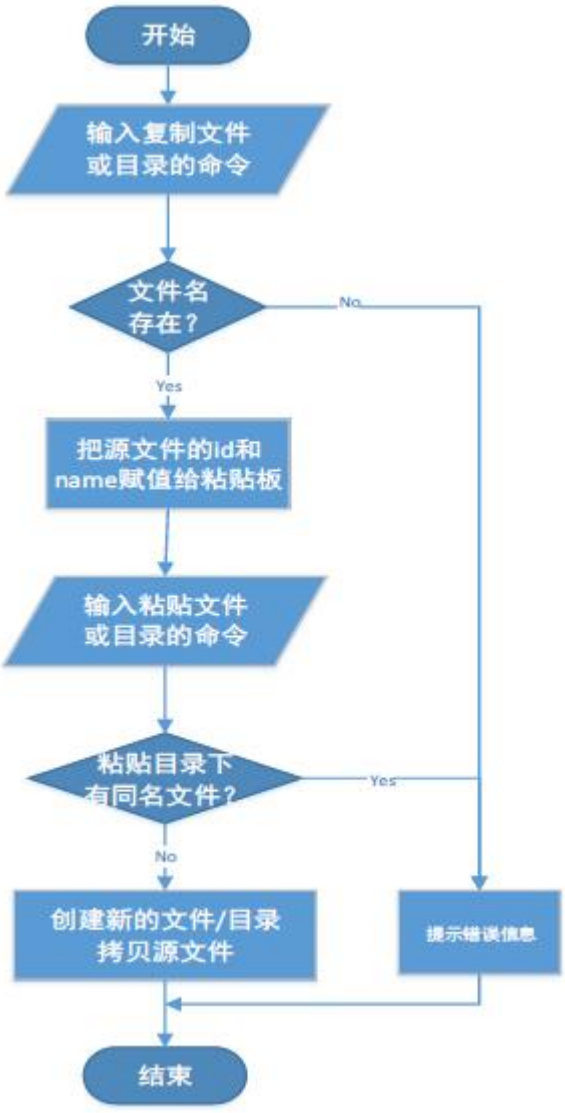


图3.13.1　复制粘贴操作流程

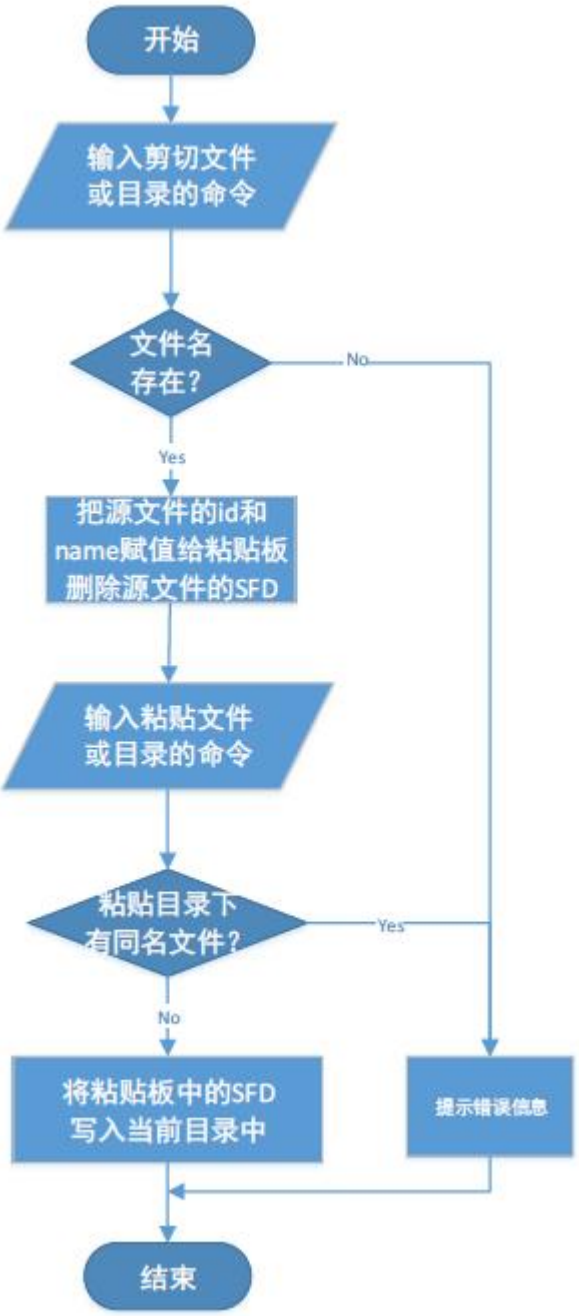


图3.13.2　剪切操作流程

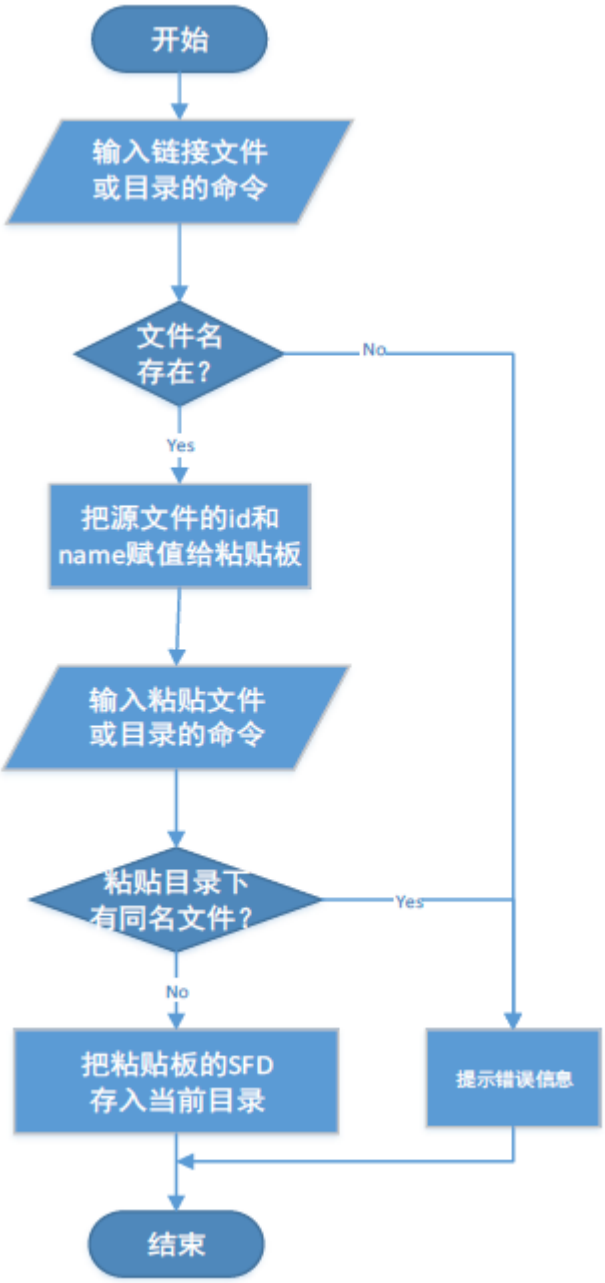


图3.13.3　创建快捷方式操作流程

3.14 查找和重命名模块

3.14.1 功能

查找指令：可以搜索出当前目录及子目录中指定名字的所有文件或目录，及其路径信息。

重命名指令：可以对文件或目录重新进行命名。

3.14.2 数据结构

```
1  struct SingleSFD{//单个目录项
2      string name;    //文件名
3      int id; //文件对应的i结点的id号
4  };
6  struct SFD{                              //目录结构
7      int sfdNum; // 动 态 数 组 的 大 小
8      vector<SingleSFD> sfdVec; //目录下的sfd数组
9  };
```

3.14.3 算法

```
1  void find_fd(string cur_path, string name);//查找文件或目录
2  void rname_fd(string old_name, string new_name);//重命名文件或目录
```

find_fd(string cur_path, string name)函数会遍历当前目录，以及当前目录下的子目录，查找名为name的文件或目录，并显示出他们的具体路径信息。

3.14.4 流程图

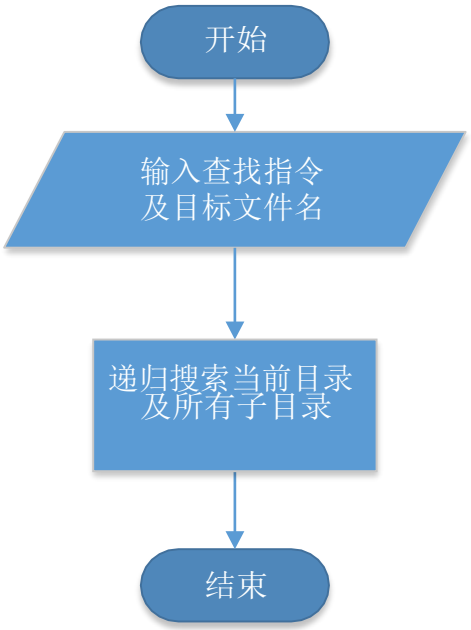


图3.14.1　查找操作流程

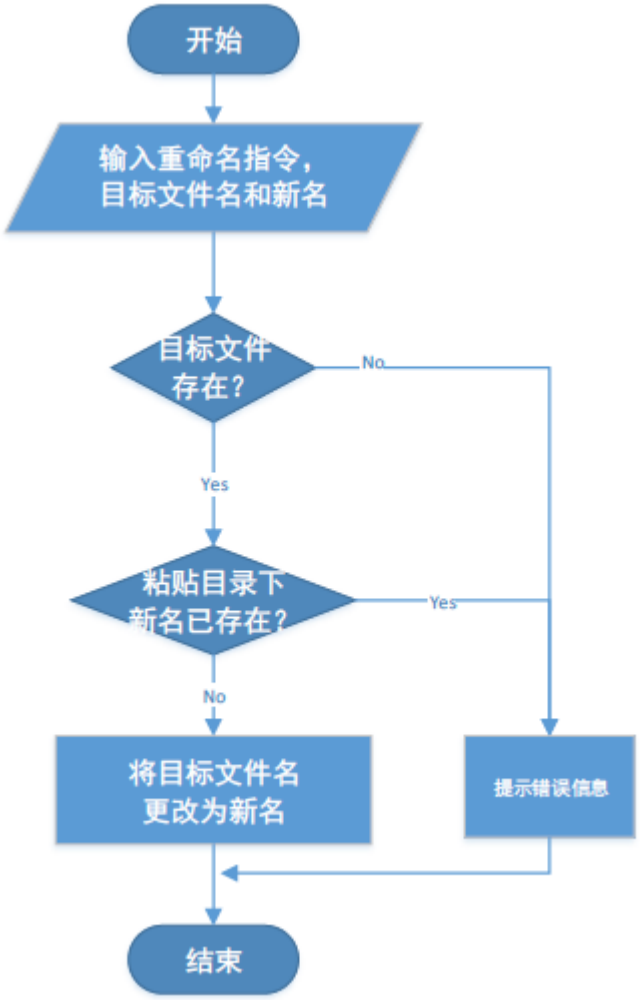


图3.14.2　重命名操作流程

3.15 格式化模块

3.15.1 功能

该模块可以实现对当前登录用户的所有文件及其目录结构的彻底删除，以及对相应i 结点、目录项、磁盘块等模拟文件系统的资源回收功能。

3.15.2 算法

该模块只用到了一个函数：

```
void formatUser();  // 用 户 的 格 式 化
```

通过该函数在该用户的根目录下遍历所有子目录结构，进行文件的删除和目录的级联删除就可以实现当前用户的初始化功能了。此外，在界面函数 display() 中， 清空界面显示的所有状态和内容。

3.16 退出保存模块

3.16.1 功能

该模块在文件系统程序执行结束退出之前需要自动执行的部分，对模拟文件系统的一些重要结构——超级块、i 节点、磁盘块和 SFD 进行使用后的保存，使其能在下次程序运行时将本次操作之后的所有结果重新装入模拟文件系统的数据结构中，以使得模拟文件系统具有持久性。

3.16.2 算法

在本模块中所涉及到的函数如下：

```
void exitSFD();                 //退出前保存 SFD 内容
void exitDiskBlock();           //退出前保存磁盘块内容
void exitINode();               //退出前保存 i 节点内容
void exitSuperBlock();          //退出前保存超级块内容
void exitSystem();              //退出系统
```

其中各个函数分别对超级块、I 结点、磁盘块和目录结构的各个数据域进行

保存的过程，每个功能函数中首先打开存储相应数据的 txt 文件，之后将相应数据域中的内容从模拟文件系统的数据结构中写回到文件中，实现对整个模拟文件系统的所有操作之后内容的保存功能。

3.16.3 流程图

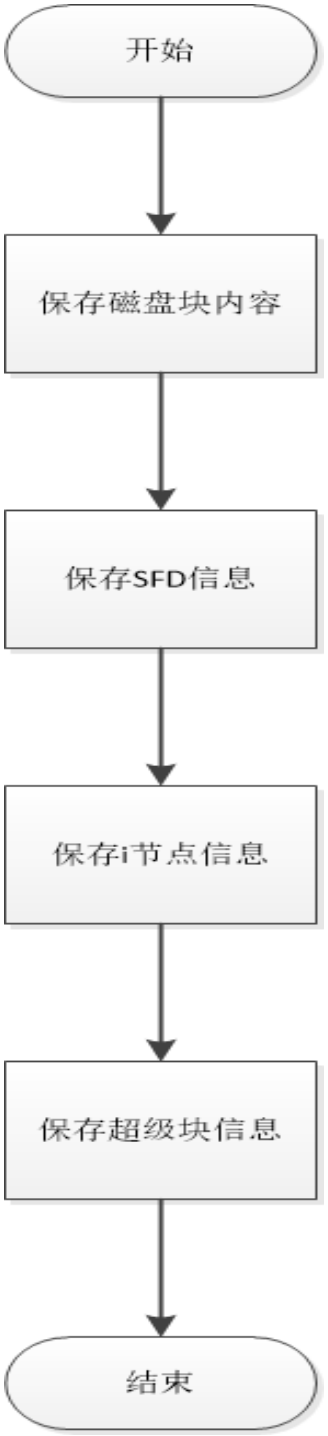


图3.16.1　格式化操作流程

4 程序设计及实现

4.1 程序流程图

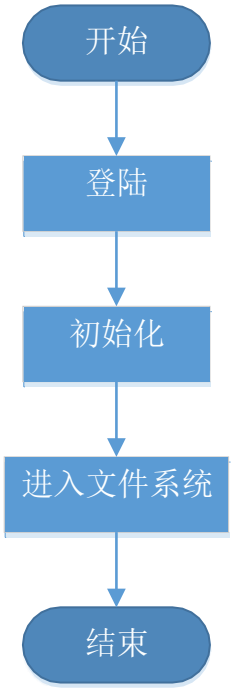


图 4.1 程序总体流程图（main 函数流程）

其余流程图已包含在各模块中。

4.2 程序说明

程序运行环境：

程序的开发环境	**Windows10** 家庭中文版
开发环境为	Microsoft Visual Studio 2017

程序中用到的所有函数及其说明如下：

1	//*************************初始化模块**************************
2	void initSuperBlock(); //初始化超级块
3	void initINode(); // 初 始 化 i 结 点
4	void initDiskBlock(); //初始化磁盘块
5	void initSFD(); // 初 始 化 SFD
6	void init(); // 初 始 化
7	
8	//*************************用户模块***************************
9	void login(); // 登 录 用 户 名
10	void logout(); // 登 出
11	int checkUser(string user); //判断此时用户
12	int getInodeNum(); //获得当前目录的i结点号
13	int checkFileAuth(string filename); //检查当前用户的读写权限
14	int checkDirAuth(string filename); //检查当前用户对目录的读写权限
15	
16	//*************************界面模块***************************
17	void dis_help(); //显示帮助
18	void display(); //界面主函数，用来实现大部分输入输出功能
19	void textcolor(int color); //设置字体颜色
20	int checkIn(string in); //用来检测命令是否存在以及指令的种类
21	
22	//*************************自动补全模块**************************
23	void cin_name(string &name);//文件/目录名输入
24	void cin_command(string &in);//指令输入
25	
26	//*******************空闲磁盘块分配释放模块**********************
27	void writeABlock(int r); //将装满的栈内容写入一个特定磁盘块
28	void readABlock(int r); //将一个写着空闲磁盘块好的磁盘块内容写入栈
29	void freeABlock(int BlockNo); //在成组链接中回收一个空闲磁盘块
30	int allocateOneBlock(); //在成组链接中分配一个空闲磁盘块
31	
32	//******************文件的创建与删除模块*************************
33	int createFirstIndexB(); //创建文件的第一个索引块
34	int createiNode(); //为新创建的文件分配一个i结点
35	int checkExitsfd(string name); //查询当前目录下一固定名的文件下标
36	int createFile(string name); //创建文件
37	int *getIaddr(int indexnum); //得到待删除文件的索引块中的磁盘

```
    块号数组
38  int freeFile(string name); //删除指定名字的文件
39  void deleteINode(int pos); //删除待删除文件对应的i结点及其指
    向的磁盘块
40  void findSinglesfd(int inodeNo); //遍历删除与待删除文件共享的文件
    目录
41
42  //******************文件的读写模块*****************************
43  void writeiaddr(int BlockNo, int *iaddr);//将索引内容写回相应的索
    引块，磁盘文件中
44  void outputBlock(int blockNO); //输出文件磁盘块内容
45  void readFile(string name); //读文件内容函数
46  int writeIndexBlock(int indexnum, int BlockNo);//文件内容的写入
47  int getCur_blockNo(int inodeNo);//返回当前文件i节点，所占用的最后
    磁盘块
48  int writeFile(string name); //写指定文件名的文件
49
50  //******************目录的创建删除模块**************************
51  void showSFD(); //展示SFD
52  void createInitINode(int useINode, int type, int filelen);//为创建
    文件或目录初始化i结点
53  int createDir(string filename); //创建一个目录
54  void deleteINodeOne(int useINode);//删除一个i结点
55  int deleteDir(string name);//级联删除一个目录及其子目录和子文件
56  int goNextDir(string filename); //进入下一级目录
57
58  //****************复制粘贴模块*****************************
59  int copyContext(string filename); //复制指定名字的目录或文件
60  int cutContext(string filename);          //剪切指定名字的目录或文件
61  void find_filedir(string cur_path, string name);//查找文件
62  int pasteContext(); //粘贴粘贴板上的内容到当前目录下
63
64  //******************格式化模块*****************************
65  void formatUser(); // 用 户 的 格 式 化
66
67  //**************************退出模块*****************************
68  void exitSFD(); // 退 出 前 保 存 SFD 内 容
69  void exitDiskBlock(); //退出前保存磁盘块内容
70  void exitINode(); //退出前保存i节点内容
71  void exitSuperBlock(); //退出前保存超级块内容
72  void exitSystem(); // 退 出 系 统
```

4.3 实验结果

运行结果截图如下：

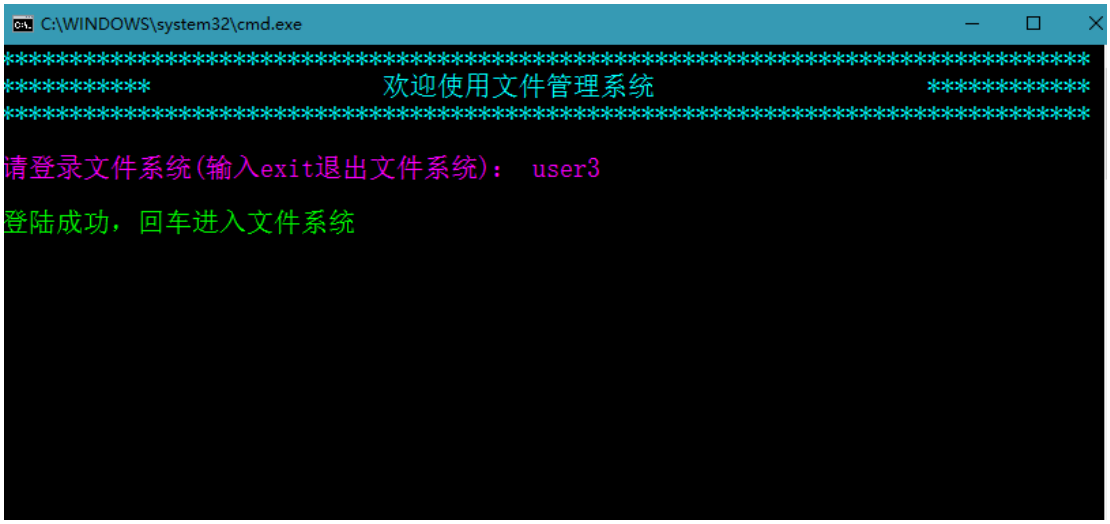


图 4.3.1 用户 user3 登陆截图

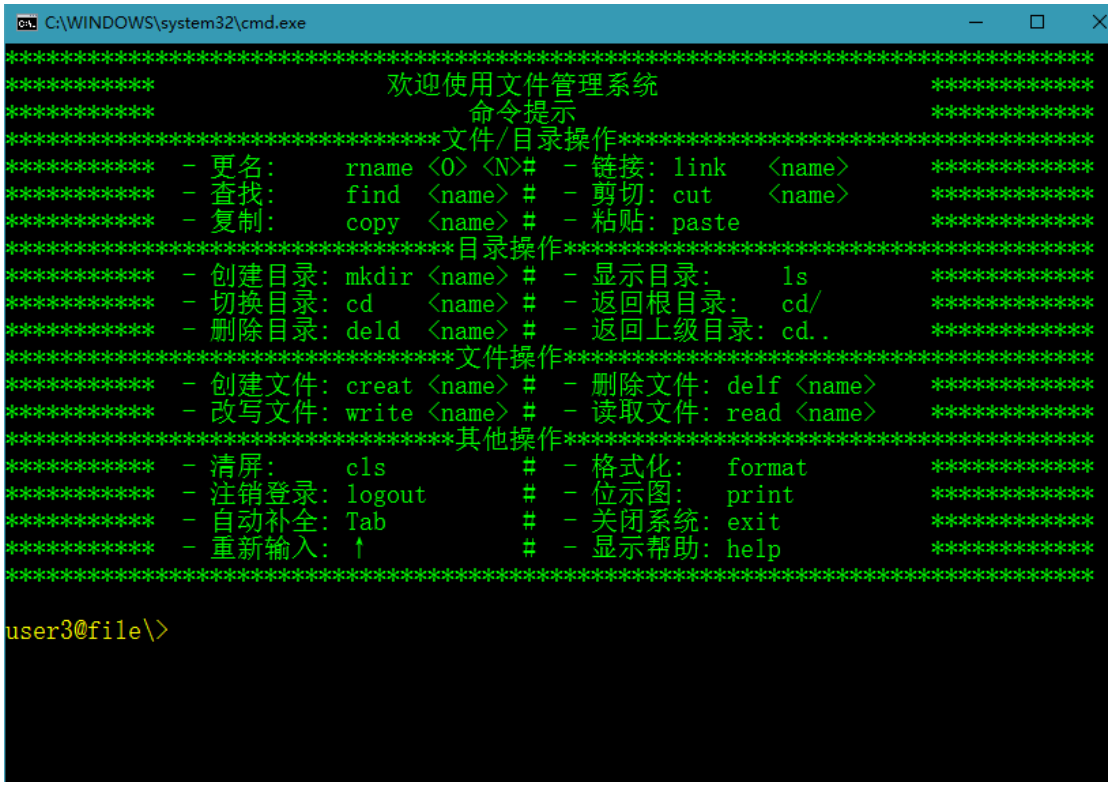


图 4.3.2 登陆之后主界面

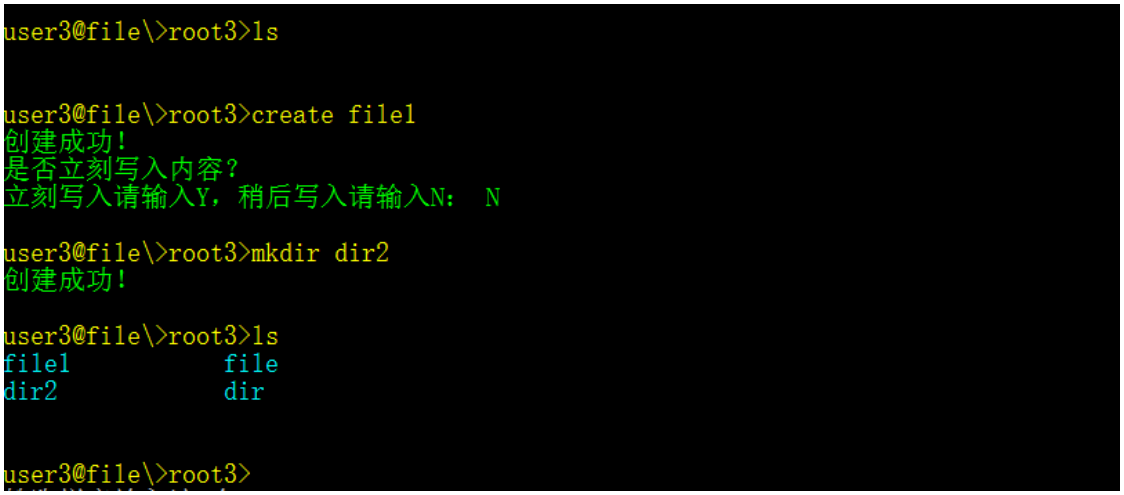


图4.3.3 文件和目录的创建

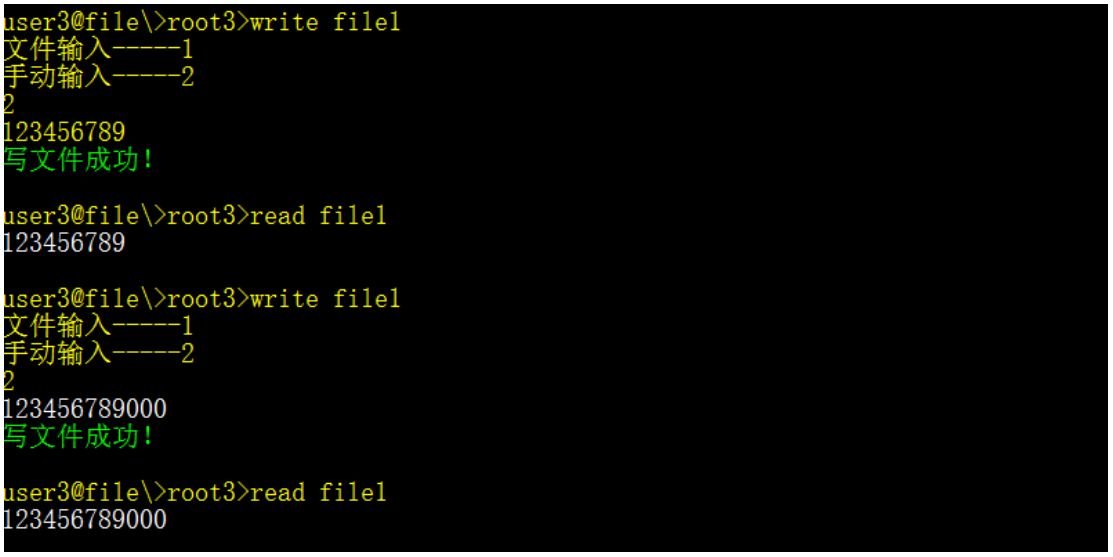


图4.3.4 文件的写入和读取

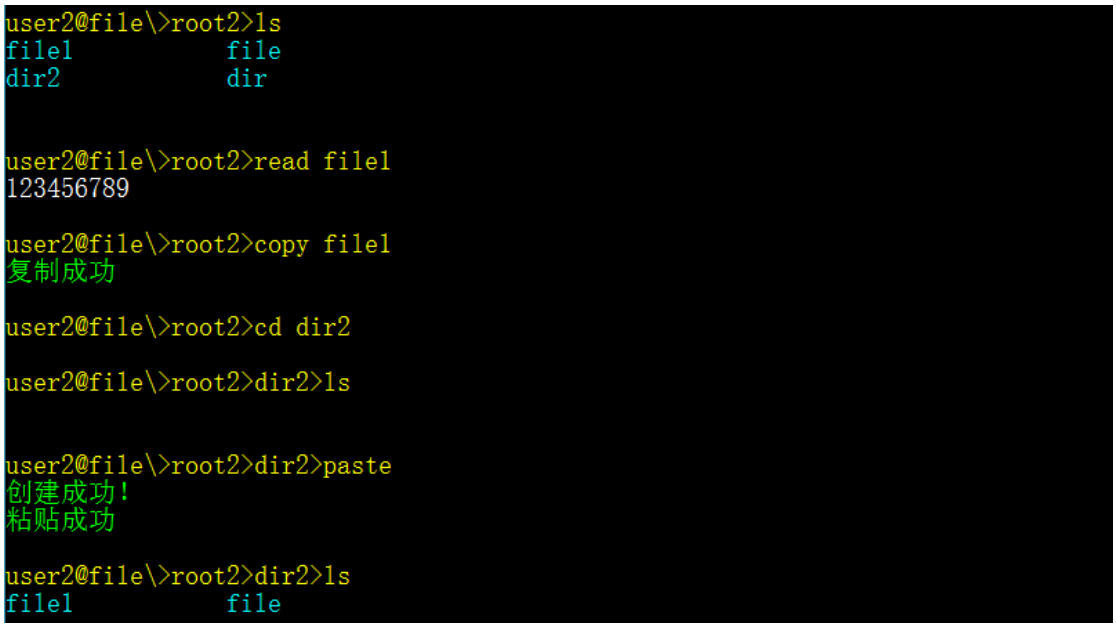


图 4.3.5 文件的复制和粘贴

图 4.3.6 复制文件的删除（源文件不受影响）

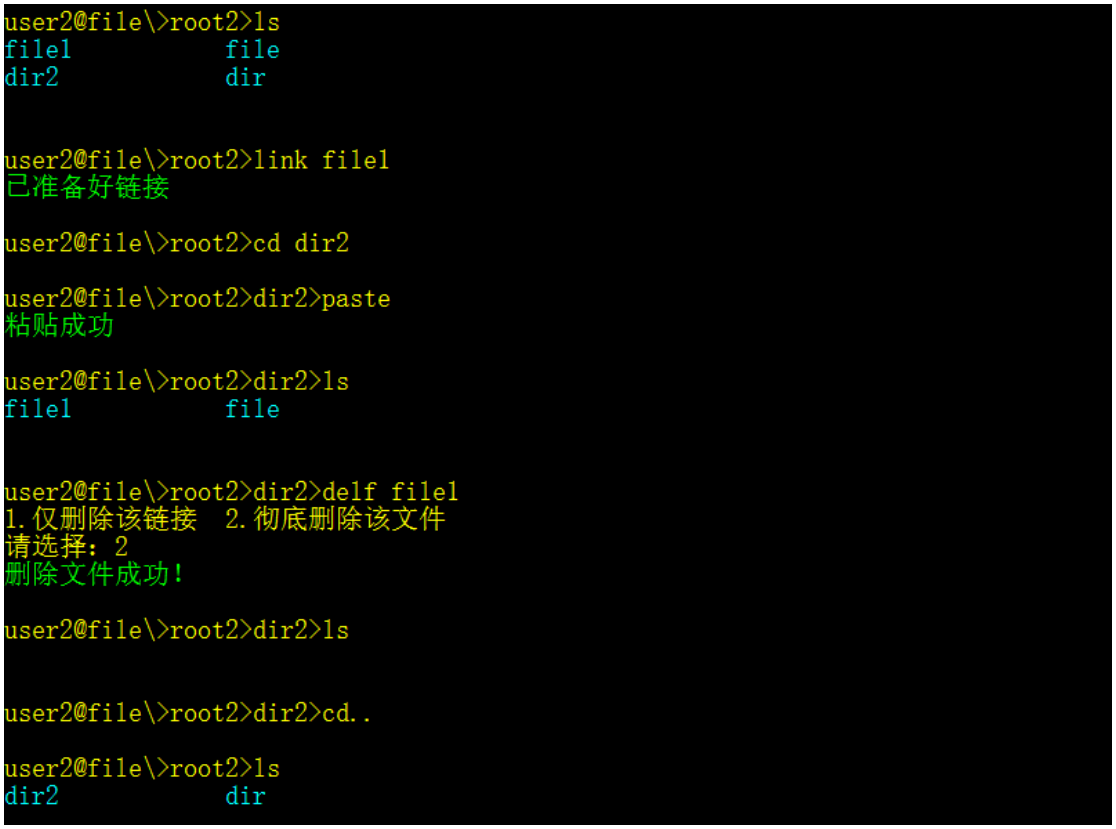


图 4.3.7 链接文件的删除（源文件受影响）

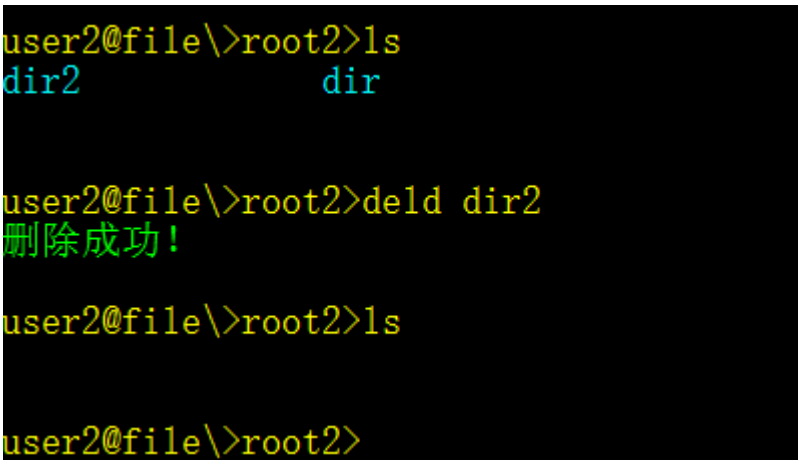


图 4.3.8 目录的删除

```
user3@file\>root3>ls


user3@file\>root3>create file1
创建成功!
是否立刻写入内容?
立刻写入请输入Y，稍后写入请输入N:  Y
文件输入-----1
手动输入-----2
2
000000000000
写文件成功!

user3@file\>root3>ls
file1           file


user3@file\>root3>rname file1 new_name
更名成功

user3@file\>root3>ls
new_name                file


user3@file\>root3>read new_name
000000000000

user3@file\>root3>
```

图 4.3.9 更名操作

```
user1@file\>find file1

文件名          路径
file1           \root1\file1
file1           \root2\file1
file1           \root2\dir\file1
file1           \root3\file1
file1           \root3\dir2\file1
file1           \root3\dir2\dir2\file1

user1@file\>find dir

文件名          路径
dir             \root2\dir
```

图 4.3.10 find 查找操作

```
user3@file\>root3>ls
file1           file
dir             dir


user3@file\>root3>format
警告：格式化将删除该用户的所有文件数据。
继续请输入Y，若想退出，请输入N：  Y
格式化成功!

user3@file\>cd root3

user3@file\>root3>ls


user3@file\>root3>
```

图 4.3.11 user3 的格式化操作

```
user3@file\>cd root1

user3@file\>root1>ls
file1           file
dir2            dir


user3@file\>root1>read file1
**********

user3@file\>root1>delf file1
删除失败，您没有在该目录下删除文件的权限

user3@file\>root1>create file4
创建失败，您没有在该目录下创建文件的权限

user3@file\>root1>
```

图 4.3.12 user3 对 user1 用户目录下文件的操作受限

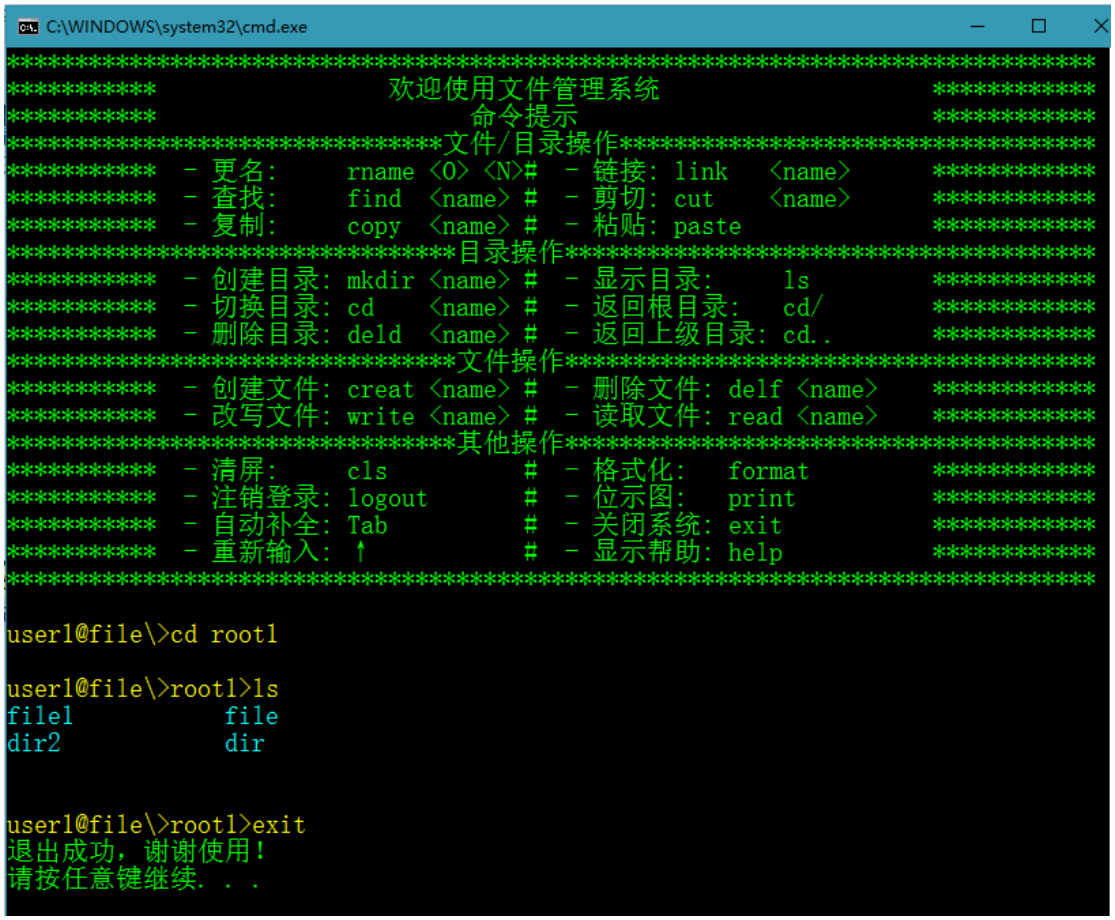


图 4.3.12 系统退出操作

图 4.3.12 退出保存

5 结论

本次课程设计我们实现了一个模拟 UNIX 文件系统。该系统能实现 UNIX 文件系统的大部分管理功能，如多用户的登入、登出及权限管理，系统初始化，系统的退出保存，文件及目录的创建、修改和删除，多级目录，目录切换等基本功能，通过成组链接法对空闲磁盘块进行分配与回收。此外，该文件系统还提供指令及文件目录名的自动补全，文件及目录的复制、粘贴、剪切、重命名、查找和创建快捷方式，格式化等扩展功能，能对用户输入的错误命令进行警告重输入， 还能高效管理磁盘的 i 节点、SFD、数据区等。且提供友好的用户界面，交互体验较好，使用过程中会有明确的交互性语句提示，提供不同颜色的提示强化效果， 指令帮助说明清晰，程序健壮性和容错性较好。

通过这次课程设计，我们对于日常使用的文件系统有了进一步了解，并且对于文件系统中文件的内部存储结构有了更清楚的认识。

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

经过两个星期的编译原理课程设计，在老师的指导下，本次课程设计顺利完成了。通过课程设计，我们受益匪浅。这次课程设计对我们而言是一次考验大家掌握知识、创新能力以及团结协作能力的挑战。

下面是各个组员的感悟收获体会：

“李灵灵：

通过这次操作系统课设，我学会了成组链接法的基本思想，并且构建了 UNIX 文件系统的数据结构，通过对用户权限和其他底层功能的设计，使我对文件系统有了更为深入的认识。我不仅在编程上的经验提升，更重要的是从底层出发，从自己设计的数据结构开始构建，仅仅这一面就很高的提升我对文件系统的认识， 并且理论上的知识也会更加清晰。

在两周的设计、实施和验收不仅感到之前的紧张和埋怨更有以后收获的喜悦，尤其是在最后验收时调试错误的焦虑和调试成功的喜悦形成鲜明的对比。这次课 程设计过程中，我参加了校外实习，所以投入不是很多，在这个过程中我学习了 很多，懂得了如何去协调自己的时间，了解了工作的烦恼和生活的困难，但是我 在课设中的投入相比其他人差了很多，上层的功能函数几乎没有触及，只是在扩 展功能上提出了一些自己的想法，但是我从中获益匪浅，更加坚定了自己对未来 的选择，相信我一定不会辜负现在努力的自己。”

“刘英杰：

本次操作系统课程设计，我主要负责完成多级索引结构设计、文件的创建删除，以及文件的读写功能。课程设计刚开始的时候，由于对功能要求不太理解， 采用先读例子代码再自行设计的方式来进行设计。通过对老师提供的源代码通读分析，对数据结构和整体功能有了大致的了解，在此基础上，重新设计并完成了文件操作的相关功能。

在课程设计中，遇到过很多 BUG，这些问题大部分是由于考虑不周全造成的，比如刚开始读取文件内容时，会读取出来一些与索引相关的信息。经过反复调试，发现是由于之前所占用的块回收后，其中的内容并没有完全清除造成的。

在本次课设中，也有一些有瑕疵的设计，比如在 i 节点数据结构中，将索引信息以磁盘块号方式存放在 i 节点中，将所有索引信息存放磁盘块中，每个文件首个索引块中只存放 13 个块号，虽然也可以实现索引功能，但磁盘空间浪费严重。

通过本次课程设计，对文件系统的实现有了深入了解，收获的不仅是知识，更重要的是独立思考、动手实践、团队合作能力。回顾这两周课程设计，至今仍感慨颇多，虽然这期间免不了挑灯夜战的疲惫，免不了找不到 BUG 的无奈，但也享受着成功的喜悦，团队合作的快乐。这期间可以说是苦多于甜，但是也学到很多东西，不仅巩固了以前学过的知识，而且学到了很多书本上没有讲过的知识。本次课程设计让我懂得了理论与实践相结合的重要性，只有理论知识是远远不够的，只有把所学的理论与实践相结合，才能充分理解理论，实践以理论为基础， 在理论中得出结论，在实践中发现完善修改结论的不足，并且实践过程中得到更好的理论。

特别感谢这段时间老师的悉心指导，让我们能更清楚的认识到设计的不足之处；也感谢队友的包容、理解和鼓励，让我能在这期间能坚持下来。”

“吉晓琪：

经过两周忙碌又充实的时间，终于完成了本次课程设计。由于准备夏令营，同时还有生产实习和考试，时间非常紧张。但是通过本次课程设计，作为小组组长的我收获还是非常巨大的，不仅仅是在知识方面、更是对团队合作有了更深刻的认识。

通过本次操作系统的课程设计，使得我对文件系统相关的知识有了更加深刻的认识和理解，对计算机系统是如何对存储资源进行分配回收，如何管理如此庞大的文件资源有了更加深刻的理解。另外就是在实践方面获得的一些收获，通过本次课程设计，锻炼了我对一个完整系统的把握能力，调试能力。

然后就是在团队合作方面的一些收获，这么短的时间内完成一个完整的工程，仅仅靠个人的力量是非常困难的。我们能较为圆满地完成此次课程设计的任务，很重要的一点就是最开始的时候分工合理明确，并且团队每一名成员都积极配合，都在某一方面发挥这不可或缺的作用。每个组员各司其职，但互相之间及时沟通 交流，通过网络或者当面交流各自的进度情况以及遇到的问题和困惑，然后小组

之间再讨论解决问题，并且不断完善设计方案。在这个过程中，我们一起经历了初次看到课题时的迷茫、无从下手；经历了遇到难点时的沮丧、焦急；但更多的是大家坚定地一起完成课设的信心和完成之后的欣喜和自豪感，我们也更加体会到团队合作的重要性，并且团队合作的能力都在潜移默化之中的到了提升，为我们以后的学习和工作都打下了良好的基础。通过该课程设计，我们全面系统的理解了文件系统的原理和实现方法。把死板的课本知识变得生动有趣，激发了学习的积极性。把学过的理论知识强化，能够把课堂上学的知识通过自己设计的程序表示出来，加深了对理论知识的理解。同时，此次课程设计也给了我们信心，让我们相信不管遇到设么样的困难，只要不放弃地多学习多请教、多查阅资料，困难总会解决。

课程设计需要很强的设计能力和动手能力，与平时的课堂教学有着很大的不同。在课堂上学习理论知识时觉得很抽象，但是当时感受并没有太深，然而直到开始做课程设计时才发现无从下手，无奈之下只得回到书本上，重新学习，更深层次去理解它们的内部的联系，最后再一步一步地在程序中实现出来。当然在编程实现的过程中也遇到了非常多的问题。课设中的一次次失败让我们体会到了
“纸上得来终觉浅，绝知此事要躬行”的道理，只有经过实践才能知道自己是不 是真的掌握了书本内容。

这次的课程设计对我们来说是一次很好地锻炼机会，不仅让我们体会到了将理论应用于实践的乐趣，又反过来通过实践提升了我们的理论水平。课程设计培养了我们的设计思维，增强了动手能力，也让我们更加严谨，明白了科学研究是半点都马虎不得的。不管是理论知识本身，还是实践水平、编程能力、团队合作能力都得到了提升。

至于建议方面就是，本次课程设计时间比较紧张，无论是在程序编写的时间还是最后验收的时间都比较紧，就导致有很多地方虽然想法很好，但由于时间限制只能放弃，即使编写的程序很出色，由于验收时间比较紧张，就导致无法将自己的成果很好地展示给验收老师。这是让人比较遗憾的地方。

最后，我们要对老师们说一声谢谢，是你们的辛苦工作才让我们有了此次提升自己的机会。老师，你们辛苦了！