同济大学计算机系

操作系统课程设计实验报告

学　　号 1652250

姓　　名 邓泓

专　　业 计算机科学与技术

授课老师 方钰

1 程序任务

1.1 输入

在控制台中输入类似命令行的命令。

1.2 输出形式

对文件进行处理。并在控制台提示相应信息。

1.3 程序功能

模拟二级文件系统，对其中的文件进行打开关闭读写等文件操作。

2 概要设计

2.1 任务分解

任务分为两部分：

第一部分为模拟命令行。

第二部分为文件操作（此部分占据程序大头）。

2.2 数据类型定义

输入皆为字符串形式。

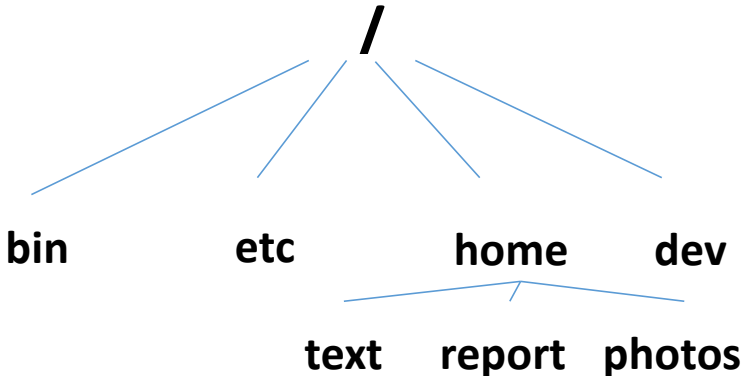
2.3 主程序流程

命令行方式，等待用户输入，给用户不同的输入，返回结果。

根据要求需要咨执行操作流程如下：

1 格式化文件卷。-ffformat

2mkdir 命令创建如下图子目录。-mkdir



3 把随意一个纯文本文件放入/home/texts，把课设报告放入/home/reports，把一张图片放入/home/phots 文件夹。（拷贝文件数据进入）

4 新建文件/test/Jerry，打开该文件，写入 800 个字节。

5 将文件读写指针定位到 500 字节，独读出 20 个字节。

2.4 模块间调用关系

程序就拥有一个主程序。不需要其他模块。

3 详细设计

3.1 磁盘文件结构定义

使用一个 16MB 名为 MyDisk.img 文件模拟一个磁盘，如下图所示：

3.2 文件操作

命令	解释	操作	使用	举例
fformat	格式化文件卷	初始化 SuperBlock 和 Inode，文件数据区清零	fformat	fformt
ls	列目录	显示磁盘信息，列出所有文件和文件夹，及相关信息	ls	ls
mkdir	创建目录	在当前目录下创建文件夹	mkdir name	mkdir /bin
fcreat	新建文件	在当前目录下创建文件	fcreat name	fcreat /test/Jerry

fopen	打开文件	将文件信息放入程序中	fopen name	fopen /test/Jerry
fclose	关闭文件	将程序中的文件信息清除	fclose	fclose
fread	读文件	根据读写指针位置读取 n 个字节信息到程序中	fread n	fread 100
fwrite	写文件	根据读写指针写 n 个字节到文件中，n 个字节数据为程序中定义好的字符串，如果读写指针为 0 写到文件末尾	fwrite n	fwrite 100
flseek	定位读写指针	更改读写指针为 n，n 表示在从 n-1 个字节开始，默认为 0	flseek n	flseek 100（表示从编号为 99 的字节开始）
fdelete	删除文件	删除文件	fdelete name	fdelete /test/Jerry
remove	删除目录	删除目录及子目录及子文件	remove /bin	remove /bin
help	帮助	在程序中提示相关指令及示例	help	help

3.3SuperBlock 设计（SuperBlock 占用 1 个块）

1 每个 block 大小（512 个字节）
2block 总数（16*1024*1024/512=32768 个）
3Inode 大小（128 个字节）
4Inode 总数（512*10/128=40，占用 10 个 block）
5block 未使用数
6Inode 未使用数
7 空白填充，使其占满 512 字节

```
class SuperBlock {
public:
    int block_size;      //block大小（单位Byte）
    int block_num;       //block数量
    int Inode_size;      //inode大小（单位Byte）
    int Inode_num;       //inode数量
    int block_unuse_num;//block未使用数量
    int Inode_unuse_num;//inode未使用数量
    char filler[488];    //superblock空白填充取，地址范围是24~447

    void SuperBlockSet(int, int, int, int, int, int);
};
```

3.4Inode 设计

1 类型
2 文件创建时间
3 最近一次读取时间
4 最近一次修改时间
5 文件大小，单位为 Byte
6 文件占用块数（其中大型文件不包括一级索引）
7 名称
8 文件索引表
9 填充至 128 字节

```
class Inode {//一个inode占用128个字节
public:
    int kind;           //类型：1为目录，2为小文件，3为大型文件，4为巨型文件
    int creat_time;     //创建时间（时间中的数字表示自1970年到现在经过的秒数）
    int amend_time;     //修改时间
    int read_time;      //读取时间
    int size;           //文件大小=占用块数*512（单位字节）
    int loc_num;        //如果为文件，文件占用地址表数量
    char name[16];      //文件/目录名
    int location[10];   //文件索引表
    int filler[12];     //填充
    Inode() {
        for (int i = 0; i < 16; i++) {
            name[i] = '\0';
        }
        for (int i = 0; i < 10; i++) {
            location[i] = 0;
        }
        for (int i = 0; i < 12; i++) {
            filler[i] = 0;
        }
    }
};
```

3.5Inode 节点分配与回收算法设计与实现

3.5.1 分配

1 创建文件或者目录
2 判断名字是否合理
3 相关信息录入
4 顺序写入 inode 区
5 修改 superblock

3.5.2 回收

1 寻找与输入名称相匹配的 inode 号
2 在地址表中修改占用 block 信息
3 将后面的 inode 向前移动一个单位
4 修改 superblock
5 如果是目录则不进行第二步

3.6 文件数据区的分配与回收算法设计与实现

3.6.1 分配

1 寻找 block 占用信息中第一个为 0 的标号
2 将这个文件的 inode 地址表[0,5]顺序改为刚找到的标号
3 写入这一块
4 如果没写完就重复 123
5 如果是大型文件，地址表[6,7]改为找到的标号，在标号物理块中存放地址表信息，可以存放 6+128*2 个
6 修改 superblock 信息和更新地址表信息

3.6.2 回收

1 将 inode 地址表和存放的地址表写入缓存中
2 顺序将地址表中的 block 编号的状态位改为 0
3 如果是大型文件则将 inode 地址表[6,7]中的 block 标号状态位改为 0
4 修改 superblock 信息和更新地址表信息

3.6.3 说明

1 文件数据区在回收时不会修改数据，只是将占用情况修改，即有的 block 可能有数据，但系统将其视为空白块，可以对其进行其他数据的录入，减少了程序工作量。

3.7 重点变量

1 全局变量区

```
int running = true;
SuperBlock superblock;
char buff[BLOCKSIZE];//缓存块
int open = 0;
/*打开文件的标志位
//如果为0则表示没有文件打开
//如果为正整数
//则代表打开的文件的Inode起始位置数值应为512到39*128+512之间*/
int seek = 0;//文件读写位置
bitset<4096> bits;//block占用情况
char writebuff[1024] = { 0 };
char readbuff[1024];
```

running 程序状态位，如果置 0 会跳出控制台循环，即终止程序
superblock 超级块信息缓存
buff 缓存块
seek 文件读写位置
bits 文件占用情况

这是一个 4096bit 位的数据，即 4096/=512 字节，是一种 c++节省空间的封装类，其作用就是按位操作。这个数据会存在 block 号 11 到 18 号，占用 8 位，每一位表示对应 block 号占用情况，比如说 superblock+inode+bits 这三个文件信息占用 19 块 block，则在初始化的时候，第 0 到第 18 位 bits 就是 1，其他为 0

writebuff 写文件缓存

初始化为 0-9 循环，方便测试数据

readbuff 读文件缓存

2 局部变量区

```
char com[COMNUM];
```

com 主函数指令缓存，从输入缓冲区读取数据到 com，对其进行判断和反馈用户

3 其他局部变量

这部分将放在 3.6 重点函数中讲解

3.8 重点函数

3.8.1 函数声明区

```
void Fformat();
void Mkdir(char*);
void Ls();
void Fcreat(char*);
void Fdelete(char*);
void Remove(char*);
void Fopen(char*);
void Fwrite(int);
void Fread(int);

void IOread(int);
void IOwrite(int);
void buff_zero();//清空缓存区
void SuperBlock_Change();//更新superblock
int Judge_Name(char*, int);//判断名字合法性
void DeleteInode(int);//删除一个inode
void DeleteChild(char*);//删除子节点，删除目录时连带删除子目录及目录下文件
int Serch_Block();//寻找空block
void Block_Set_Num(int, int);//block占用位改变
```

3.8.2 具体函数解释

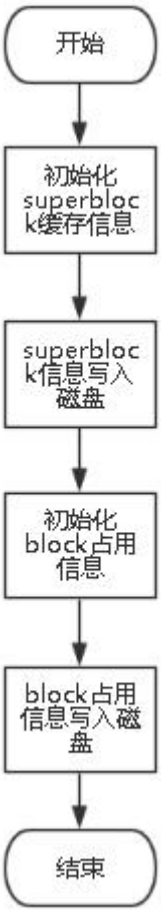
```
void Fformat();
输入：
输出：
执行：初始化磁盘
void Mkdir(char*);
输入：目录名
输出：
执行：根据目录名创建目录，分配 Inode
void Ls();
输入：
输出：
执行：显示磁盘信息，列出所有目录及文件和他们的信息
void Fcreat(char*);
输入：文件名
输出：
执行：根据文件名创建文件，分配 Inode
void Fdelete(char*);
输入：文件名
输出：
执行：根据文件名删除文件，回收 Inode
```

```
void Remove(char*);
输入：目录名
输出：
执行：根据目录名删除目录，回收 Inode
void Fopen(char*);
输入：文件名
输出：
执行：根据文件名打开文件，文件连接程序
void Fwrite(int);
输入：数量
输出：
执行：从缓存输入数量大小的字节到已经打开的文件中，分配 block
void Fread(int);
输入：数量
输出：
执行：从已经打开的文件中读取数量大小的字节到缓存
void IOread(int);
输入：位置，单位为 byte
输出：
执行：根据位置信息从磁盘中读取固定 512 字节的内容到 buff
void IOwrite(int);
输入：位置，单位为 byte
输出：
执行：根据位置信息从 buff 向磁盘写入固定 512 字节的内容
void buff_zero();//清空缓存区
输入：
输出：
执行：将 buff 置零
void SuperBlock_Change();//更新 superblock
输入：
输出：
执行：将程序 superblock 信息写入磁盘
int Judge_Name(char*, int);//判断名字合法性
输入：文件名/目录名，类型
输出：错误/正确信息
执行：判断名字合法性，大小，是否以/为开头等
void DeleteInode(int);//删除一个 inode
输入：位置，单位为个
输出：
执行：根据位置信息将该位置 Inode 信息删除，并将后面的 Inode 向前移动。
void DeleteChild(char*);//删除子节点，删除目录时连带删除子目录及目录下文件
输入：name
输出：
执行：删除已 name 开头的所有目录及文件如/1/a 和/1/b 会因为/1 的删除而删除
```

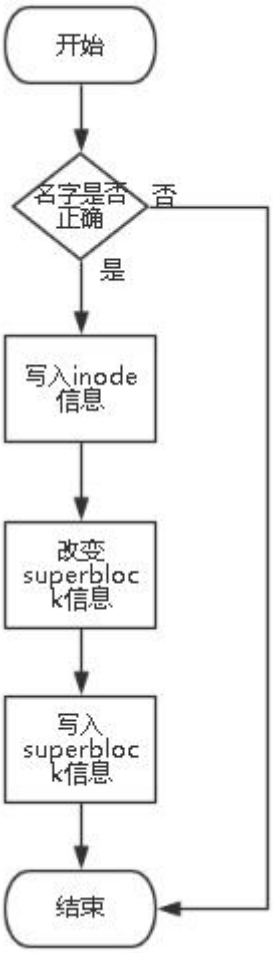
```
int Serch_Block();//寻找空 block
输入：
输出：位置，单位为个
执行：读取磁盘 block11-18 号，找到为 0 的位置返回
void Block_Set_Num(int, int);//block 占用位改变
输入：位置，单位为个，0/1
输出：
执行：将这个位置的标志位变成 0/1
```

3.9 重点功能部分（流程图）

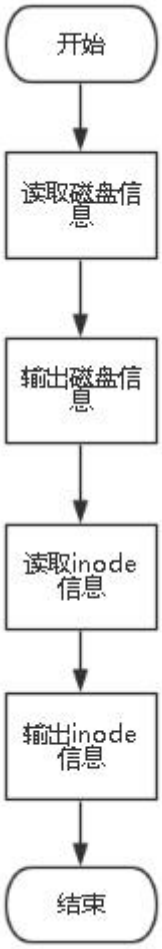
3.9.1fformat



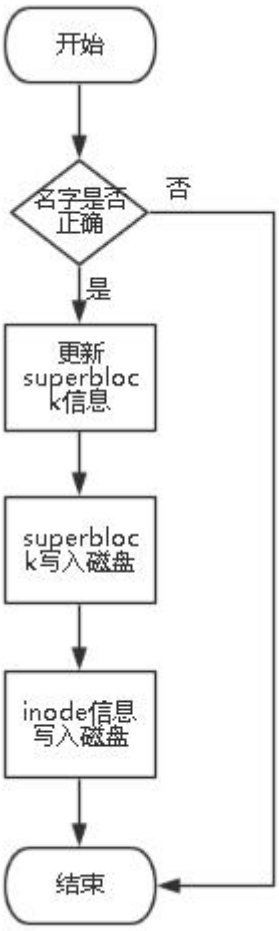
3.9.2mkdir



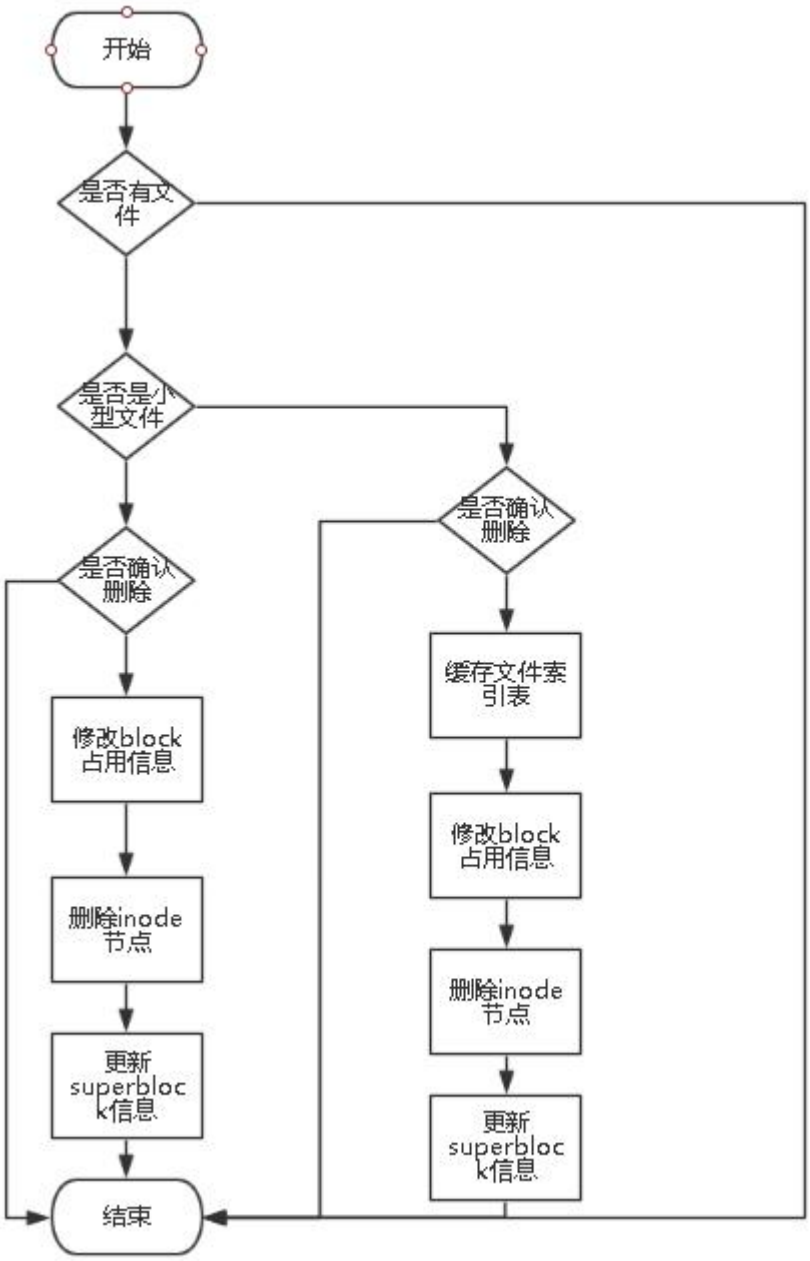
3.9.3ls



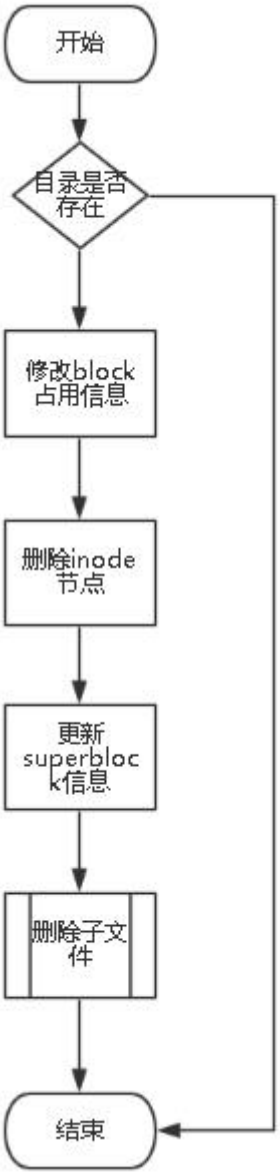
3.9.4fcreat


开始
名字是否正确
否
是
更新superblock信息
superblock写入磁盘
inode信息写入磁盘
结束

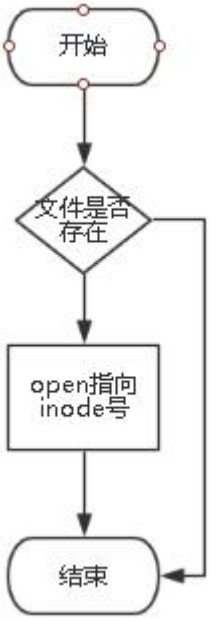
3.9.5fdelete


开始
是否有文件
是否是小型文件
是否确认删除
修改block占用信息
删除inode节点
更新superblock信息
是否确认删除
缓存文件索引表
修改block占用信息
删除inode节点
更新superblock信息
结束

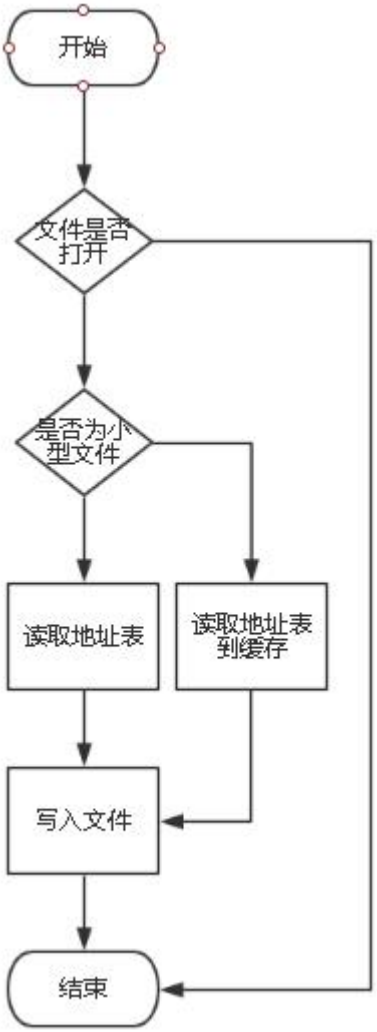
3.9.6remove



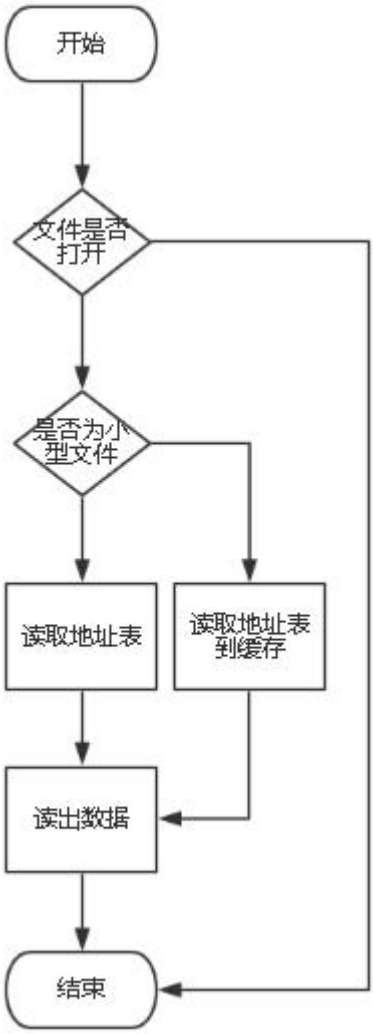
3.9.7fopen



3.9.8fwrite



3.9.9fread



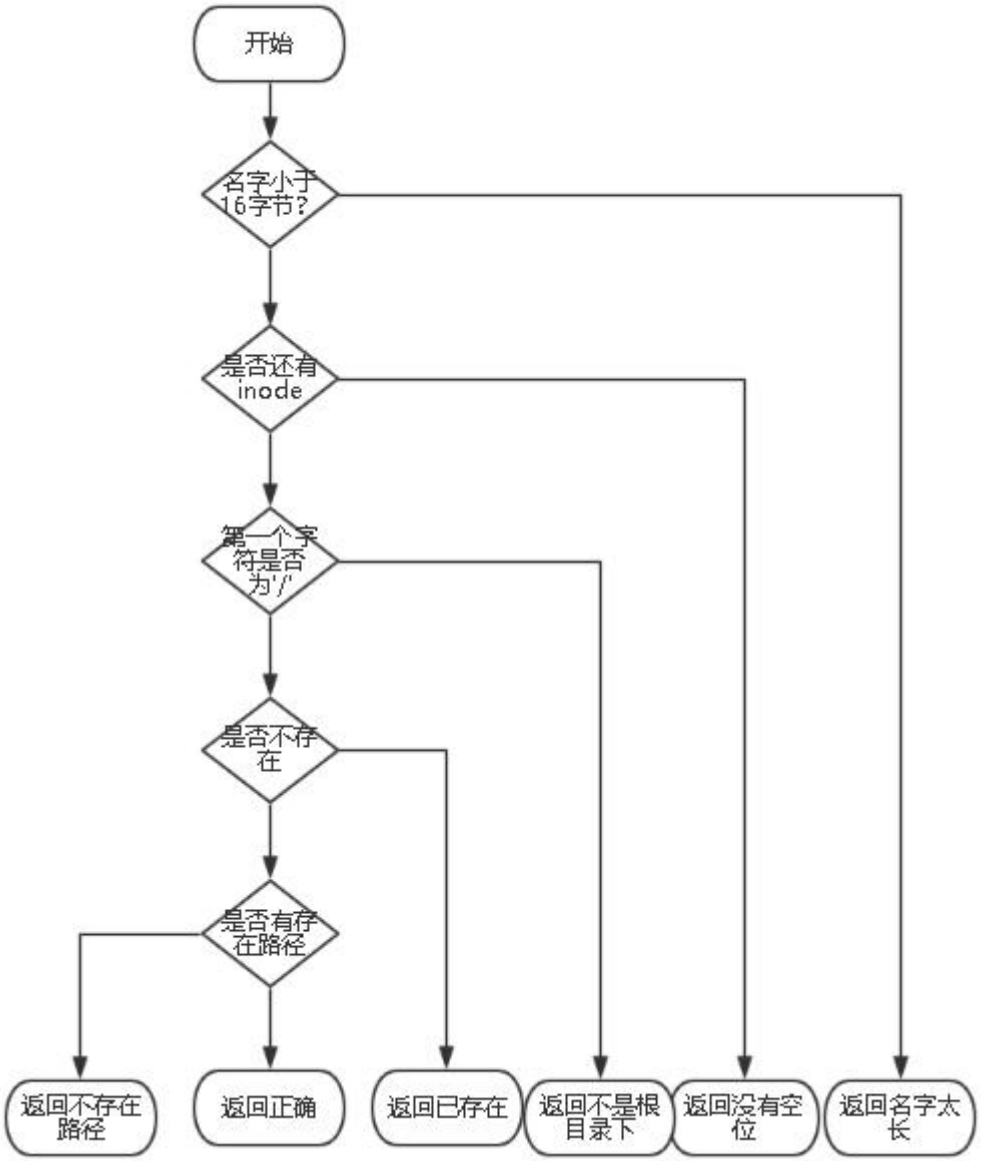
3.9.10IOread

3.9.11IOread

3.9.12buff_zero

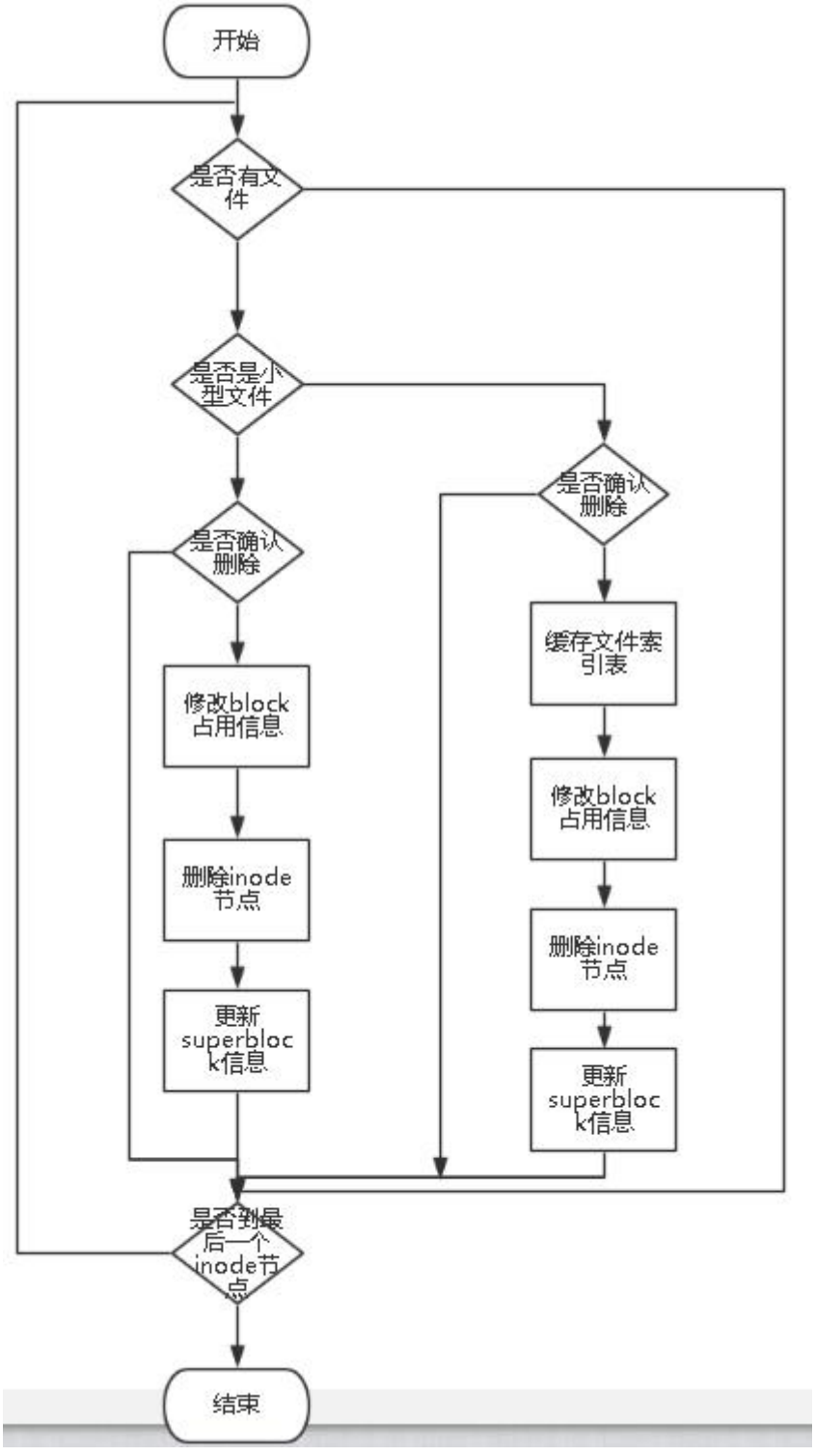
3.9.13super_block_change

3.9.14judge_name



3.9.15deleteinode

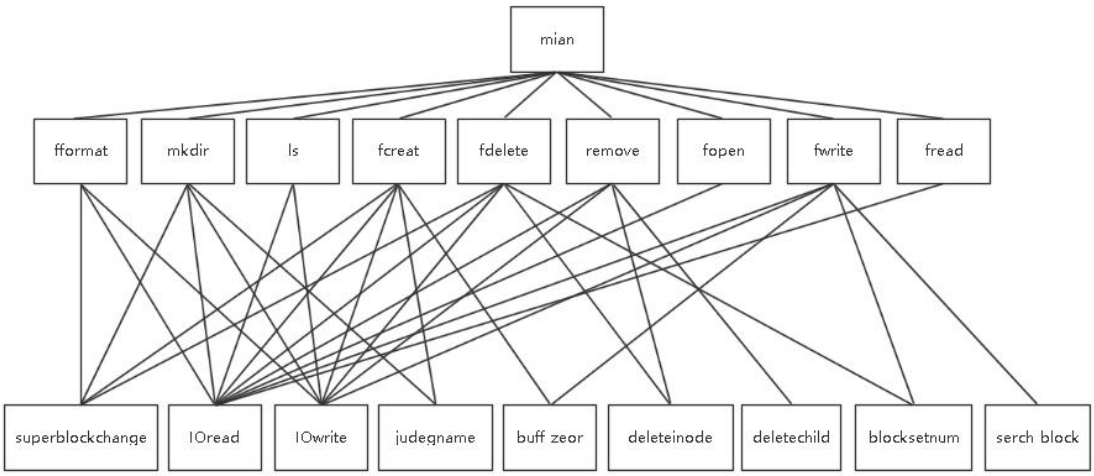
3.9.16deletechild



3.9.17serchblock

3.9.18block_set_num

3.10 函数调用关系（图）

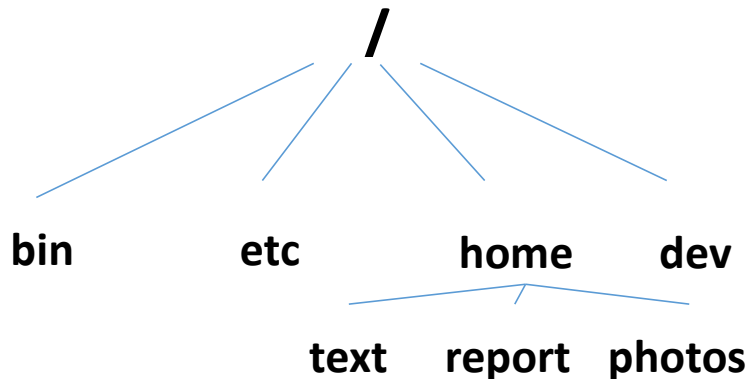


上层调用下层

4 调试分析

4.1 测试数据

格式化文件卷
mkdir 创建目录



新建文件/test/Jerry,打开文件，写入 800 个字节
文件读写指针定位到 500，读出 20 个字节
将随意的一个文本文件，报告，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

4.2 测试输出结果

1 建立如图文件结构 fformat 初始化磁盘空间

```
Impor 'help' and you can see the commonds.
[/]#mkdir /bin
Mkdir success.
[/]#mkdir /etc
Mkdir success.
[/]#mkdir /home
Mkdir success.
[/]#mkdir /dev
Mkdir success.
[/]#mkdir /home/text
Mkdir success.
[/]#mkdir /home/report
Mkdir success.
[/]#mkdir /home/photos
Mkdir success.
[/]#ls
磁盘扇区大小:       512   kb
磁盘扇区总数:       32768 个
磁盘INODE大小:      128   kb
磁盘INODE总数:      40    个
磁盘未使用扇区数:   32749 个
磁盘未使用INODE数:  32    个
/1--------------------------文件----1000B----Mon Jun 10 01:30:42 2019
/bin-------------------------目录----0B----Mon Jun 10 17:45:39 2019
/etc-------------------------目录----0B----Mon Jun 10 17:45:46 2019
/home------------------------目录----0B----Mon Jun 10 17:45:51 2019
/dev-------------------------目录----0B----Mon Jun 10 17:45:55 2019
/home/text-------------------目录----0B----Mon Jun 10 17:46:06 2019
/home/report-----------------目录----0B----Mon Jun 10 17:46:15 2019
/home/photos-----------------目录----0B----Mon Jun 10 17:46:26 2019
[/]#fformat
Format diskette completely.
[/]#ls
磁盘扇区大小:       512   kb
磁盘扇区总数:       32768 个
磁盘INODE大小:      128   kb
磁盘INODE总数:      40    个
磁盘未使用扇区数:   32749 个
磁盘未使用INODE数:  40    个
[/]#
```

2 建立文件结构

```
[/]#mkdir /bin
Mkdir success.
[/]#mkdir /etc
Mkdir success.
[/]#mkdir /home
Mkdir success.
[/]#mkdir /dev
Mkdir success.
[/]#mkdir /home/text
Mkdir success.
[/]#mkdir /home/report
Mkdir success.
[/]#mkdir /home/photos
Mkdir success.
[/]#ls
磁盘扇区大小：       512   kb
磁盘扇区总数：       32768 个
磁盘INODE大小：      128   kb
磁盘INODE总数：      40    个
磁盘未使用扇区数：   32749 个
磁盘未使用INODE数：  33    个
/bin--------------------------目录----0B----Mon Jun 10 17:47:40 2019
/etc--------------------------目录----0B----Mon Jun 10 17:47:43 2019
/home-------------------------目录----0B----Mon Jun 10 17:47:45 2019
/dev--------------------------目录----0B----Mon Jun 10 17:47:47 2019
/home/text--------------------目录----0B----Mon Jun 10 17:47:48 2019
/home/report------------------目录----0B----Mon Jun 10 17:47:50 2019
/home/photos------------------目录----0B----Mon Jun 10 17:47:51 2019
[/]#
```

3 新建/test/Jerry，发现反馈错误信息，还没有建立/test 目录，所以创建文件失败

```
[/]#fcreat /test/Jerry
Please check the location.
[/]#
```

4 新建/test，新建/test/Jerry

```
[/]#mkdir /test
Mkdir success.
[/]#fcreat /test/Jerry
Fcreat success.
[/]#ls
磁盘扇区大小：       512   kb
磁盘扇区总数：       32768 个
磁盘INODE大小：      128   kb
磁盘INODE总数：      40    个
磁盘未使用扇区数：   32749 个
磁盘未使用INODE数：  31    个
/bin--------------------------目录----0B----Mon Jun 10 17:47:40 2019
/etc--------------------------目录----0B----Mon Jun 10 17:47:43 2019
/home-------------------------目录----0B----Mon Jun 10 17:47:45 2019
/dev--------------------------目录----0B----Mon Jun 10 17:47:47 2019
/home/text--------------------目录----0B----Mon Jun 10 17:47:48 2019
/home/report------------------目录----0B----Mon Jun 10 17:47:50 2019
/home/photos------------------目录----0B----Mon Jun 10 17:47:51 2019
/test-------------------------目录----0B----Mon Jun 10 17:49:56 2019
/test/Jerry-------------------文件----0B----Mon Jun 10 17:49:58 2019
[/]#
```

5 测试 open 和 lseek，write，read，如果不先 open 会反馈失败，由于是 0-9 循环，所以输出正确

```
[/]#fwrite 800
Please fopen file first.
[/]#fopen /test/Jerry
Fopen /test/Jerry seccuss.
[/]#fwrite 800
[/]#flseek 500
[/]#fread 20
read (20) byte:
01234567890123456789
[/]#
```

6 查看文件，发现 Jerry 大小为 800B

```
/bin--------------------------目录----0B----Mon Jun 10 17:47:40 2019
/etc--------------------------目录----0B----Mon Jun 10 17:47:43 2019
/home-------------------------目录----0B----Mon Jun 10 17:47:45 2019
/dev--------------------------目录----0B----Mon Jun 10 17:47:47 2019
/home/text--------------------目录----0B----Mon Jun 10 17:47:48 2019
/home/report------------------目录----0B----Mon Jun 10 17:47:50 2019
/home/photos------------------目录----0B----Mon Jun 10 17:47:51 2019
/test-------------------------目录----0B----Mon Jun 10 17:49:56 2019
/test/Jerry-------------------文件----800B----Mon Jun 10 17:49:58 2019
[/]#
```

7 数据正确性测试，读取非 10 倍数做测试，结果正确

```
[/]#fread 22
read (22) byte:
0123456789012345678901
[/]#
```

8 数据正确性测试，读写指针非 10 倍，读取非 10 倍，结果正确

```
[/]#flseek 99
[/]#fread 11
read (11) byte:
90123456789
[/]#
```

9 将报告，文本和图片写入磁盘。（注：由于报告还未完成，所以和磁盘中报告和报告不完全一致，由于名字长度限制，报告命名为 rp，图片为 pho，文本为 txt，个人能力原因，路径和名字统一为名字）

4.3 复杂度分析

```
对每个函数进行复杂度分析
int main();
时间：∞，只要不退出程序，就会永远运行下去
void Fformat();
时间：O(1)
void Mkdir(char*);
时间：O(1)
void Ls();
```

```
时间：O(n)
void Fcreat(char*);
时间：O(1)
void Fdelete(char*);
时间：O(1)
void Remove(char*);
时间：O(1)
void Fopen(char*);
时间：O(1)
void Fwrite(int);
时间：O(n)
void Fread(int);
时间：O(n)
void IOread(int);
时间：O(1)
void IOwrite(int);
时间：O(1)
void buff_zero();//清空缓存区
时间：O(1)
void SuperBlock_Change();//更新 superblock
时间：O(1)
int Judge_Name(char*, int);//判断名字合法性
时间：O(1)
void DeleteInode(int);//删除一个 inode
时间：O(n)
void DeleteChild(char*);//删除子节点，删除目录时连带删除子目录及目录下文件
时间：O(n)
int Serch_Block();//寻找空 block
时间：O(n)
void Block_Set_Num(int, int);//block 占用位改变
时间：O(n)
```

4.4 每个模块设计和调用时存在的问题的思考

只有一个问题是将其他文件写入，由于没有定义其他文件如何写入，所以我的解决方案是写一个新的命令，将已有文件写入磁盘中。

5 用户使用说明

输入相应指令

输入 help 可以查看其它指令格式

```
Impor 'help' and you can see the commonds.
[/]#help
All commond and example for commond:
fformat (or FFORMAT):     fformat       初始化磁盘
mkdir (or MKDIR):         mkdir /1      创建目录
ls (or LS):               ls            读取磁盘信息
fcreat (or FCREAT):       fcreat /1     创建/1文件
fdelete (or FDELETE):     fdelete /1    删除/1文件
remove (or REMOVE):       remove /1     删除/1目录
fopen (or FOPEN):         fopen /1      打开/1文件
fclose (or FCLOSE):       close         关闭上一个open的文件
fwrite (or FWRITE):       fwrite 1000   写入1000字节
flseek (or FLSEEK):       fwrite 1000   读取1000字节
fread (or FREAD):         fread 1000    读写指针改为1000
help (or HELP):           help          帮助
[/]#
```

6 运行结果分析

运行结果正确，对用户操作有及时的反馈。

7 实验总结

7.1 实验收获

通过这次对操作系统文件系统的编写，我对文件系统有了更深入的了解。比如说在课程上无法理解的一些设定，在实现过程中都逐步感受到这些设定带来的作用。这将使我在操作系统方面具有更加完备的知识体系。同时编程方面也有提高，解决问题能力也有提升。

7.2 遇到问题及解决问题过程的思考

首先遇到的问题是输入缓冲区的问题，通过对不同函数的使用，解决输入缓冲区问题，实现了类似于控制台的模拟界面。

第二个遇到的问题是 inode 的回收问题，顺序存储带来的是查询上的方便，对于删除来说并不方便，这要将所有 inode 读取出来，再存进去，幸运的是这次 inode 数量不多，如果磁盘大小扩大，inode 节点数量不支持顺序向前移动的时候，我想应该和数据存储一样，设定 inode 存储地址表，来对 inode 进行增删。

遇到的第三个问题也是困扰时间最长的问题，数据如何存储。第一个想法和 inode 一样，顺序存储，每当有 block 写入，就将之后的 block 顺序后移，这带来的问题是数量并不和 inode 一样只有 40 个之内，而是 32768 这样一个无法估计系统代价的数字，所以顺序存储想法失败。这样就消耗了非常多的思考时间，在读取操作系统书籍后，发现了地址表这个东西，理论上是随机存储，实际上是顺序的对 block 进行读取和写入。这个时候就遇到了第四个问题。

第四个问题是，如果是碎片化存储模式，只有地址表映射到物理地址，那么要怎么样才能找到一个“空白”存储块，将数据对其写入呢？这个时候面临了一个磁盘结构的设计问题。

怎样设计，才能使空间达到理想的使用效率呢？由于 c++中对数据处理的单位是字节，8 个比特。对 32768 这个数字，就要使用 32768*8/512=512 块物理块对其进行存储，512 个物理块是 256KB，这占用了 16MB 不少的存储空间。在一番思考和网上寻找解决方案后，找到了一个叫 bitset 的包，他是按照比特位进行数据处理的，存储的时候是按照 int 整形存储的，这就是我的解决方案。对于 32768 个 block，我只需要用到 32768/8/512=8 个物理块存储地址表，这对于 256KB 是多大的一个改进呢，节省近 8 倍的存储空间。

7.3 对课程的认识

磁盘格式问题，一个磁盘不能所有存储空间都存放数据，而是占用一定存储空间对磁盘分区，磁盘信息进行存储。而文件系统就是读取这些信息，对磁盘进行读写数据操作。

8 参考文献

操作系统原理（讲义）同济大学 2017-9 版