实现

目录实现

本文件系统采用线性列表来实现目录。每个目录下有一个文件列表和文件夹列表。文件系统有一个固定的根目录“/”来容纳所有新建的文件和文件夹。

分配方法

本文件系统采用链接分配（**FAT**）实现分配方法。对应位的值表示文件下一个块的位置，-1表示块终点。

空闲空间管理

本文件系统采用链表的方法管理空闲空间。每个块实例有一个变量指向下一个空闲块的位置，并且有全局变量存储

第一个空闲块的位置，以此为文件分配新的空间。

# 数据块结构

//块结构

class diskBlock { public:

char data[BLOCK\_SIZE]; //数据

short nextFreeBlock; //下一个空闲块位置diskBlock(short nFB);

};

文件结构

// 文 件 结 构 class myFile { public:

myFile(short parent, const char\* name = DEFAULT\_FILE\_NAME);

~myFile();

void read(char\*& str); //读取文件内容void write(string content); //写入文件内容

friend class FSManager; friend class OS3;

friend void restoreDirectory();

private:

char name[NAME\_LENGTH]; //文件名

short parentFolder; //所在目录的位置short sizeByBlock; //文件的大小

short startBlock; //文件的起始块位置

short extend(short finalBlock); //扩展文件空间

void relink(short first, short len); //重新连接空闲块链表

};

文件夹结构

// 文 件 夹 结 构 class myFolder { public:

myFolder(short parent, const char\* \_name = DEFAULT\_FOLDER\_NAME);

~myFolder();

void setInfo(short parent, const char\* name, short index); //设置文件夹相关参数void addFile(myFile\* file); //添加文件

void addFolder(myFolder\* folder); //添加文件夹short parentFolder; //所在目录的位置

friend class FSManager; friend class OS3;

friend void storeMyFolders();

friend void clearFolder(myFolder\* folder); private:

char name[NAME\_LENGTH]; //文件夹名

short index; //本文件夹的位置vector<myFile\*> fileList; //本目录下文件列表vector<myFolder\*> folderList; //本目录下文件夹列表

};

文件系统管理者

//文件系统管理者

class FSManager { public:

FSManager();

~FSManager();

myFile\* newFile(myFolder\* parent, string name);

//新建文件

myFolder\* newFolder(myFolder\* parent, string name); //新建文件夹

void format(); //格式化

void removeFile(myFolder\* parent, myFile\* file); //删除文件void removeFolder(myFolder\* parent, myFolder\* folder);//删除文件夹string getDirPath(); //返回当前路径

vector<myFolder\*> currentPath;

//按顺序存储当前路径下的各目录

};

# 文件系统存储

本文件系统需要存储的数据有： 文件分配表（FAT）

块列表、首空闲快的位置（ﬁrstEmptyBlock） 所有文件的列表（allFiles）

所有文件夹的列表（allFolders）

所有空缺文件夹位置的列表（removedFolderLocations）

每个数据都有对应的文件，在程序退出时文件系统会将当前状态下这些数据存到同目录下。

# 文件系统恢复

文件系统在初始化时尝试读取同目录下的数据，这里有两种情况：

读取成功。文件系统会读取所有数据，并以此初始化相应的变量。然后恢复目录结构 读取失败。文件系统会将初始化成原始状态。

界面

初始界面**/**显示目录



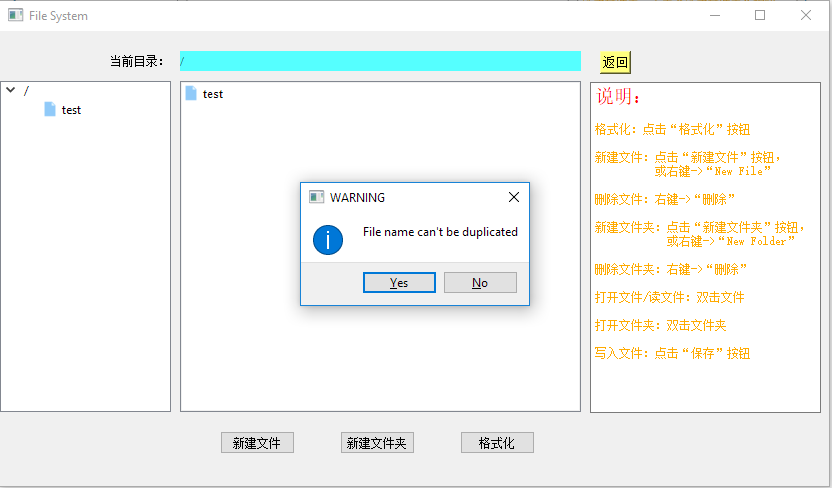
# 新建文件

## 输入文件名

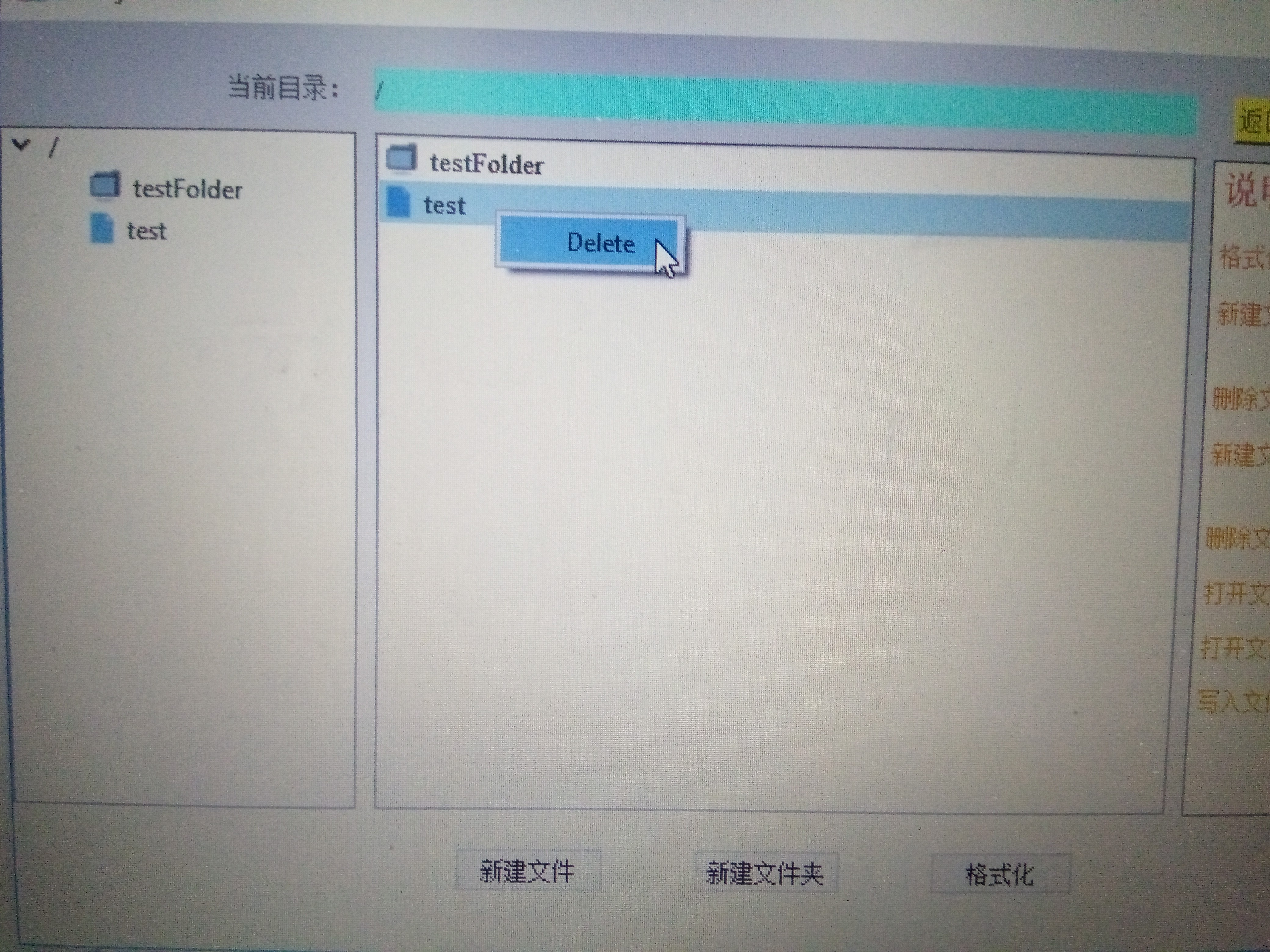


右键->New File

## 文件名重复



删除文件



右键->“delete”

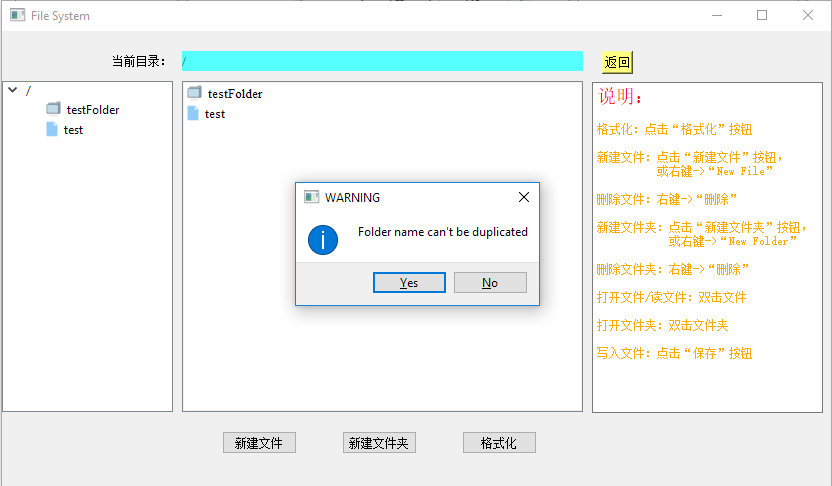
# 新建文件夹

## 输入文件夹名

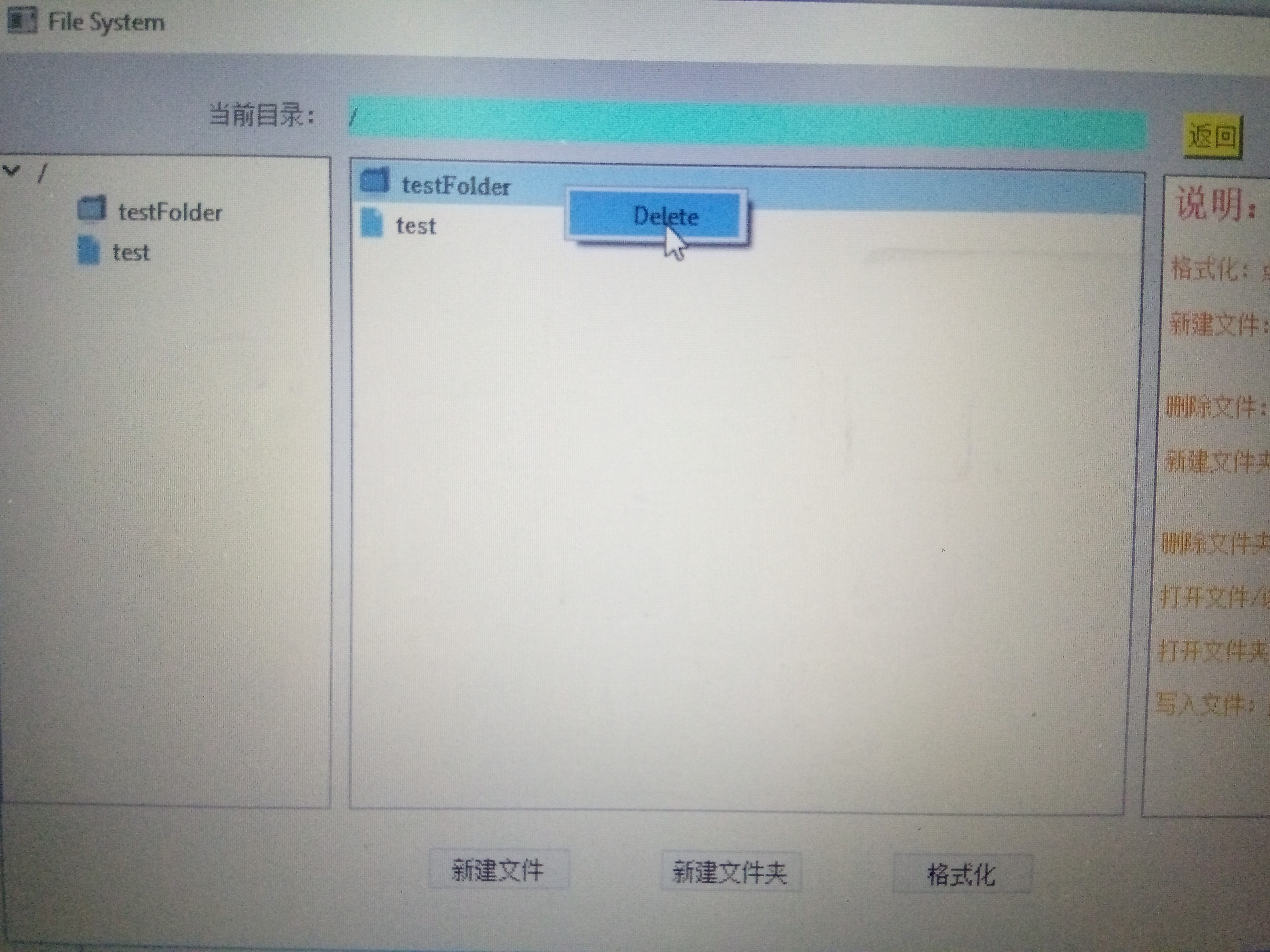


右键->New Folder

## 文件夹重复

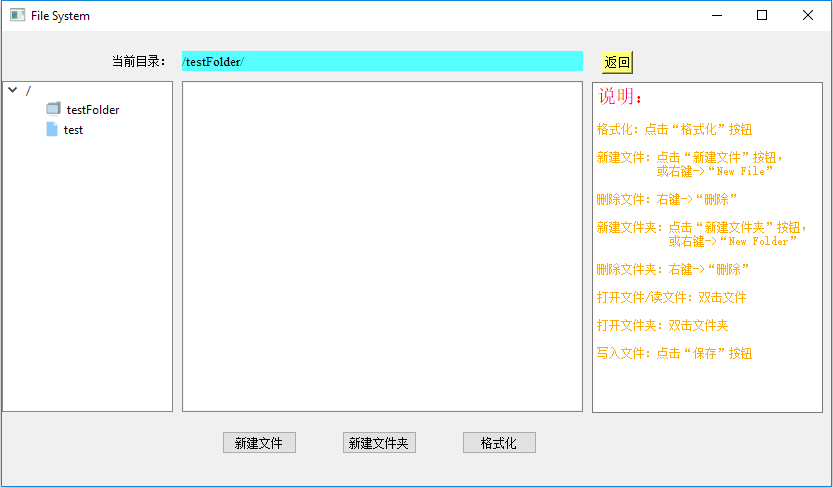


删除文件夹

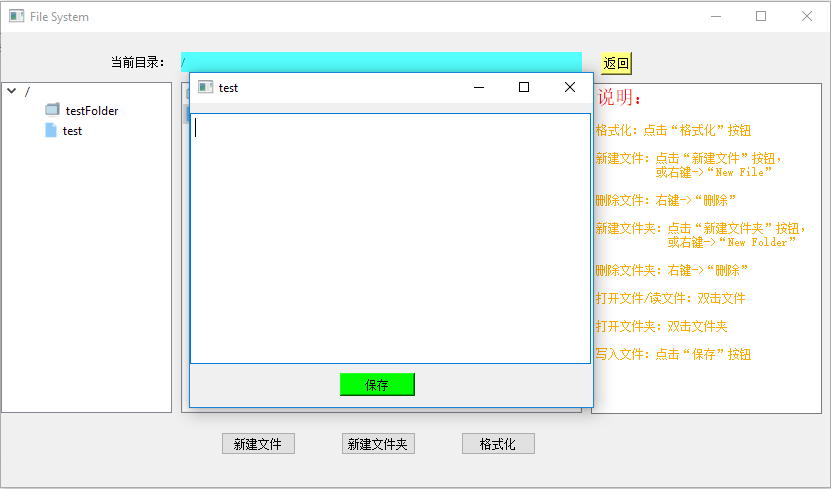


右键->"delete"

# 更改当前目录

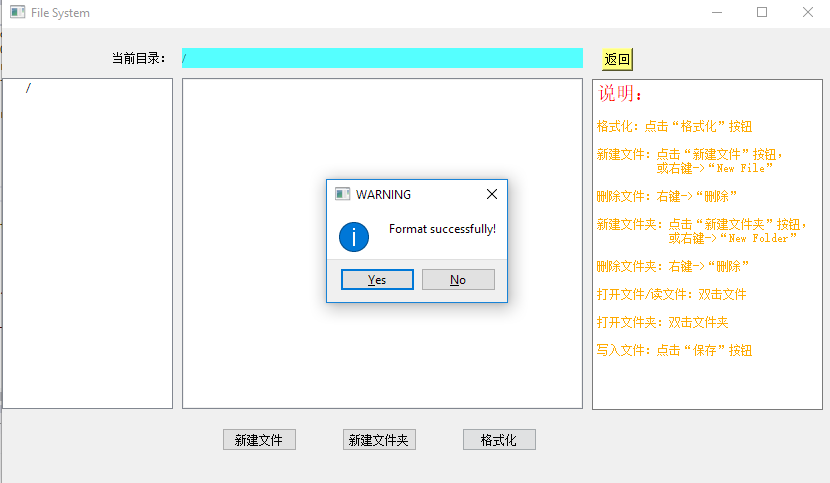


双击"testFolder"即可更改当前目录到“testFolder”

打开文件、读文件

# 写文件

格式化



算法

## 空闲空间管理

分配空闲快

short myFile::extend(short finalBLock) { if (finalBLock == -1) {

startBlock = firstEmptyBlock; //文件为空

}

else {

FAT[finalBLock] = firstEmptyBlock;

}

finalBLock = firstEmptyBlock; //分配一个块FAT[finalBLock] = -1;

firstEmptyBlock = blockList[firstEmptyBlock]->nextFreeBlock; //更新第一个空闲块位置sizeByBlock++;

return finalBLock;

}

### 回收空闲块

myFile::~myFile(){

if (startBlock == -1) return;

short current = startBlock, next = FAT[current];

while (current != -1 && next != -1) { //循环将待删除文件的块加入空闲块链表blockList[current]->nextFreeBlock = next;

current = next; next = FAT[next];

}

blockList[current]->nextFreeBlock = firstEmptyBlock; firstEmptyBlock = startBlock; //更新第一个块位置

}

合并文件块的内容

}

content.append(blockList[current]->data); current = FAT[current];

}

str = new char[content.length() + 1]; myStrCpy(str, content.c\_str(), content.length());

//循环合并所有块的内容

void myFile::read(char\*& str){ short current = startBlock; string content;

while (current >= 0) {

### 将内容分成块写入文件

void myFile::write(string content) { if (content.empty())

return;

int block\_needed = (content.length() - 1) / (BLOCK\_SIZE - 1) + 1;//计算当前内容需要的块个数int extra = content.length() % (BLOCK\_SIZE - 1);//计算未满一个块的内容长度

if (startBlock == -1) extend(startBlock);

if (extra == 0)

extra = BLOCK\_SIZE - 1; short current = startBlock;

for (short i = 0; i < block\_needed - 1; i++) { //循环写入

myStrCpy(blockList[current]->data, content.substr(i \* (BLOCK\_SIZE - 1), (BLOCK\_SIZE

- 1)).c\_str(), BLOCK\_SIZE - 1);

if (FAT[current] == -1) current = extend(current);

else

current = FAT[current];

}

myStrCpy(blockList[current]->data, content.substr((BLOCK\_SIZE - 1) \* (block\_needed - 1), extra).c\_str(), extra);

relink(current, sizeByBlock - block\_needed - 1); //重新链接

sizeByBlock = block\_needed; //更新文件大小

}

恢复目录结构

}

allFolders[i->parentFolder]->addFolder(i);

}

else if(i->parentFolder == -1) //根目录Root = i;

}

for (auto i : allFiles) { //恢复所有文件if (i->parentFolder >= 0) {

allFolders[i->parentFolder]->addFile(i);

}

}

//恢复所有文件夹

void restoreDirectory(){

for (auto i : allFolders){

if (i->parentFolder >= 0) {

性能

# 新建文件**/**新建文件夹

时间复杂度均为O(1)

# 读文件

设文件占有n个块，每个块大小为S, 则时间复杂度为O(nS)

# 写文件

设将要写入文件的内容占n个块，每个快大小为S, 分配一个空闲块复杂度为O(1), 则时间复杂度为O(nS + n)

# 删除文件

设文件占有n个块，由于需要重新链接空闲块链表，时间复杂度为O(n)

# 删除文件夹

设文件夹有m个文件，平均每个文件有n个块，同样时间复杂度为O(mn)

# 打开文件夹

设文件夹下有m个文件夹，n个文件，则更新UI界面的时间复杂度为O(m + n)