**同济大学软件工程**

**模拟文件管理系统项目报告**

****

**学 号 2151636**

**姓 名 李骁**

**专 业 软件工程**

**班 级 42036903**

**授课老师 王冬青**

**完成日期 2023/6/17**

**目录**

[模拟文件管理系统 3](#_Toc138110785)

[一 设计要求 3](#_Toc138110786)

[1. 基本要求 3](#_Toc138110787)

[2. 具体要求 3](#_Toc138110788)

[3. 提供操作 3](#_Toc138110789)

[二 项目组成及操作说明： 3](#_Toc138110790)

[1. 代码文件 3](#_Toc138110791)

[2. ui及图标库文件 4](#_Toc138110792)

[3. 保存的文件 4](#_Toc138110793)

[4. 操作说明 4](#_Toc138110794)

[三 界面及功能展示 4](#_Toc138110795)

[1. 进入提示 4](#_Toc138110796)

[2. 主界面 4](#_Toc138110797)

[3. 目录操作界面 5](#_Toc138110798)

[4. 文件操作界面 6](#_Toc138110799)

[5. 格式化 7](#_Toc138110800)

[四 设计思路 8](#_Toc138110801)

[1. 主要逻辑部分设计思路 8](#_Toc138110802)

[**1.1.** **宏定义、常量、枚举** 9](#_Toc138110803)

[**1.2.** **主类的子结构体定义** 10](#_Toc138110804)

[**FCB类（文件控制块）：** 10](#_Toc138110805)

[**Directory类（目录）：(刚好128B，即一个物理块的长度)** 10](#_Toc138110806)

[**Disk类（虚拟磁盘）：** 11](#_Toc138110807)

[**1.3.** **主类(fileSystem)的定义：** 11](#_Toc138110808)

[**1)** **读写文件函数：** 13](#_Toc138110809)

[**2)** **Create函数：（新建文件/目录）** 15](#_Toc138110810)

[**3)** **Delete函数：（删除文件/目录）** 16](#_Toc138110811)

[**4)** **SaveFS、LoadFS系统保存与读入函数及Format格式化：** 18](#_Toc138110812)

[2. 界面部分设计思路 19](#_Toc138110813)

[**1)** **fileSystem窗口构造函数：** 19](#_Toc138110814)

[**2)** **DrawCell、DrawTable函数：** 20](#_Toc138110815)

[**3)** **DrawCell函数：** 23](#_Toc138110816)

[**4)** **DrawBitmap函数：** 26](#_Toc138110817)

# 模拟文件管理系统

1. 设计要求
   1. 基本要求

**在内存中开辟一个空间作为文件存储器，在其上实现一个简单的文件系统;**

**退出这个文件系统时，需要该文件系统的内容保存到磁盘上，以便下次可以将其回复到内存中来。**

* 1. 具体要求

**文件存储空间管理可采取链接结构（如FAT文件系统中的显式链接等）或者其他学过的方法；**

**空闲空间管理可采用位图或者其他方法；**

**文件目录采用多级目录结构，目录项目中应包含：文件名、物理地址、长度等信息。**

* 1. 提供操作

**格式化**

**创建子目录**

**删除子目录**

**显示目录**

**更改当前目录**

**……**

**创建文件**

**打开文件**

**关闭文件**

**写文件**

**读文件**

**删除文件**

**……**

1. 项目组成及操作说明：
   1. 代码文件

**SOURCES += \**

**filesystem.cpp \**

**main.cpp**

**HEADERS += \**

**filesystem.h**

* 1. ui及图标库文件

**FORMS += \**

**filesystem.ui**

**RESOURCES += \**

**icons.qrc**

* 1. 保存的文件

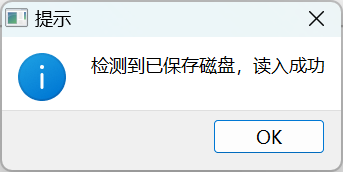
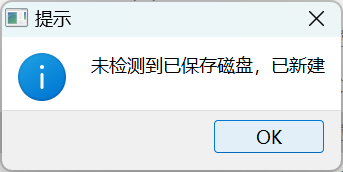
**默认名为FileSystem.dat，默认路径为项目根目录（Release版本即FileSystem\_Release目录下）**

* 1. 操作说明

**点击FileSystem\_Release下的FileSystem.exe即可运行。正常退出时文件自动保存。**

1. 界面及功能展示
   1. 进入提示

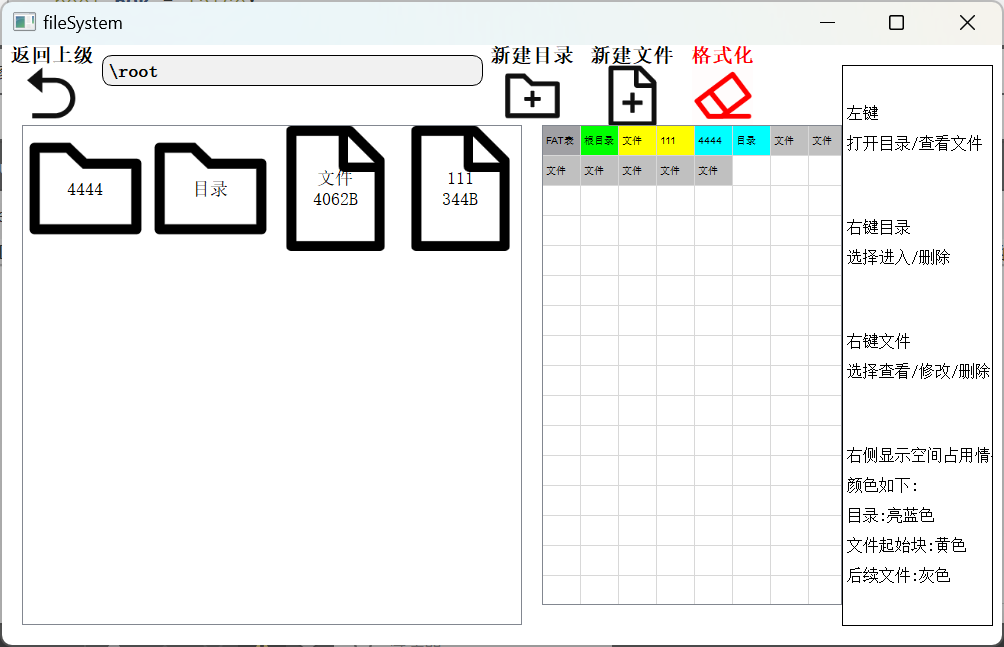
**提示是否有已保存的虚拟磁盘文件**

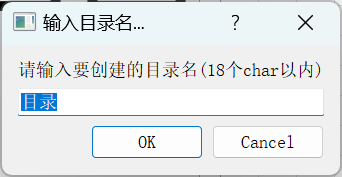
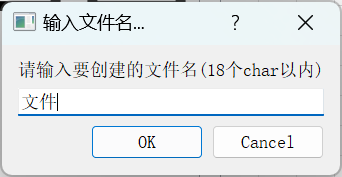


* 1. 主界面

**上方从左到右依次为：返回上级目录、当前路径显示、新建目录、新建文件、格式化。**

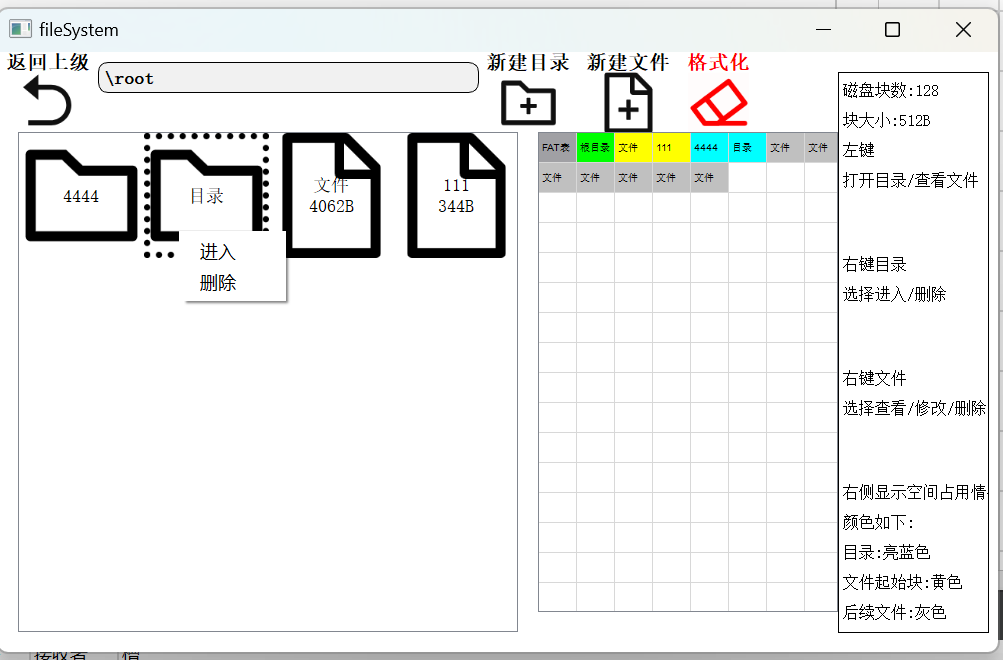
**下方左侧为当前目录下目录与文件列表（四行四列，同时16也是单个目录下存储的最多文件+目录的数量），中间为空间占用情况，右侧为界面操作说明。**

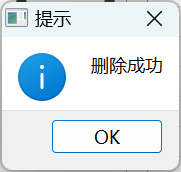
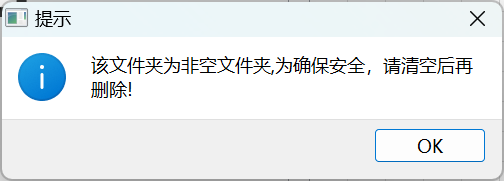


* 1. 目录操作界面

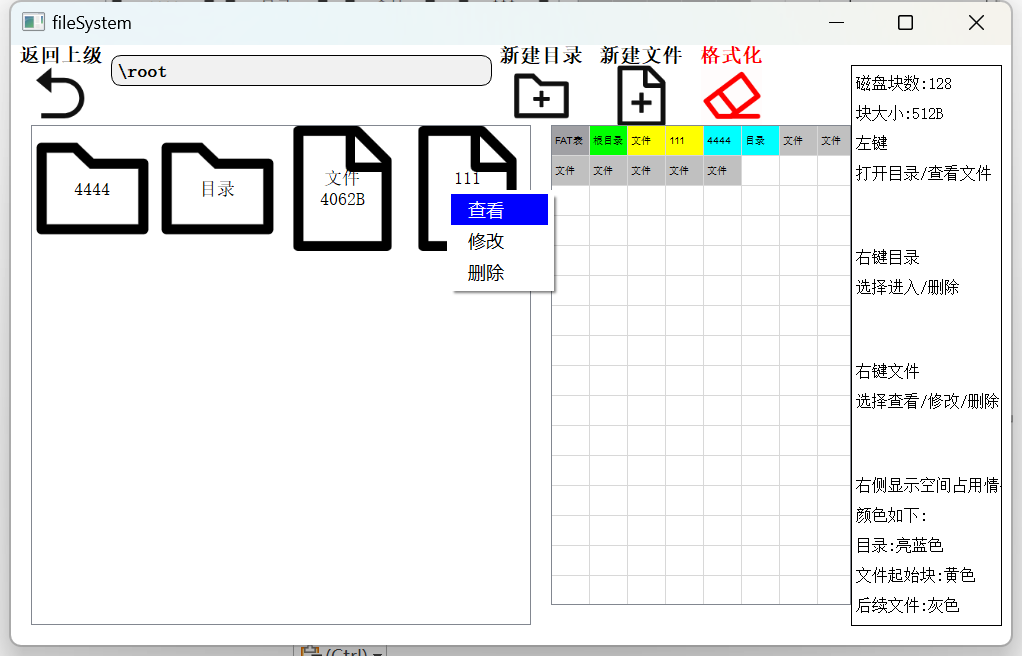
**右键后出现选择菜单(选中会蓝色高亮)。可选择进入或删除。删除会提示删除成功。若目标位非空目录，则会提醒用户清空后再尝试删除。**

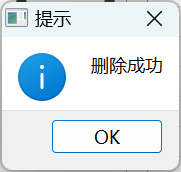
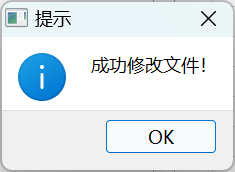
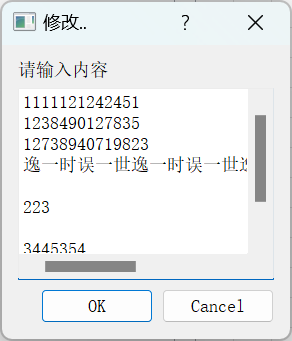
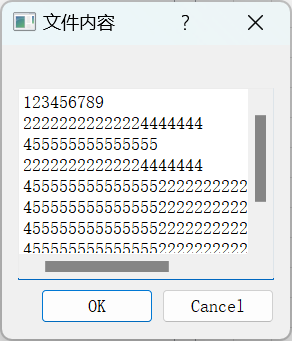


* 1. 文件操作界面

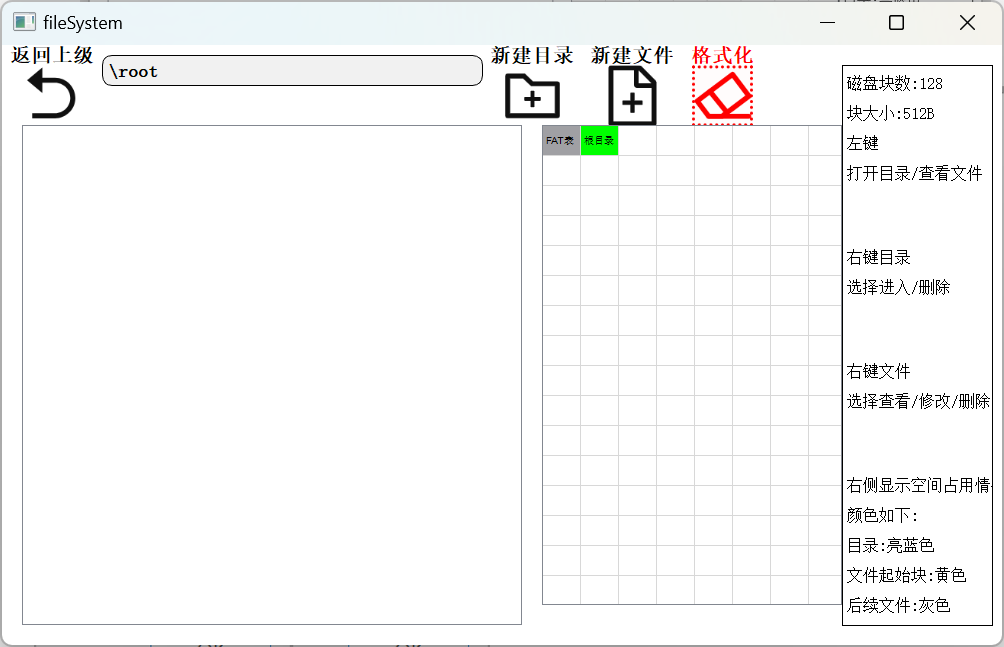
**右键后出现选择菜单(选中会蓝色高亮)。可选择查看、修改或删除。查看时也可以对文本进行编辑，但不会实际保存。删除同样提示删除成功。**

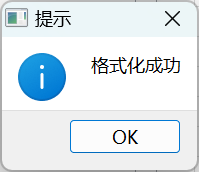




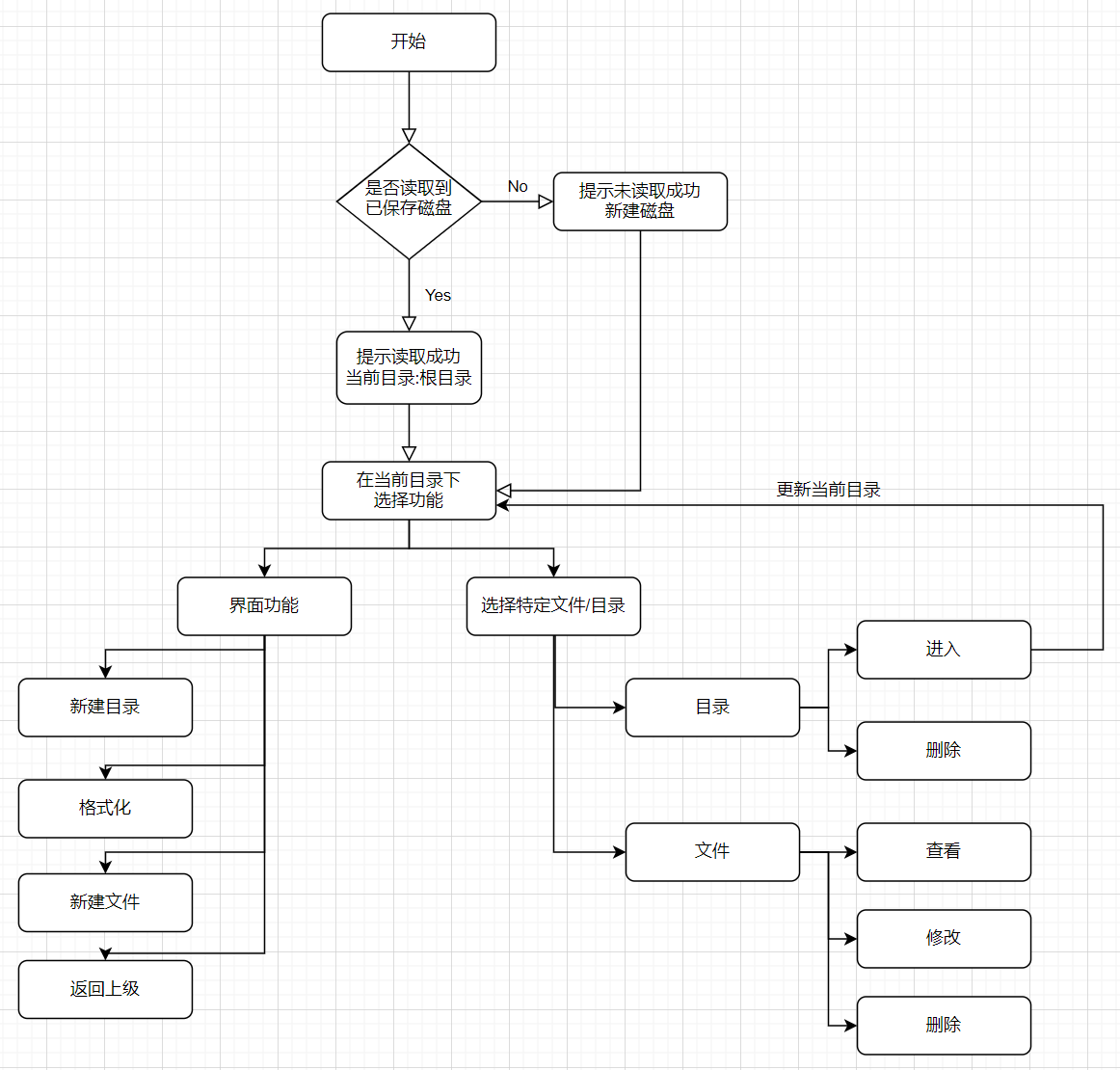
* 1. 格式化

**会提示格式化成功。格式化后文件系统会自动保存一次。**





1. 设计思路
   1. 主要逻辑部分设计思路



* + 1. **宏定义、常量、枚举**

#define FILE\_PATH "./FileSystem.dat" // 默认文件路径在项目根目录下

#define BLOCK\_SIZE 512                        // 模拟磁盘块大小(字节)

#define BLOCK\_NUM 128                         // 模拟磁盘块数

const int DISK\_SIZE = BLOCK\_SIZE \* BLOCK\_NUM; // 磁盘大小

#define MAX\_NAME 19 // 最长文件名

enum FILE\_TYPE { // FCB的type

  EMPTY,         // 空闲

  MYFILE,        // 文件

  DIRECTORY,     // 目录

};

enum FAT\_STATE { // FAT存储数值代表含义(在这之外的表示文件下一个盘块地址)

  FREE, // 空闲

  FILE\_END,

  //已分配，该块为文件结尾(创建文件初始化使用，等到写入长度过长再更改值为下一个块地址)

  DIR, // 已分配为目录

};

enum ERROR { //创建与删除文件/目录的返回信息

  SUCCESS,   //成功

  //创建文件

  DUPLICATED, //重名

  DISK\_FULL,  //磁盘已满

  DIR\_FULL,   //当前目录已满

  //删除文件

  NOT\_EMPTY, //当前目录非空

};

enum Path { //更改路径类别

  FORWARD,  //进入子目录

  BACK,     //返回上一级

};

* + 1. **主类的子结构体定义**

**FCB类（文件控制块）：**

////长度：19+1+4+4+4=32

struct FCB {

  char name[MAX\_NAME]; //文件名

  char type;           //文件类型

  int size;            //文件大小

  int ParentAddress;   //当前的上级目录地址(物理块号)

  int Address;         //当前目录地址

  FCB(const char \*name = "\0", int type = EMPTY, int Address = 0,

      int ParentAddress = 0, int size = 0) {

    strcpy(this->name, name);

    this->type = type;

    this->size = size;

    this->Address = Address;

    this->ParentAddress = ParentAddress;

  }

};

const int MAX\_DIRECTORY = BLOCK\_SIZE / sizeof(FCB) - 1; //目录文件的最多FCB数

**Directory类（目录）：(刚好128B，即一个物理块的长度)**

struct Directory //目录结构

{

  FCB DirectoryFCB; //自身FCB

  FCB SonFCB[MAX\_DIRECTORY];

  Directory(const char \*name = "\0", int Address = 0, int ParentAddress = 0) {

    DirectoryFCB =

        FCB(name, DIRECTORY, Address, ParentAddress, DirectoryFCB.size);

    for (int i = 0; i < MAX\_DIRECTORY; ++i) { //子FCB的上级目录地址为当前块号

      SonFCB[i] = FCB("\0", EMPTY, 0, Address, 0);

    }

  }

};

**Disk类（虚拟磁盘）：**

**虚拟磁盘模拟了BLOCK-NUM=126个物理块，以字符形式存储目录或文件内容，FAT表和根目录对象则模拟存储在第0和第1块中，这一点在界面的文件占用情况图中也有显示。**

struct Disk {

  int FAT[BLOCK\_NUM];    // FAT

  struct Directory root; //根目录

  char data[BLOCK\_NUM - 2][BLOCK\_SIZE];

  static const int ROOT = 1;

  Disk() {

    memset(FAT, 0, sizeof(FAT));

    FAT[0] = FAT[1] = DIR; // 0,1盘块号依次代表FAT,根目录区

    root = Directory("\\root", ROOT, ROOT); //根目录区

    memset(data, '\0', sizeof(data));

  }

};

* + 1. **主类(fileSystem)的定义：**

**有一个虚拟磁盘指针\*pFile，供fstream读写文件的指针fp、磁盘基地址（char\*）Disk\_Address，记录已用物理块数量等变量，以及qt窗口ui指针。**

class fileSystem : public QMainWindow {

  Q\_OBJECT

public:

  //数据操作部分：

  void Format();

  //格式化

  void SaveFS();

  //保存文件系统

  bool LoadFS();

  //载入文件系统

  //返回值：是否读入成功

  Directory \*LocateAddress(const int dir\_num);

  // 根据指定块号的目录定位并返回对应目录指针(默认盘块号为当前目录)

  // 返回值：指定块号的目录对应目录指针(根目录或转换为\*Directory的物理块)

  int FindSon(Directory \*dir, const char \*name, const char type);

  // 查找到当前目录下对应名称、对应类别的SonFCB对应下标

  // 返回值：查找成功则返回盘块号，查找失败则返回-1

  int Create(const char \*NewName, const char type);

  // 在当前目录下创建目录/文件

  // 返回值：创建状态信息(0为成功，1为当前目录下有同名且同类别的文件，2为磁盘已满，3为当前目录已满)

  bool Delete(const char \*DelName, const char type);

  // 删除当前目录下的目录/文件

  // 返回值：删除是否成功

  bool ChangePath(const char type, const char \*DirName = "\0");

  //更改当前路径，type为更改类别(0为进入，1为返回上一级)

  //返回值：是否更改路径成功(已在根目录时不能返回上一级)

  string ReadFile(const FCB \*OpenFCB);

  //查看文件内容

  bool write(const char \*content, FCB \*OpenFCB); //修改文件

  //界面绘制部分：

  void DrawCell(const int row, const int col, FCB \*SonFCB);

  //绘制文件列表中单元格：目录/文件

  void DrawTable();

  //绘制文件列表界面

  void DrawBitMap(const Directory \*dir);

  //绘制文件占用位图

  fileSystem(QWidget \*parent = NULL);

  ~fileSystem() {

    //析构函数中，即程序退出时存储文件

    SaveFS();

    delete ui;

  };

public slots:                //槽函数

  void CreateDirectoryFuc(); //新建文件夹

  void CreateFileFuc();      //新建文件

private:

  Disk \*pFile = new Disk();  //虚拟磁盘指针

  FILE \*fp = new FILE();     // fstream操作文件地址

  char \*Disk\_Address = NULL; //虚拟磁盘空间基地址

  int used\_block = 2;        //已用物理块数量

  Ui::fileSystem \*ui;

};

* + 1. **主类(fileSystem)重要成员函数的实现：**
       1. **读写文件函数：**

**WriteFile函数更新文件时，先找到对应物理块，将其后续文件的FAT表逐级清空（只需要FAT标记为空闲即可，物理块不需要清空），然后根据输入字符串长度，寻找空闲块并写入相应数据，同时更新各个块对应FAT。**

**ReadFile函数读入文件时，同样逐级读到当前块对应FAT表为FILE\_END为止。**

**WriteFile：**

bool fileSystem::WriteFile(const char \*content,

                       FCB \*OpenFCB) { //在指定的文件里记录信息

  char \*p = pFile->data[OpenFCB->Address - 2];

  int len = strlen(content);               //输入内容长度

  const int len\_per\_file = BLOCK\_SIZE - 1; //每个文件能存储的最大长度

  int file\_num = (len + len\_per\_file - 1) / len\_per\_file; //需要的文件数量

  if (file\_num > BLOCK\_NUM - used\_block) { //如果要新建的文件长度超出空闲长度

    messageBox("磁盘剩余空间不足！");

    return false;

  }

  //先把之前的后续文件全部置空

  int next\_Address = pFile->FAT[OpenFCB->Address];

  int tmp\_address;

  while (next\_Address != FILE\_END) {

    tmp\_address = next\_Address;

    next\_Address = pFile->FAT[next\_Address]; //找到后继文件

    pFile->FAT[tmp\_address] = FREE;

  }

  //很重要!!!!!

  pFile->FAT[OpenFCB->Address] = FILE\_END;

  OpenFCB->size = len;

  used\_block = file\_num;

  //先填写开始的当前文件

  strncpy(p, content, len\_per\_file); //拷贝长度为BLOCK\_SIZE-1=511的字符串

  p[len\_per\_file] = '\0';            //最后附上一个尾零

  char \*temp\_p = NULL;

  int temp\_address = OpenFCB->Address; //存储上一个填写的块地址

  //注意i=1开始，填入后续文件

  for (int i = 1; i < file\_num; ++i) {

    for (int j = 2; j < BLOCK\_NUM; ++j) { //从FAT[2]开始搜索,即除去FAT和根目录

      if (!pFile->FAT[j]) {               // FAT为空闲

        temp\_p = pFile->data[j - 2];  // 当前文件指针

        pFile->FAT[j] = FILE\_END;     // 标记为文件结尾

        pFile->FAT[temp\_address] = j; // 前一块的后继为当前块

        //按块装入

        strncpy(temp\_p, content + i \* len\_per\_file,

                len\_per\_file);  //拷贝长度为BLOCK\_SIZE-1=511的串到目标

        p[len\_per\_file] = '\0'; //最后附上一个尾零

        temp\_address = j;

        break; //注意这个break

      }

    }

  }

  messageBox("成功修改文件！");

  return true;

}

**ReadFile：**

//查看文件内容

string fileSystem::ReadFile(const FCB \*OpenFCB) {

  char \*p = pFile->data[OpenFCB->Address - 2];

  string content;

  int next\_Address = OpenFCB->Address;

  do { //未到文件结尾

    content += p;

    next\_Address = pFile->FAT[next\_Address]; //找到后继文件地址

  } while (next\_Address != FILE\_END);

  return content;

}

* + - 1. **Create函数：（新建文件/目录）**

**定位到当前目录，并根据type类别新建并初始化文件或目录，同时更新Fat表和占用情况。**

// 在当前目录下创建目录/文件

// 返回值：创建状态信息(0为成功，1为当前目录下有同名且同类别的文件，2为磁盘已满，3为当前目录已满)

int fileSystem::Create(const char \*NewName, char type) {

  Directory \*dir = LocateAddress();

  //查找目录下是否有同名文件夹

  if (FindSon(dir, NewName, type) != -1)

    return DUPLICATED;

  //查找空闲FCB

  for (int i = 0; i < MAX\_DIRECTORY; ++i) {

    if (dir->SonFCB[i].type == EMPTY) {

      //查找到空闲FCB，查找空闲物理块

      for (int j = 2; j < BLOCK\_NUM; ++j) { //从FAT[2]开始搜索,即除去FAT和根目录

        if (!pFile->FAT[j]) {               // FAT为空闲

          dir->SonFCB[i] = FCB(NewName, type, j, CurrentDirectory);

          pFile->FAT[j] = (type == DIRECTORY ? DIR : FILE\_END);

          if (type == DIRECTORY) {

            Directory \*NewDir =

                (Directory \*)(pFile->data[j - 2]); // 对应的字块需要-2

            \*NewDir = Directory(NewName, j, CurrentDirectory);

          } //上级目录设置为当前目录，位置设置为j(即找到的空闲物理块)

          ++used\_block; //已用物理块+1

          return SUCCESS;

        }

      }

      return DISK\_FULL;

    }

  }

  return DIR\_FULL;

}

* + - 1. **Delete函数：（删除文件/目录）**

**定位到当前目录，查找到对应物理块。若type为文件，则需要逐级删除其后续文件（同WriteFile函数中的操作，FAT置空即可）；若type为目录，则需要检查其是否非空并决定是否终止删除并返回信息。对应位置使用memset填充空白物理块，同时更新Fat表和占用情况。**

//删除当前目录下的目录/文件

//返回值：删除是否成功

bool fileSystem::Delete(const char \*DelName, const char type) {

  Directory \*dir = LocateAddress();

  int target = FindSon(dir, DelName, type); //对应下标

  if (target == -1) {

    qDebug() << "指定目录/文件不存在！";

    return false;

  }

  int address = dir->SonFCB[target].Address;

  if (type == DIRECTORY) { //如果删除目标为目录，则还需要考虑其子目录

    Directory \*DelDir =

        (struct Directory \*)(pFile->data[address - 2]); //查找到目标对应文件块

    for (int i = 0; i < MAX\_DIRECTORY; i++) { //查找目标目录是否为空目录

      if (DelDir->SonFCB[i].type != EMPTY) {

        messageBox("该文件夹为非空文件夹,为确保安全，请清空后再删除!");

        return false;

      }

    }

  }

  if (type ==

      MYFILE) { //如果删除目标为文件，则还需要删除其后续文件(FAT置零即可，不需要后续删除)

    int next\_Address = pFile->FAT[dir->SonFCB[target].Address];

    //如果该块不是文件结尾

    while (next\_Address != FILE\_END) {

      int tmp = next\_Address;

      next\_Address = pFile->FAT[next\_Address]; //找到后继文件

      pFile->FAT[tmp] = FREE;

    }

  }

  //更新位图

  pFile->FAT[address] = FREE; //清空FAT

  dir->SonFCB[target] = FCB(); //当前目录中，目标目录/文件对应FCB初始化

  char \*p = pFile->data[address - 2]; //格式化对应物理块

  memset(p, '\0', BLOCK\_SIZE);

  --used\_block; //已用物理块-1

  return true;

}

//更改当前路径，type为更改类别(0为进入，1为返回上一级)

//返回值：是否更改路径成功(已在根目录时不能返回上一级)

bool fileSystem::ChangePath(const char type, const char \*DirName) {

  Directory \*dir = LocateAddress();

  //返回上一级

  if (type == BACK) {

    if (CurrentDirectory == Disk::ROOT) {

      return false;

    }

    CurrentDirectory = dir->DirectoryFCB.ParentAddress;

    int lastSlashIndex =

        CurrentPath.lastIndexOf('\\'); // 查找最后一个反斜杠的索引

    if (lastSlashIndex != -1) {

      CurrentPath.remove(lastSlashIndex, CurrentPath.length() - lastSlashIndex);

      //删除最后一个反斜杠及其后面的内容

    }

    return true;

  }

  //进入子目录

  int target = FindSon(dir, DirName, DIRECTORY);

  if (target == -1) {

    qDebug() << "指定子目录不存在！";

    return false;

  }

  //修改当前文件信息

  CurrentDirectory = dir->SonFCB[target].Address;

  CurrentPath += "\\";

  CurrentPath += dir->SonFCB[target].name;

  return true;

}

* + - 1. **SaveFS、LoadFS系统保存与读入函数及Format格式化：**

**写入：使用fstream头文件下的相关操作，将FileSystem的虚拟磁盘以char形式写入指定路径的文件中**

**读入：读入指定路径的文件，将char转换为Disk\*形式，重新获取虚拟磁盘数据，并返回是否读入成功。**

**格式化：直接重置pFile指针指向新的Disk结构体，并直接写入保存文件系统。**

// 格式化

void fileSystem::Format() {

  CurrentDirectory = Disk::ROOT;

  CurrentPath = "\\root"; //当前路径

  \*pFile = Disk();        //打开文件列表初始化

  fp = fopen(FILE\_PATH, "w+");

  fwrite(Disk\_Address, sizeof(char), DISK\_SIZE, fp);

  fclose(fp);

}

// 保存文件系统

void fileSystem::SaveFS() {

  fp = fopen(FILE\_PATH, "w+");

  fwrite(Disk\_Address, sizeof(char), DISK\_SIZE, fp);

  fclose(fp);

  free(pFile);

}

// 载入文件系统

// 返回值：是否读入成功

bool fileSystem::LoadFS() {

  Disk\_Address = (char \*)malloc(DISK\_SIZE); //申请虚拟空间并且初始化

  pFile = (Disk \*)(Disk\_Address);           //虚拟磁盘初始化

  fp = fopen(FILE\_PATH, "r");

  if (fp != NULL) { //加载磁盘文件

    fread(Disk\_Address, sizeof(char), DISK\_SIZE, fp);

    return true;

  } else {

    Format();

    return false;

  }

}

* 1. 界面部分设计思路
     + 1. **fileSystem窗口构造函数：**

**弹出是否保存磁盘提示，绘制初始界面。**

fileSystem::fileSystem(QWidget \*parent)

    : QMainWindow(parent), ui(new Ui::fileSystem) {

  ui->setupUi(this);

  if (LoadFS())

    messageBox("检测到已保存磁盘，读入成功");

  else

    messageBox("未检测到已保存磁盘，已新建");

  QFont font("Arial", 5);

  QTableWidgetItem \*FATitem = new QTableWidgetItem("FAT表");

  FATitem->setBackground(Qt::gray);

  FATitem->setFont(font);

  ui->BitMap->setItem(0, 0, FATitem);

  QTableWidgetItem \*Rootitem = new QTableWidgetItem("根目录");

  Rootitem->setBackground(Qt::green);

  Rootitem->setFont(font);

  ui->BitMap->setItem(0, 1, Rootitem);

  ui->PathLabel->setText(CurrentPath);

  DrawTable();

  // 绑定按钮与函数

  connect(ui->FormatBtn, &QPushButton::clicked, this, [=]() {

    Format();

    ui->PathLabel->setText(CurrentPath);

    DrawTable();

    messageBox("格式化成功");

  });

  connect(ui->BackBtn, &QPushButton::clicked, this, [=]() {

    if (!ChangePath(BACK))

      messageBox("已在根目录下，无法返回");

    ui->PathLabel->setText(CurrentPath);

    DrawTable();

    ;

  });

  connect(ui->AddFileBtn, &QPushButton::clicked, this,

          &fileSystem::CreateFileFuc);

  connect(ui->AddDirBtn, &QPushButton::clicked, this,

          &fileSystem::CreateDirectoryFuc);

}

* + - 1. **DrawCell、DrawTable函数：**

**DrawCell：以按钮形式在列表特定位置绘制文件与目录，绑定相关函数等。**

**DrawTable：根据FAT占用情况，调用DrawCell刷新列表。**

**DrawCell：**

//绘制文件列表中单元格：目录/文件

void fileSystem::DrawCell(const int row, const int col, FCB \*SonFCB) {

  string content;

  QPushButton \*Btn = new QPushButton;

  if (SonFCB->type == DIRECTORY) {

    Btn->setText(QString(SonFCB->name));

    Btn->setStyleSheet("QPushButton {"

                       "  border-image: url(:/icons/Dir1.png);"

                       "}"

                       "QPushButton:hover {"

                       "  border-image: url(:/icons/Dir2.png);"

                       "}"

                       "QPushButton:pressed {"

                       "  border-image: url(:/icons/Dir3.png);"

                       "}");

    // 创建右键菜单

    QMenu \*menu = new QMenu(Btn);

    QAction \*enterAction = new QAction("进入", menu);

    QAction \*deleteAction = new QAction("删除", menu);

    menu->addAction(enterAction);

    menu->addAction(deleteAction);

    menu->setStyleSheet("QMenu::item:selected {"

                        "background-color:blue;"

                        "color:#FFFFFF;"

                        "}");

    // 关联按钮和右键菜单

    Btn->setContextMenuPolicy(Qt::CustomContextMenu);

    connect(Btn, &QPushButton::customContextMenuRequested,

            [=](const QPoint &pos) { menu->exec(Btn->mapToGlobal(pos)); });

    connect(Btn, &QPushButton::clicked, enterAction, &QAction::triggered);

    connect(enterAction, &QAction::triggered, [=]() {

      ChangePath(FORWARD, SonFCB->name);   // 更改路径

      ui->PathLabel->setText(CurrentPath); // 更新路径

      DrawTable();

    });

    connect(deleteAction, &QAction::triggered, [=]() {

      if (Delete(SonFCB->name, DIRECTORY))

        messageBox("删除成功");

      else

        messageBox("删除失败，请检查目标目录是否为空");

      DrawTable();

    });

  } else if (SonFCB->type == MYFILE) {

    Btn->setText(QString(SonFCB->name) + "\n" + QString::number(SonFCB->size) +

                 "B");

    Btn->setStyleSheet("QPushButton {"

                       "  border-image: url(:/icons/File1.png);"

                       "}"

                       "QPushButton:hover {"

                       "  border-image: url(:/icons/File2.png);"

                       "}"

                       "QPushButton:pressed {"

                       "  border-image: url(:/icons/File3.png);"

                       "}");

    // 创建右键菜单

    QMenu \*menu = new QMenu(Btn);

    QAction \*viewAction = new QAction("查看", menu);

    QAction \*modifyAction = new QAction("修改", menu);

    QAction \*deleteAction = new QAction("删除", menu);

    menu->addAction(viewAction);

    menu->addAction(modifyAction);

    menu->addAction(deleteAction);

    menu->setStyleSheet("QMenu::item:selected {"

                        "background-color:blue;"

                        "color:#FFFFFF;"

                        "}");

    // 关联按钮和右键菜单，以及关联按钮单击事件

    Btn->setContextMenuPolicy(Qt::CustomContextMenu);

    connect(Btn, &QPushButton::customContextMenuRequested,

            [=](const QPoint &pos) { menu->exec(Btn->mapToGlobal(pos)); });

    connect(Btn, &QPushButton::clicked, viewAction, &QAction::triggered);

    connect(viewAction, &QAction::triggered, [=]() {

      string defaultInput = ReadFile(SonFCB);

      QInputDialog \*Dialog = new QInputDialog(this);

      Dialog->setOptions(QInputDialog::NoButtons);

      Dialog->getMultiLineText(this, //弹出文本框

                               "文件内容", "", defaultInput.c\_str());

    });

    connect(modifyAction, &QAction::triggered, [=]() {

      string defaultInput = ReadFile(SonFCB);

      bool bOk = false;

      QString text = QInputDialog::getMultiLineText(this, //弹出输入文本框

                                                    "修改..", "请输入内容",

                                                    defaultInput.c\_str(), &bOk);

      if (bOk) {                                           //点击确定

        write((char \*)text.toStdString().c\_str(), SonFCB); //修改文件内容

      }

      DrawTable();

    });

    connect(deleteAction, &QAction::triggered, [=]() {

      if (Delete(SonFCB->name, MYFILE))

        messageBox("删除成功");

      else

        qDebug() << ("删除错误");

      DrawTable();

    });

  } else

    qDebug() << "插入视图出错！";

  //插入按钮

  ui->tableWidget->setCellWidget(row, col, Btn);

}

**DrawTable：**

//绘制文件列表界面

void fileSystem::DrawTable() {

  for (int i = 2; i < BLOCK\_NUM; ++i) {

    QTableWidgetItem \*item = new QTableWidgetItem("");

    item->setBackground(Qt::white);

    ui->BitMap->setItem(i / 8, i % 8, item);

  }

  DrawBitMap(LocateAddress(Disk::ROOT));

  Directory \*dir = LocateAddress();

  int SonNum = 0;

  //先显示目录

  for (int i = 0; i < MAX\_DIRECTORY; ++i) {

    if (dir->SonFCB[i].type == EMPTY)

      continue;

    if (dir->SonFCB[i].type == DIRECTORY) { //查找目录文件

      DrawCell(SonNum / 4, SonNum % 4, &dir->SonFCB[i]);

      ++SonNum;

    }

  }

  //再显示文件

  for (int i = 0; i < MAX\_DIRECTORY; ++i) {

    if (dir->SonFCB[i].type == EMPTY)

      continue;

    if (dir->SonFCB[i].type == MYFILE) { //查找普通文件

      DrawCell(SonNum / 4, SonNum % 4, &dir->SonFCB[i]);

      ++SonNum;

    }

  }

  for (int i = SonNum; i < 16; ++i) //清空多余单元格(自动delete旧单元格内容)

    ui->tableWidget->setCellWidget(i / 4, i % 4, new QLabel());

}

* + - 1. **DrawCell函数：**

**以按钮形式在列表特定位置绘制文件与目录，绑定相关函数等。**

//绘制文件列表中单元格：目录/文件

void fileSystem::DrawCell(const int row, const int col, FCB \*SonFCB) {

  string content;

  QPushButton \*Btn = new QPushButton;

  if (SonFCB->type == DIRECTORY) {

    Btn->setText(QString(SonFCB->name));

    Btn->setStyleSheet("QPushButton {"

                       "  border-image: url(:/icons/Dir1.png);"

                       "}"

                       "QPushButton:hover {"

                       "  border-image: url(:/icons/Dir2.png);"

                       "}"

                       "QPushButton:pressed {"

                       "  border-image: url(:/icons/Dir3.png);"

                       "}");

    // 创建右键菜单

    QMenu \*menu = new QMenu(Btn);

    QAction \*enterAction = new QAction("进入", menu);

    QAction \*deleteAction = new QAction("删除", menu);

    menu->addAction(enterAction);

    menu->addAction(deleteAction);

    menu->setStyleSheet("QMenu::item:selected {"

                        "background-color:blue;"

                        "color:#FFFFFF;"

                        "}");

    // 关联按钮和右键菜单

    Btn->setContextMenuPolicy(Qt::CustomContextMenu);

    connect(Btn, &QPushButton::customContextMenuRequested,

            [=](const QPoint &pos) { menu->exec(Btn->mapToGlobal(pos)); });

    connect(Btn, &QPushButton::clicked, enterAction, &QAction::triggered);

    connect(enterAction, &QAction::triggered, [=]() {

      ChangePath(FORWARD, SonFCB->name);   // 更改路径

      ui->PathLabel->setText(CurrentPath); // 更新路径

      DrawTable();

    });

    connect(deleteAction, &QAction::triggered, [=]() {

      if (Delete(SonFCB->name, DIRECTORY))

        messageBox("删除成功");

      else

        messageBox("删除失败，请检查目标目录是否为空");

      DrawTable();

    });

  } else if (SonFCB->type == MYFILE) {

    Btn->setText(QString(SonFCB->name) + "\n" + QString::number(SonFCB->size) +

                 "B");

    Btn->setStyleSheet("QPushButton {"

                       "  border-image: url(:/icons/File1.png);"

                       "}"

                       "QPushButton:hover {"

                       "  border-image: url(:/icons/File2.png);"

                       "}"

                       "QPushButton:pressed {"

                       "  border-image: url(:/icons/File3.png);"

                       "}");

    // 创建右键菜单

    QMenu \*menu = new QMenu(Btn);

    QAction \*viewAction = new QAction("查看", menu);

    QAction \*modifyAction = new QAction("修改", menu);

    QAction \*deleteAction = new QAction("删除", menu);

    menu->addAction(viewAction);

    menu->addAction(modifyAction);

    menu->addAction(deleteAction);

    menu->setStyleSheet("QMenu::item:selected {"

                        "background-color:blue;"

                        "color:#FFFFFF;"

                        "}");

    // 关联按钮和右键菜单，以及关联按钮单击事件

    Btn->setContextMenuPolicy(Qt::CustomContextMenu);

    connect(Btn, &QPushButton::customContextMenuRequested,

            [=](const QPoint &pos) { menu->exec(Btn->mapToGlobal(pos)); });

    connect(Btn, &QPushButton::clicked, viewAction, &QAction::triggered);

    connect(viewAction, &QAction::triggered, [=]() {

      string defaultInput = ReadFile(SonFCB);

      QInputDialog \*Dialog = new QInputDialog(this);

      Dialog->setOptions(QInputDialog::NoButtons);

      Dialog->getMultiLineText(this, //弹出文本框

                               "文件内容", "", defaultInput.c\_str());

    });

    connect(modifyAction, &QAction::triggered, [=]() {

      string defaultInput = ReadFile(SonFCB);

      bool bOk = false;

      QString text = QInputDialog::getMultiLineText(this, //弹出输入文本框

                                                    "修改..", "请输入内容",

                                                    defaultInput.c\_str(), &bOk);

      if (bOk) {                                           //点击确定

        write((char \*)text.toStdString().c\_str(), SonFCB); //修改文件内容

      }

      DrawTable();

    });

    connect(deleteAction, &QAction::triggered, [=]() {

      if (Delete(SonFCB->name, MYFILE))

        messageBox("删除成功");

      else

        qDebug() << ("删除错误");

      DrawTable();

    });

  } else

    qDebug() << "插入视图出错！";

  //插入按钮

  ui->tableWidget->setCellWidget(row, col, Btn);

}

* + - 1. **DrawBitmap函数：**

**根据FAT表情况绘制文件占用情况图。**

//绘制文件占用位图

void fileSystem::DrawBitMap(const Directory \*dir) {

  for (int i = 0; i < MAX\_DIRECTORY; ++i) {

    if (dir->SonFCB[i].type == EMPTY)

      continue;

    QTableWidgetItem \*item = new QTableWidgetItem(dir->SonFCB[i].name);

    QFont font("Arial", 5);

    item->setFont(font);

    if (dir->SonFCB[i].type == DIRECTORY) {

      item->setBackground(Qt::cyan);

      DrawBitMap(LocateAddress(dir->SonFCB[i].Address));

    } else {

      item->setBackground(Qt::yellow);

      int next\_Address = pFile->FAT[dir->SonFCB[i].Address];

      //如果该块不是文件结尾

      while (next\_Address != FILE\_END) {

        QTableWidgetItem \*additem = new QTableWidgetItem(dir->SonFCB[i].name);

        QFont font("Arial", 5);

        additem->setFont(font);

        additem->setBackground(Qt::lightGray);

        ui->BitMap->setItem(next\_Address / 8, next\_Address % 8, additem);

        next\_Address = pFile->FAT[next\_Address]; //找到后继文件

      }

    }

    ui->BitMap->setItem(dir->SonFCB[i].Address / 8, dir->SonFCB[i].Address % 8,

                        item);

  }

}