

文件管理 - 文件系统

操作系统第三次课程作业 - 文件系统

!!! 请勿删除可执行程序目录下的 `BitMapInfo.txt`, `CategoryInfo.txt`, `MyDiskInfo.txt`, 及其他配置文件, 否则可能导致文件系统无法正常运行!!!

文件管理 - 文件系统

操作系统第三次课程作业 - 文件系统

项目需求

基本任务

功能描述

项目目的

开发环境

项目结构

操作说明

系统分析

显示链接法

位图、FAT表

系统设计

界面设计

主界面MainForm

帮助界面HelpForm

记事本界面NoteForm

类设计

目录项FCB

虚拟磁盘类

目录类

目录结点类

状态设计

系统实现

虚拟磁盘

给文件分配空间并添加内容

读取文件内容

删除文件内容

更新文件内容

目录

清空某目录(当参数为root是为格式化)

搜索文件

在文件夹中创建文件

删除文件夹/文件

判断同级目录下是否重名

寻找该目录下根目录的名称

记事本窗口

加载记事本

关闭记事本

主窗口

依照目录加载文件系统

新建文件夹/文件

删除文件夹/文件

- 返回上一级目录
- 格式化
- 目录树
 - 创建目录树
 - 双击目录树结点
 - 目录树中加入结点
- 日志信息
 - 更新日志信息
 - 读/写目录信息
 - 读/写位图文件
 - 读/写虚拟磁盘文件
- 功能实现截屏展示
- 帮助信息界面
- 新建文件/文件夹
- 打开/删除文件项
- 编辑文本文件
- 查看文件信息
- 展开目录树
- 返回上级目录
- 格式化磁盘
- 创建最大数量的文件项
- 退出系统重新进入后恢复目录
- 作者

项目需求

基本任务

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，以便下次可以将其恢复到内存中来。

功能描述

- 文件存储空间管理可采取显式链接（如FAT）或者其他方法。（即自选一种方法）
- 空闲空间管理可采用位图或者其他方法。如果采用了位图，可将位图和FAT表合二为一。
- 文件目录采用多级目录结构。至于是否采用索引节点结构，自选。目录项目中应包含：文件名、物理地址、长度等信息。同学可在这里增加一些其他信息。
- 文件系统提供的操作：
 - 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

项目目的

- 熟悉文件存储空间的管理；
- 熟悉文件的物理结构、目录结构和文件操作；
- 熟悉文件系统管理实现；
- 加深对文件系统内部功能和实现过程的理解

开发环境

- **开发环境:** Windows 10
- **开发软件:**
Visual Studio 2017 15.9.28307.665
- **开发语言:** C#

项目结构

```
| BitMapInfo.txt
| CategoryInfo.txt
| FileManageSystem.exe
| FileManageSystem.exe.config
| FileManageSystem.pdb
| MyControl.dll
| MyControl.pdb
| MyDiskInfo.txt
| README.md
| 文件管理系统设计方案报告.md
| 文件管理系统设计方案报告.pdf
|
|—Resources
| | file18.png
| | file25.png
| | fileopen48.ico
| | folder18.png
| | folder25.png
| |
| |└icon
| |help.ico
| |icon.ico
| |note.ico
|
|—src | Category.cs
```

| FCB.cs
| Program.cs
| VirtualDisk.cs
|
└─Form
└─HelpForm
| HelpForm.cs
| HelpForm.Designer.cs
| HelpForm.resx
|
└─MainForm
| MainForm.cs
| MainForm.Designer.cs
| MainForm.resx
|
└─NoteForm
NoteForm.cs
NoteForm.Designer.cs
NoteForm.resx

操作说明

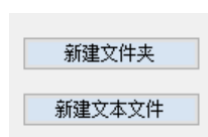
- 双击目录下 `FileManagementSystem.exe` 可执行文件进入文件系统模拟界面



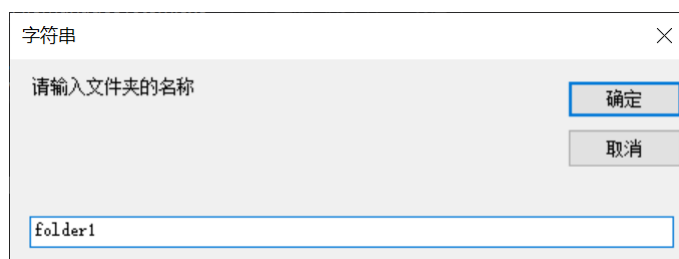
- 请详细阅读**操作帮助**了解模拟器功能, 点击 **我知道了** 关闭**帮助信息窗口**
- 单击**鼠标右键**, 新建文件夹/新建文件



- 您也可以点击**右侧按钮**进行创建文件夹/文本文件



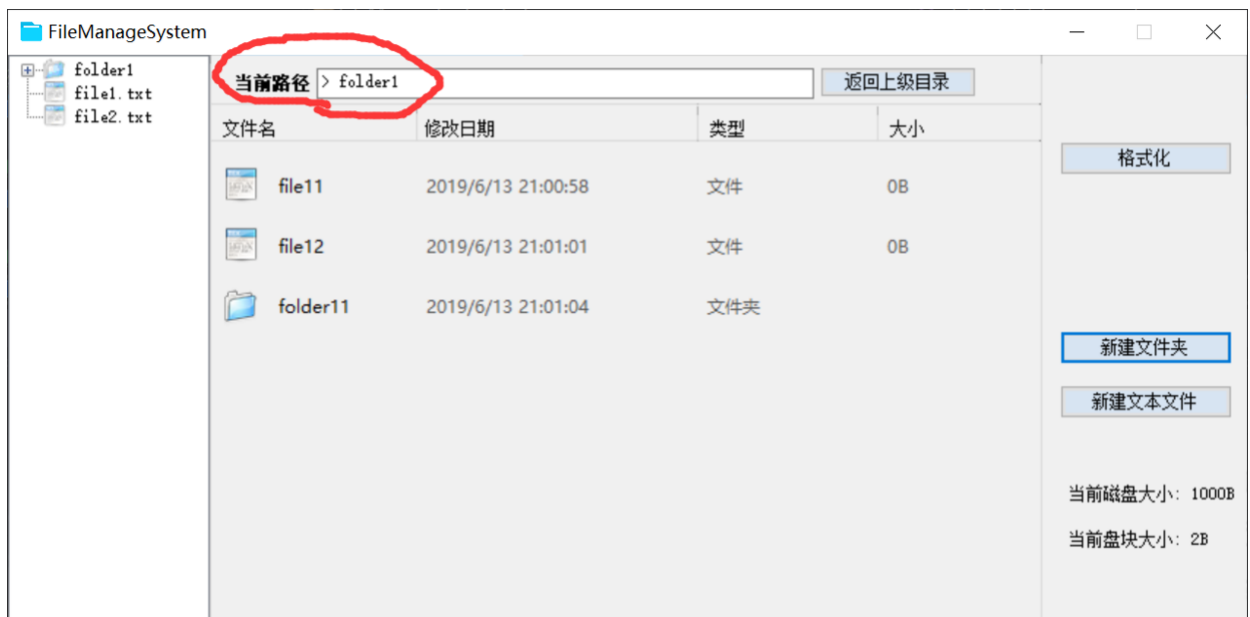
- 输入文件夹名或文件名, 即可在目录中查看到创建的文件夹/文件



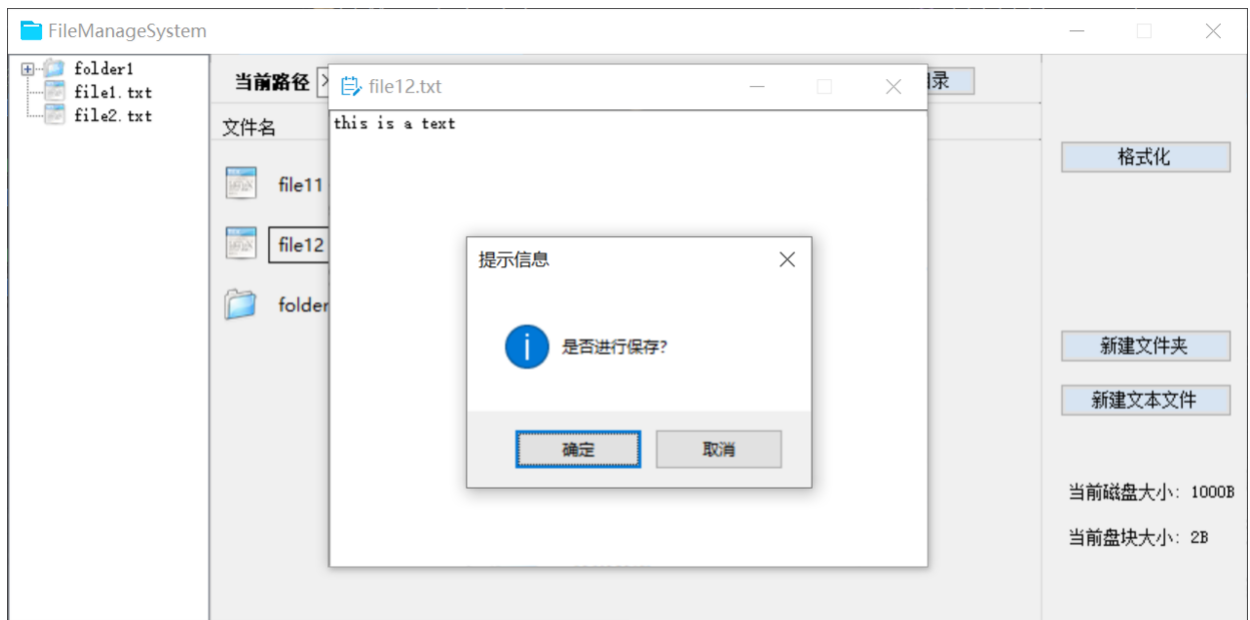
- 在文件夹/文件上点击鼠标右键可选择打开/删除



- 单击文件夹可进入下一级文件夹, 上方可查看当前文件路径



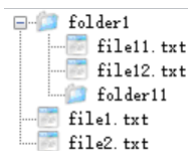
- 单击文件可打开编辑窗口, 编辑完成后点击右上角的 x, 确定以保存, 取消则直接退出



- 可查看**文件相应信息**(文件名, 上次修改日期, 文件大小(自定义单位))

	file	2019/6/13 21:15:52	文件	150B
--	------	--------------------	----	------

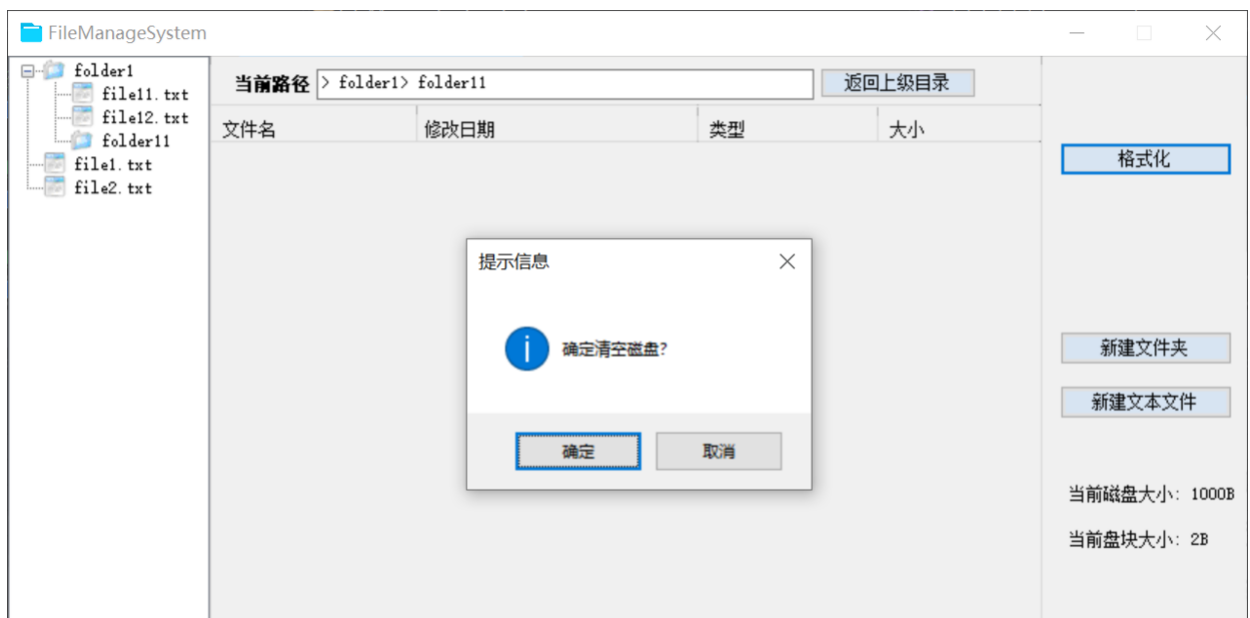
- 左侧的**目录树**可点击展开, 以查看当前目录结构



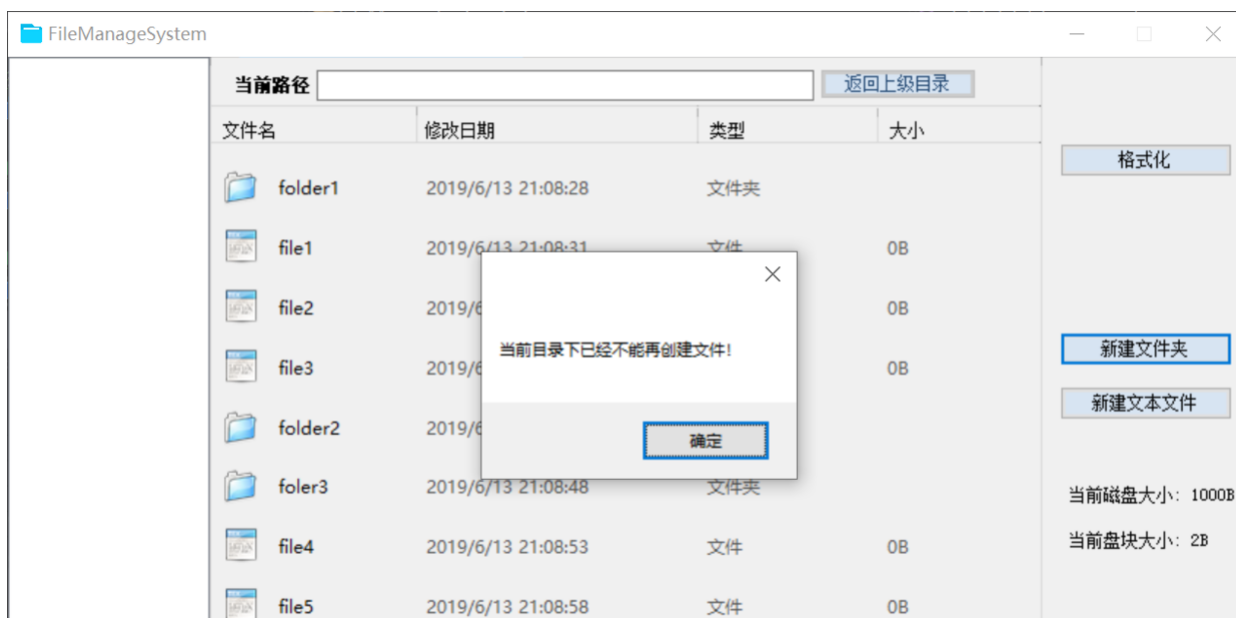
- **双击目录树**中的文件夹打开文件夹, 双击目录树中的文件可打开文件进行浏览和编辑
- 点击**返回上级目录**可跳转到上级文件夹, 在root文件夹时无法返回

当前路径 > folder1> folder11 返回上级目录

- 点击右侧**格式化**并再次确定可清空磁盘, 系统会清空所有文件夹和文本文件, 并清空目录树



- !!!受展示限制, 本文件系统模拟器在一个目录下最多可创建8个子项目, 超过8个时会受到系统提醒(只是受展示方式限制, 物理和逻辑上的存储理论上都允许创建无限多的子项目)



系统分析

显示链接法

本文件系统中, 文件存储空间管理使用**显示链接**的方法, 文件中的内容存放在磁盘不同的块中, 每次创建文件时为文件分配数量合适的空闲块。每次写文件时按顺序将文件内容写在相应块中; 删除文件时将原先有内容的位置置为空即可。

位图、FAT表

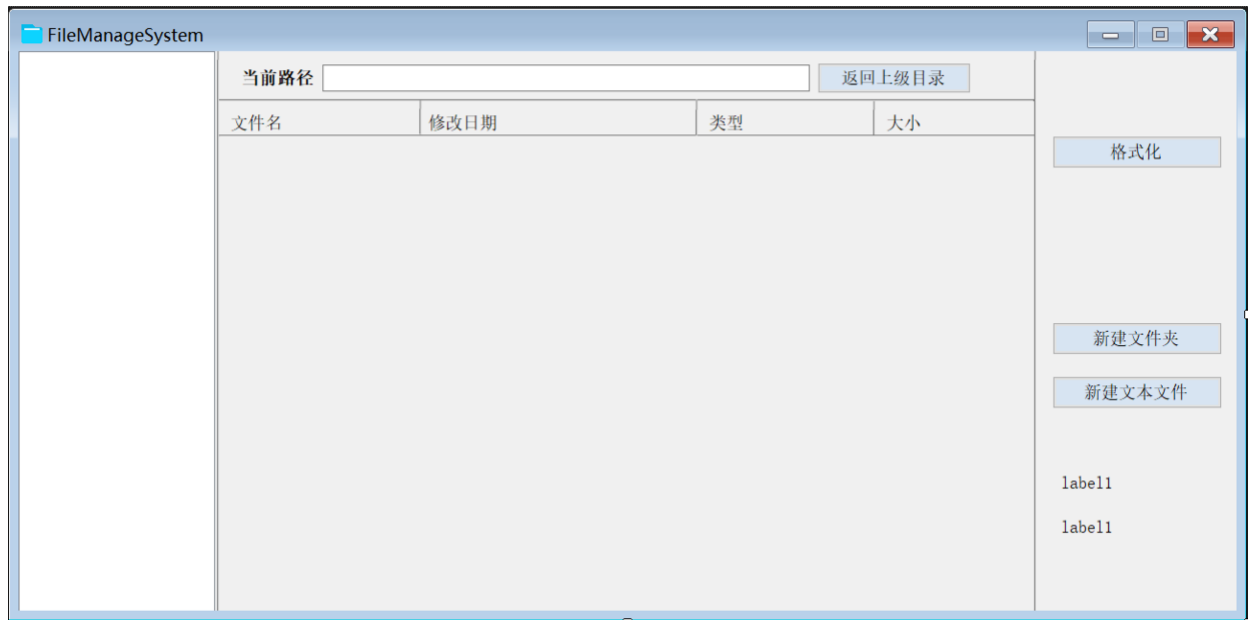
磁盘空闲空间管理在**位图**的基础上进行改造, 将存放磁盘上文件位置信息的**FAT表**与传统的位图进行结合, 磁盘空闲的位置使用 `EMPTY = -1` 标识, 放有文件的盘块存放文件所在的下一个盘块的位置, 文件存放结束的盘块位置使用 `END = -2` 标识。

系统设计

界面设计

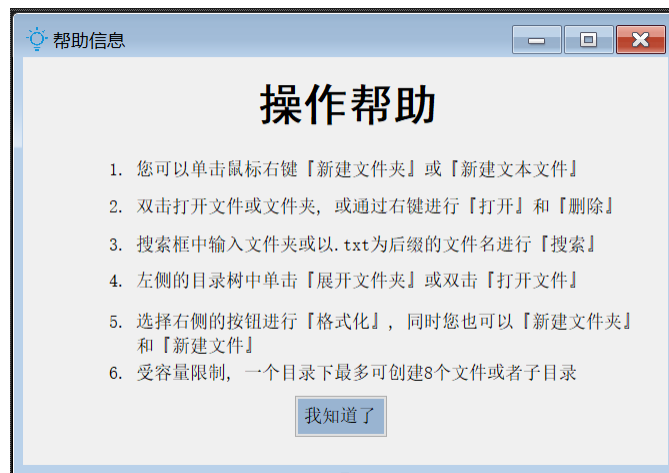
主界面MainForm

- System.Windows.Forms.Button
- System.Windows.Forms.Label
- System.Windows.Forms.TextBox
- System.Windows.Forms.TreeView



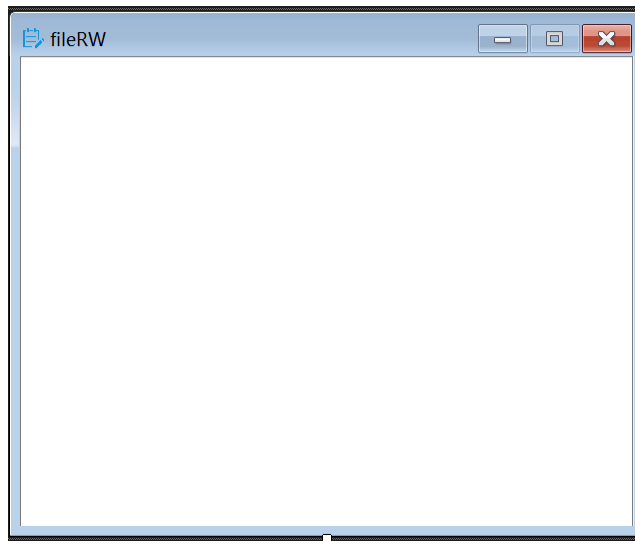
帮助界面HelpForm

- System.Windows.Forms.Label
- System.Windows.Forms.Button



记事本界面NoteForm

- System.Windows.Forms.TextBox



类设计

目录项FCB

```
//目录项
public class FCB
{
    public string fileName;           //文件名
    public int start;                 //文件在内存中初始存放的位置
    public int type;                  //文件类型 => TXTDILE/FOLDER
    public string lastModify;         //最近修改时间
    public int size;                  //文件大小,文件夹不显示大小

    public FCB() { }
    public FCB(string name,int type, string time, int size)
    {
        this.fileName = name;
        this.type = type;
        this.lastModify = time;
        this.size = size;
    }
    public FCB(string name, int type, string time, int size,int start)
    {
        this.fileName = name;
        this.type = type;
        this.lastModify = time;
        this.size = size;
        this.start = start;
    }
}
```

虚拟磁盘类

```
public class virtualDisk
{
    public int size;                  //磁盘容量
}
```

```

public int remain;           //磁盘剩余空间
public int blockSize;       //块大小
public int blockNum;        //磁盘块数
public string[] memory = new string[] { }; //内存空间
public int[] bitMap = new int[] { };      //位图表

public VirtualDisk(int size, int blockSize)
{
    this.size = size;
    this.blockSize = blockSize;
    this.blockNum = size / blockSize;
    this.remain = blockNum;

    this.memory = new string[blockNum];
    this.bitMap = new int[blockNum];
    for (int i = 0; i < blockNum; i++)
    {
        this.bitMap[i] = EMPTY; //初始化位图表为全部可用
        this.memory[i] = "";   //初始化内存内容为空
    }
}

/*给文件分配空间并添加内容*/
public bool giveSpace(FCB fcb, string content)
{...}

/*读取文件内容*/
public string getFileContent(FCB fcb)
{...}

/*删除文件内容*/
public void deleteFileContent(int start,int size)
{...}

/*更新文件内容*/
public void fileUpdate(int oldStart,int oldSize, FCB newFcb, string newContent)
{...}

/*获取所占块数*/
private int getBlockSize(int size)
{
    return size / blockSize + (size % blockSize != 0 ? 1 : 0);
}
}

```

目录类

```

public class Category
{
    public Node root; //目录的根节点
}

```

```

public Category()
{
    root = null;
}
public Category(FCB rootName)
{
    root = new Node(rootName);
    if (root == null)
    { return; }
}

/*清空某目录(当参数为root是为格式化)*/
public void freeCategory(Node pNode)
{...}

/*删掉结点*/
public void delete(Node pNode)
{...}

/*搜索文件*/
public Node search(Node pNode, string fileName, int type)
{...}

/*在文件夹中创建文件*/
public void createFile(string parentFileName, FCB file)
{...}

/*删除文件夹*/
public void deleteFolder(string name)
{...}

/*删除文件*/
public void deleteFile(string name)
{...}

/*判断同级目录下是否重名*/
public bool noSameName(string name, Node pNode, int type)
{...}

/*寻找该目录下根目录的名称*/
public Node currentRootName(Node pNode, string name, int type)
{...}
}

```

目录结点类

```

//目录节点
public class Node
{
    //节点存储的数据
    public FCB fcb = new FCB();
}

```

```

public Node firstChild = null;    //左孩子
public Node nextBrother = null;  //右兄弟
public Node parent = null;       //父结点

public Node() { }
public Node(FCB file)
{
    fcb.fileName = file.fileName;
    fcb.lastModify = file.lastModify;
    fcb.type = file.type;
    fcb.size = file.size;
}
public Node(string name, int type)
{
    fcb.fileName = name;
    fcb.type = type;
}
}

```

状态设计

1. 文本文件标识: `public const int TXTFILE = 0;`
2. 文件夹标识: `public const int FOLDER = 1;`
3. 当前块为空: `public const int EMPTY = -1;`
4. 文件结束: `public const int END = -2;`

系统实现

虚拟磁盘

给文件分配空间并添加内容

- 获取FCB指向的文件大小
- 该文件大小大于剩余空间 => 无法存储下该文件, 直接返回
- 该文件大小小于剩余空间 => 通过位图找到第一个 `EMPTY` 位置, 从此位置开始存放 => 将FCB中的起始位置信息更新
- 从此位置开始逐项的存储信息 => 剩余空间减一 => 并以链接的方式存储信息
- 当处理到最后一个数据时 => 将位图中填入 `END` 标识作为文件尾

```

public bool givespace(FCB fcb, string content)
{
    int blocks = getBlockSize(fcb.size);

    if (blocks <= remain)
    {
        /*找到该文件开始存放的位置*/
        int start = 0;

```

```

for (int i = 0; i < blockNum; i++)
{
    if (bitMap[i] == EMPTY)
    {
        remain--;
        start = i;
        fcb.start = i;
        memory[i] = content.Substring(0, blockSize);

        break;
    }
}

/*从该位置往后开始存放内容*/
for (int j = 1, i = start + 1; j < blocks && i < blockNum; i++)
{
    if (bitMap[i] == EMPTY)
    {
        remain--;

        bitMap[start] = i; //以链接的方式存储每位数据
        start = i;

        if (blocks != 1)
        {
            if (j != blocks - 1)
            {
                memory[i] = content.Substring(j * blockSize, blockSize);
            }
            else
            {
                bitMap[i] = END; //文件尾
                if (fcb.size % blockSize != 0)
                {
                    memory[i] = content.Substring(j * blockSize, content.Length - j
* blockSize);
                }
                else
                {
                    memory[i] = content.Substring(j * blockSize, blockSize);
                }
            }
        }
        else
        { memory[i] = content; }

        j++; //找到一个位置
    }
}
return true;
}
else

```

```
{ return false; }  
  
}
```

读取文件内容

- FCB中的起始位置为 `EMPTH` 状态 => 该文件无内容, 直接返回
- FCB中的起始位置不为 `EMPTH` 状态 => 首先获取FCB指向的文件所占块的数量和起始地址
- 从起始位置开始逐一读取内存中的数据, 并拼接起来, 之后跳转到下一数据位置继续读取

```
public string getFileContent(FCB fcb)  
{  
    if (fcb.start == EMPTY)  
    { return ""; }  
    else  
    {  
        string content = "";  
        int start = fcb.start;  
        int blocks = getBlockSize(fcb.size);  
  
        int count = 0, i = start;  
        while(i < blockNum)  
        {  
            if (count == blocks)  
            {  
                break;  
            }  
            else  
            {  
                content += memory[i];           //内容拼接内存的一个单元的数据  
                i = bitMap[i];                   //跳转到位图指向的下一个存储单元  
                count++;  
            }  
        }  
  
        return content;  
    }  
}
```

删除文件内容

- 首先获取到被删除内容所占块的数量
- 从该内容的起始块位置开始, 逐个内存单元的清空虚拟磁盘上的信息
- 在删除操作中要首先记录即将跳转的位置(防止清空后找不到接下来的位置)
- 之后清空该位, 并跳转到下一个数据位置继续删除

```
public void deleteFileContent(int start, int size)  
{  
    int blocks = getBlockSize(size);  
  
    int count = 0, i = start;  
    while(i < blockNum)
```

```

{
    if (count == blocks)
    {
        break;
    }
    else
    {
        memory[i] = "";           //逐内存单元的清空
        remain++;

        int next = bitMap[i];     //先记录即将跳转的位置
        bitMap[i] = EMPTY;       //清空该位
        i = next;

        count++;
    }
}
}

```

更新文件内容

- 由于更新后的文件size与原文件不同, 因此不能采用直接覆盖的方式
- 首先调用方法删除原内容
- 再在虚拟磁盘上开辟新的块, 并按顺序在新的块中添加内容

```

public void fileUpdate(int oldStart,int oldSize, FCB newFcb, string newContent)
{
    deleteFileContent(oldStart, oldSize); //删除原内容
    giveSpace(newFcb, newContent);        //开辟新的块并添加内容
}

```

目录

清空某目录(当参数为root是为格式化)

当参数传递的为root时<=> 格式化磁盘

- 如果当前目录为空 => 不需要删除, 直接返回
- 左孩子不为空 => 递归调用清空方法删除左孩子的子树 => 将左孩子设置为空
- 右兄弟不为空 => 递归调用清空方法删除右兄弟的子树 => 将右兄弟设置为空
- 将自己设置为空

```

public void freeCategory(Node pNode)
{
    if (pNode == null)
    { return; }

    if (pNode.firstChild != null) //清空左孩子
    {
        freeCategory(pNode.firstChild);
        pNode.firstChild = null;
    }
}

```



```

    if (pNode.nextBrother != null) //清空右兄弟
    {
        freeCategory(pNode.nextBrother);
        pNode.nextBrother = null;
    }

    pNode = null; //将自己清除
}

```

搜索文件

- 如果该目录下没有文件 => 搜索失败, 直接犯规
- 如果左孩子不为空 且 左孩子文件类型匹配 且 左孩子文件名匹配 => 返回左孩子
- 如果右兄弟不为空 且 右兄弟文件类型匹配 且 右兄弟文件名匹配 => 返回右兄弟
- 递归的搜索左孩子子树 => 找到了 => 返回该结点
- 递归的搜索右兄弟子树 => 返回搜索结果(找到了/没找到)

```

public Node search(Node pNode, string fileName, int type)
{
    if (pNode == null)
    { return null; }
    if (pNode.fcb.fileName == fileName && pNode.fcb.type == type)
    { return pNode; }
    if (pNode.firstChild == null && pNode.nextBrother == null)
    { return null; }
    else
    {
        Node firstChild = search(pNode.firstChild, fileName, type); //递归的搜索左孩子的子树
        if (firstChild != null)
        { return firstChild; }
        else
        { return search(pNode.nextBrother, fileName, type); } //递归的搜索右兄弟的子树
    }
}

```

在文件夹中创建文件

- 如果是在root目录下新建文件 => 直接创建
- 如果实在其他目录下创建文件 => 获取到该目录的父文件夹
 - 如果父节点为空 => 无法创建
 - 如果父节点左孩子为空 => 该文件夹为空 => 新建的文件为第一个文件, 放到左孩子的位置 => 将新加入的文件父节点设置为当前目录结点
 - 该文件夹中已经有很多文件 => 顺序找到该文件夹下最后一个文件存储的位置 => 将文件添加在最后的位置 => 将父节点设置为当前目录

```

public void createFile(string parentFileName, FCB file)
{
    if (root == null)
    { return; }
    Node parentNode = search(root, parentFileName, FCB.FOLDER); //找到父文件夹
}

```

```

if (parentNode == null)
{ return; }
if (parentNode.firstChild == null) //该文件夹为空
{
    parentNode.firstChild = new Node(file); //新创建的文件为第一个文件，放到左孩子的位置
    parentNode.firstChild.parent = parentNode;
    return;
}
else
{
    Node temp = parentNode.firstChild;
    while (temp.nextBrother != null) //顺序找到该文件夹下最后一个文件存储的位置
        temp = temp.nextBrother;

    temp.nextBrother = new Node(file);
    temp.nextBrother.parent = parentNode;
}
}
}

```

删除文件夹/文件

- 调用 `search` 方法找到要删除的文件夹
- 获取要删除文件夹的父节点
- 如果要删除的文件夹是父文件夹中的第一项 => 更新父文件夹的左孩子
- 如果要删除的文件夹不是父文件夹中的第一项 => 寻找到该结点的哥哥结点 => 让其指向自己的弟弟
- 判断要删除的结点类型:
 - 如果待删的是文件夹 => 最后删除当前结点下的所有文件
 - 如果待删的是文件 => 将该文件结点释放即可

```

/*删除文件夹*/
public void deleteFolder(string name)
{
    Node currentNode = search(root, name, FCB.FOLDER); //找到要删除的结点
    Node parentNode = currentNode.parent;

    if (parentNode.firstChild == currentNode) //如果要删除的文件夹是父文件夹中的第一项
    { parentNode.firstChild = currentNode.nextBrother; } //更新父文件夹的左孩子
    else
    {
        Node temp = parentNode.firstChild;
        while (temp.nextBrother != currentNode)
        {
            temp = temp.nextBrother;
        }

        temp.nextBrother = currentNode.nextBrother; //找到该文件的哥哥，让其指向自己的弟弟
    }
}

```

```

    freeCategory(currentNode);          //删除当前结点下的所有文件
}

```

```

/*删除文件*/
public void deleteFile(string name)
{
    Node currentNode = search(root, name, FCB.TXTFILE);    //找到要删除的文件
    Node parentNode = currentNode.parent;

    if (parentNode.firstChild == currentNode)    //如果要删除的文件是父文件夹中的第一项
    { parentNode.firstChild = currentNode.nextBrother; }    //更新父文件夹的左孩子
    else
    {
        Node temp = parentNode.firstChild;
        while (temp.nextBrother != currentNode)
        {
            temp = temp.nextBrother;
        }

        temp.nextBrother = currentNode.nextBrother;
    }
    currentNode = null;
}

```

判断同级目录下是否重名

- 如果该结点的左孩子为空 => 该目录中无任何文件夹和文件 => 无重名文件并返回true
- 如果该结点的左孩子不为空 且 左孩子与当前文件类型相同 且 左孩子与当前文件同名 => 发现重名文件并返回false
- 沿着右兄弟逐一寻找是否有某个结点与当前文件类型相同 且 与当前文件同名:
 - 发现 => 发现重名文件并返回false
- 找到目录结尾 => 无重名文件并返回true

```

public bool noSameName(string name, Node pNode, int type)
{
    //pNode为该级目录的根节点
    pNode = pNode.firstChild;
    if (pNode == null)    //该目录中无任何文件夹和文件
    { return true; }
    if (pNode.fcb.fileName == name && pNode.fcb.type == type)    //第一个文件夹/文件重名
    { return false; }
    else
    {
        Node temp = pNode.nextBrother;
        while (temp != null)
        {
            if (temp.fcb.fileName == name && temp.fcb.type == type)
            { return false; }
            temp = temp.nextBrother;
        }
    }
}

```

```

    }
    return true; //不重名
}
}

```

寻找该目录下根目录的名称

- 如果当前目录为root目录 => 无根目录
- 如果当前目录中无任何文件项 => 直接返回
- 如果左孩子为目标目录 => 则该结点即为根目录
- 沿着右孩子逐一寻找是否为目标目录:
 - 如果该右孩子为文本文件 => 判断该文件是否为目标目录 => 如果是则返回该结点
 - 如果该右孩子为文件夹 => 递归地在该文件中寻找目标目录

```

public Node currentRootName(Node pNode, string name, int type)
{
    if (pNode == null)
    { return null; }
    if (pNode.firstChild == null)
    { return null; }
    else
    {
        if (pNode.firstChild.fcb.fileName == name && pNode.firstChild.fcb.type == type)
        { return pNode; }
        else
        {
            Node par = pNode;
            Node temp = pNode.firstChild.nextBrother;
            while (temp != null)
            {
                if (temp.fcb.fileName == name && temp.fcb.type == type)
                { return par; }
                else
                { temp = temp.nextBrother; }
            }
            if (currentRootName(par.firstChild, name, type) != null)    //递归地在左孩子子树中寻
找
            {
                return currentRootName(par.firstChild, name, type);
            }
            else
            {
                return currentRootName(par.firstChild.nextBrother, name, type); //递归地在右兄
弟子树中寻找
            }
        }
    }
}

```

记事本窗口

加载记事本

- 加载打开的文件名
- 从目录中找到该文件的目录项
- 如果该目录项对应的文件有内容 => 读取保存的文本文件信息并渲染到TextBox中
- 刚打开该文件 => 将是否修改过标识置为false

```
private void NoteForm_Load(object sender, EventArgs e)
{
    this.Text = filename + ".txt";
    FCB nowFCB = mainForm.category.search(mainForm.category.root, filename, 0).fcb;
    if (mainForm.MyDisk.getFileContent(nowFCB) != "")
    {
        textBox1.AppendText(mainForm.MyDisk.getFileContent(nowFCB));    //读取保存的文本文件信息
    }
    ischanged = false;
}
```

关闭记事本

- 如果用户更改过记事本 => 弹出提示框提醒保存
- 如果用户没更改过记事本 => 直接退出
- 如果用户确定保存 => 获取当前时间 => 在内存上给文件分配空间
 - 如果内存剩余空间不足 => 输出消息提示, 保存失败
 - 如果内存空间充足:
 - 如果该文本文件之前为空(第一次修改) => 调用 `giveSpace` 进行开辟
 - 如果之前更改过 => 调用 `fileUpdate` 进行更新
- 如果用户点击取消 => 不保存信息

```
protected override void OnFormClosing(FormClosingEventArgs e)
{
    if (ischanged == true)    //如果用户更改记事本 => 弹出提示框提醒保存
    {
        DialogResult result = MessageBox.Show("是否进行保存? ", "提示信息",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
        if (result == DialogResult.OK)
        {
            FCB nowFCB = mainForm.category.search(mainForm.category.root, filename, 0).fcb;
            int oldSize = nowFCB.size;
            int oldStart = nowFCB.start;

            string content = textBox1.Text.Trim();
            if (content != "")
            {
                MessageBox.Show("保存成功! ");
            }

            nowFCB.size = textBox1.Text.Trim().Length;
            nowFCB.lastModify = DateTime.Now.ToLocalTime().ToString();    //获取当前时间

            //在内存上给文件分配空间
        }
    }
}
```

```

        if (nowFCB.size>0)
        {
            if(mainForm.MyDisk.remain<=textBox1.Text.Trim().Length)
            {
                MessageBox.Show("磁盘空间不足! ");
            }
            else
            {
                if(oldStart==-1)    //如果该文本文件之前为空(第一次修改)
                {
                    mainForm.MyDisk.giveSpace(nowFCB, textBox1.Text.Trim());
                }
                else                //更新
                {
                    mainForm.MyDisk.fileUpdate(oldStart, oldSize,nowFCB,
textBox1.Text.Trim());
                }
            }
        }

        mainForm.fileFormInit(mainForm.currentRoot);
    }
    else
    {
        e.Cancel = false;  //不保存直接退出
    }
}
else
    e.Cancel = false;    //用户未编辑直接退出
}

```

主窗口

依照目录加载文件系统

- 获取当前磁盘大小和当前盘块大小
- 当前目录为root => 禁用返回上一层按钮
- 更新当前路径 => 从初始化窗体
- 按照左孩子-右兄弟树的结构, 依次显示该目录下的文件夹/文件

```

public void fileFormInit(Category.Node now)
{
    labelDiskSize.Text = "当前磁盘大小: " + MyDisk.size.ToString() + "B";
    labelBlockSize.Text = "当前盘块大小: " + MyDisk.blockSize.ToString() + "B";

    if (now.fcb.fileName == "root") //当前目录为root时禁用返回上一层按钮
        buttonBack.Enabled = false;
    else
        buttonBack.Enabled = true;
}

```

```

textBoxSearch.Text = nowPath;    //更新路径

string name = now.fcb.fileName;
//窗体初始化
this.filewindow.Init();

//按照左孩子-右兄弟树的结构，依次显示该目录下的文件夹/文件
if (now.firstChild == null)
    return;
Category.Node temp = new Category.Node();
temp = now.firstChild;
filewindow.showFiles(temp.fcb.fileName, temp.fcb.lastModify, temp.fcb.type,
temp.fcb.size);
temp = temp.nextBrother;
while (temp != null)
{
    FCB current = temp.fcb;
    filewindow.showFiles(temp.fcb.fileName, temp.fcb.lastModify, temp.fcb.type,
temp.fcb.size);
    temp = temp.nextBrother;
}
}

```

新建文件夹/文件

- 用户成功提交文件夹/文件名 => 判断文件名不为空
- 如果当前路径下没有重名的文件 => 获取时间信息 => 创建文件项 => 将文件项加入目录中
- 如果当前路径下有重名的文件 => 弹出对话框提醒用户已经有同名的文件 => 创建失败

```

public void createFile()
{
    string str = Interaction.InputBox("请输入文件的名称", "字符串", "", 100, 100);
    if (str != "")
    {
        if (category.noSameName(str, currentRoot, FCB.TXTFILE))
        {
            string time = DateTime.Now.ToLocalTime().ToString();    //获取时间信息
            if (filewindow.showFiles(str, time, FCB.TXTFILE, 0))
                category.createFile(currentRoot.fcb.fileName, new FCB(str, FCB.TXTFILE,
time, 0)); //文件加入到目录中
        }
        else
        {
            MessageBox.Show("已存在名为" + str + ".txt的文件，创建失败！");
        }
    }
    //目录树更新
    treeView.Nodes.Clear();
    createTreeView(rootNode.firstChild);
}

```

```

public void createFolder()
{
    string str = Interaction.InputBox("请输入文件夹的名称", "字符串", "", 100, 100);
    if (str != "")
    {
        if(str==currentRoot.fcb.fileName)    //如果跟上级根目录重名, 添加一个_做标识
        {
            str = "_" + str;
        }
        if (category.noSameName(str, currentRoot, FCB.FOLDER))
        {
            string time = DateTime.Now.ToLocalTime().ToString();
            if (filewindow.showFiles(str, time, FCB.FOLDER, 0))
                category.createFile(currentRoot.fcb.fileName, new FCB(str, FCB.FOLDER,
time, 0)); //文件夹加入到目录中
        }
        else
        {
            MessageBox.Show("已存在名为" + str + "的文件夹, 创建失败!");
        }
    }
    //目录树更新
    treeView.Nodes.Clear();
    createTreeView(rootNode.firstChild);
}

```

删除文件夹/文件

- 删除文件 => 在目录中找到该文件的FCB => 清除该文件结点 => 更新目录中该文件占用的项 => 将虚拟磁盘中该文件占用的位置置为可用 => 重新刷新界面
- 删除文件夹 => 在目录中删除该文件夹占用的项 => 重新刷新界面
- 更新目录树

```

public void delete(string name, int type)
{
    if (type == FCB.TXTFILE)    //删除文件
    {
        FCB deleteFCB = category.search(rootNode, name, type).fcb;
        category.deleteFile(name);
        category.delete(category.search(rootNode, name, FCB.TXTFILE));
        MyDisk.deleteFileContent(deleteFCB.start, deleteFCB.size);
        fileFormInit(currentRoot);
    }
    else    //删除文件夹
    {
        category.deleteFolder(name);
        fileFormInit(currentRoot);
    }
    //目录树更新
    treeView.Nodes.Clear();
    createTreeView(rootNode.firstChild);
}

```


返回上一级目录

- 获取当前节点的父节点
- 更新当前路径
- 刷新目录界面

```
private void buttonBack_Click(object sender, EventArgs e)
{
    nowPath = nowPath.Replace("> " + currentRoot.fcb.fileName, "");
    currentRoot = currentRoot.parent;
    fileFormInit(currentRoot);
}
```

格式化

- 用户点击格式化按钮 => **为安全起见**弹出对话框再次提醒用户是否确定删除
- 确定 => 删除目录项并刷新界面 => 清空虚拟内存 => 位图置为空 => 将剩余空间重新置为最大 => 更新所有日志文件

```
private void buttonDelete_Click(object sender, EventArgs e)
{
    DialogResult result = MessageBox.Show("确定清空磁盘?", "提示信息",
    MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
    if (result == DialogResult.OK)
    {
        category.freeCategory(category.root);
        for (int i = 0; i < MyDisk.blockSize; i++)
        {
            MyDisk.memory[i] = "";
            MyDisk.bitMap[i] = -1;
            MyDisk.remain = MyDisk.blockNum;
        }
        MessageBox.Show("磁盘已清空。");
        fileFormInit(rootNode);

        nowPath = "";
        textBoxSearch.Text = "";
        treeView.Nodes.Clear();

        updateLog();           //清空所有日志文件
    }
}
```

目录树

创建目录树

- 如果目录为空 => 不需要创建目录树

- 如果目录不空 => 对于文件夹和文件分别创建目录树结点(目录树中只需要一个左孩子和一个右兄弟的数据结构来存储)

```
public void createTreeview(Category.Node pNode)
{
    if (pNode == null)        //目录为空, 不需要创建目录树
        return;

    /*文件夹和文本文件分别创建目录树结点*/
    TreeNode tn = new TreeNode();
    if (pNode.fcb.type == FCB.FOLDER)
    {
        tn.Name = pNode.fcb.fileName;
        tn.Text = pNode.fcb.fileName;
        tn.Tag = 1;
        tn.ImageIndex = 1;
        tn.SelectedImageIndex = 1;
    }
    else if (pNode.fcb.type == FCB.TXTFILE)
    {
        tn.Name = pNode.fcb.fileName + ".txt";
        tn.Text = pNode.fcb.fileName + ".txt";
        tn.Tag = 0;
        tn.ImageIndex = 0;
        tn.SelectedImageIndex = 0;
    }

    /*只需按照一个左孩子一个右兄弟建立目录树*/
    if (pNode.parent == rootNode)
    {
        treeview.Nodes.Add(tn);
    }
    else
    {
        CallAddNode(treeview, pNode.parent.fcb.fileName, tn);
    }
    createTreeview(pNode.firstChild);
    createTreeview(pNode.nextBrother);
}
```

双击目录树结点

- 双击目录树中的文件夹 => 调转到该文件夹下的目录 => 更新当前路径
- 双击目录树中的文件 => 打开编辑界面

```
private void treeview_MouseDoubleClick(object sender, MouseEventArgs e)
{
    TreeNode tn = treeview.SelectedNode;

    if (tn.Tag.ToString() == "1") //打开文件夹
    {
        Stack<TreeNode> s = new Stack<TreeNode>();
    }
}
```

```

        s.Push(tn);
        currentRoot = category.search(rootNode, tn.Text, FCB.FOLDER);

        //更新路径
        nowPath = "";
        while (tn.Parent != null)
        {
            s.Push(tn.Parent);
            tn = tn.Parent;
        }
        nowPath = "";
        while (s.Count() != 0)
        {
            nowPath += "> " + s.Pop().Text;
        }
        textBoxSearch.Text = nowPath;

        //刷新新目录下的界面
        fileFormInit(currentRoot);
    }
    else if (tn.Tag.ToString() == "0") //打开文件
    {
        NoteForm file = new NoteForm(tn.Text.Replace(".txt", ""), this);
        file.Show();
    }
}

```

目录树中加入结点

```

public TreeNode AddNode(TreeNode tnParent, string tnStr, TreeNode newTn)
{
    if (tnParent == null)
        return null;
    if (tnParent.Name == tnStr)
    {
        tnParent.Nodes.Add(newTn);
    }

    TreeNode tnRet = null;
    foreach (TreeNode tn in tnParent.Nodes)
    {
        tnRet = AddNode(tn, tnStr, newTn);
        if (tnRet != null)
        {
            tnRet.Nodes.Add(newTn);
            break;
        }
    }
    return tnRet;
}

public TreeNode CallAddNode(Treeview tree, string tnStr, TreeNode newTn)
{

```

```

foreach (TreeNode n in tree.Nodes)
{
    TreeNode temp = AddNode(n, tnStr, newTn);
    if (temp != null)
        return temp;
}
return null;
}

```

日志信息

更新日志信息

- 将所有的内容写入到本地 `CategoryInfo.txt` 文件中, 用户退出文件系统后, 下次在登陆时可根据日志信息回复文件系统上的目录和其他信息
- 主窗体关闭时, 必须调用该方法把所有的内容写入到本地

```

public void updateLog()
{
    if (File.Exists(Application.StartupPath + "\\CategoryInfo.txt"))
        File.Delete(Application.StartupPath + "\\CategoryInfo.txt");
    Category.Node temp = new Category.Node();
    string path = Application.StartupPath;
    Queue<Category.Node> q = new Queue<Category.Node>();
    q.Enqueue(category.root);

    while (q.Count() != 0)
    {
        temp = q.Dequeue();
        temp = temp.firstChild;
        while (temp != null)
        {
            q.Enqueue(temp);
            writeCategory(temp);
            temp = temp.nextBrother;
        }
    }

    writeBitMap();
    writeMyDisk();
}

```

读/写目录信息

CategoryInfo.txt中信息格式

- 当前目录下的根节点
- 文件名称
- 文件类型
- 上次修改时间
- 文件大小

- 文件起始位置
- #分隔符

```
public void readFromDisk()
{
    //把本地保存的上次的结点信息写回来
    string path = Application.StartupPath + "\\CategoryInfo.txt";
    if (File.Exists(path))
    {
        StreamReader reader = new StreamReader(path);
        string parentName = "", Name = "";    //父结点名字, 自己的名字
        string lastModify = "";    //最后修改时间
        int type = -1, size = 0, start = -1, infoNum = 1;

        //逐行读取信息
        string str = reader.ReadLine();
        while (str != null)
        {
            switch (infoNum)
            {
                case 1:
                    parentName = str;
                    infoNum++;
                    break;
                case 2:
                    Name = str;
                    infoNum++;
                    break;
                case 3:
                    type = int.Parse(str);
                    infoNum++;
                    break;
                case 4:
                    lastModify = str;
                    infoNum++;
                    break;
                case 5:
                    size = int.Parse(str);
                    infoNum++;
                    break;
                case 6:
                    start = int.Parse(str);
                    infoNum++;
                    break;
                case 7:
                    infoNum = 1;
                    FCB now = new FCB(Name, type, lastModify, size);
                    category.createFile(parentName, now);    //把文件结点的内容加到目录中
                    break;
                default:
                    break;
            }
            str = reader.ReadLine();
        }
    }
}
```

```

    }
    reader.close();
}

}

```

```

public void writeCategory(Category.Node pNode)
{
    //当前结点的父结点
    Category.Node parentNode = category.currentRootName(rootNode, pNode.fcb.fileName,
pNode.fcb.type);
    string InfoPath = Application.StartupPath + "\\CategoryInfo.txt";
    StreamWriter writer = File.AppendText(InfoPath);

    writer.WriteLine(parentNode.fcb.fileName); //写入父结点的名字
    writer.WriteLine(pNode.fcb.fileName);      //写入文件的名字
    writer.WriteLine(pNode.fcb.type);           //写入文件的类型
    writer.WriteLine(pNode.fcb.lastModify);     //写入最后修改的时间
    writer.WriteLine(pNode.fcb.size);           //写入文件的大小
    if (pNode.fcb.type == FCB.TXTFILE)          //写入文件的开始位置
    {
        writer.WriteLine(pNode.fcb.start);
    }
    else if (pNode.fcb.type == FCB.FOLDER)       //若为文件夹则写入-1
    {
        writer.WriteLine(-1);
    }
    writer.WriteLine("#");                      //一个结点写完

    writer.close();
}

```

读/写位图文件

```

public void readBitMap()
{
    string path = Application.StartupPath + "\\BitMapInfo.txt";
    if (File.Exists(path))
    {
        StreamReader reader = new StreamReader(path);
        for (int i = 0; i < MyDisk.blockNum; i++)
        {
            MyDisk.bitMap[i] = int.Parse(reader.ReadLine());
        }
        reader.close();
    }
}

```

```

public void writeBitMap()
{
    if (File.Exists(Application.StartupPath + "\\BitMapInfo.txt"))
        File.Delete(Application.StartupPath + "\\BitMapInfo.txt");
    StreamWriter writer = new StreamWriter(Application.StartupPath + "\\BitMapInfo.txt");

    for (int i = 0; i < MyDisk.blockNum; i++)
    {
        writer.WriteLine(MyDisk.bitMap[i]);
    }
    writer.Close();
}

```

读/写虚拟磁盘文件

```

public void readMyDisk()
{
    string path = Application.StartupPath + "\\MyDiskInfo.txt";
    if (File.Exists(path))
    {
        StreamReader reader = new StreamReader(path);
        for (int i = 0; i < MyDisk.blockNum; i++)
        {
            string str = reader.ReadLine();
            if (str.IndexOf("(") >= 0)
                MyDisk.memory[i] = str.Replace("(", "\\r\\n");
            else
            {
                if (str.IndexOf('(') >= 0)
                    MyDisk.memory[i] = str.Replace('(', '\\r');
                else if (str.IndexOf(')') >= 0)
                    MyDisk.memory[i] = str.Replace(')', '\\n');
                else if (str != "#")
                    MyDisk.memory[i] = str;
                else if (str == "#")
                    MyDisk.memory[i] = "";
            }
        }
        reader.Close();
    }
}

```

```

public void writeMyDisk()
{
    if (File.Exists(Application.StartupPath + "\\MyDiskInfo.txt"))
        File.Delete(Application.StartupPath + "\\MyDiskInfo.txt");
    StreamWriter writer = new StreamWriter(Application.StartupPath + "\\MyDiskInfo.txt");

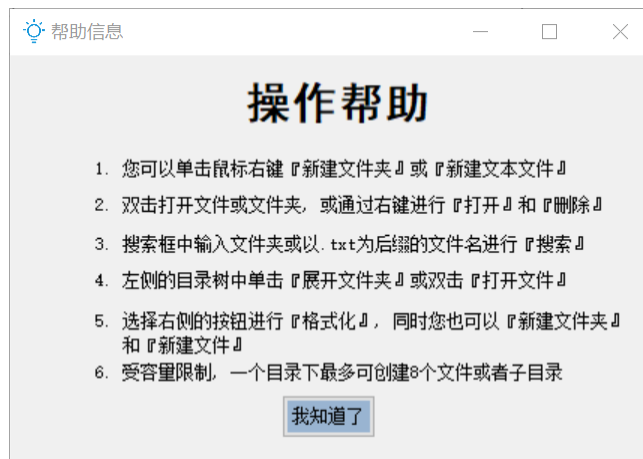
    for (int i = 0; i < MyDisk.blockNum; i++)
    {
        if (MyDisk.memory[i].IndexOf("\\r\\n") >= 0)
        {

```

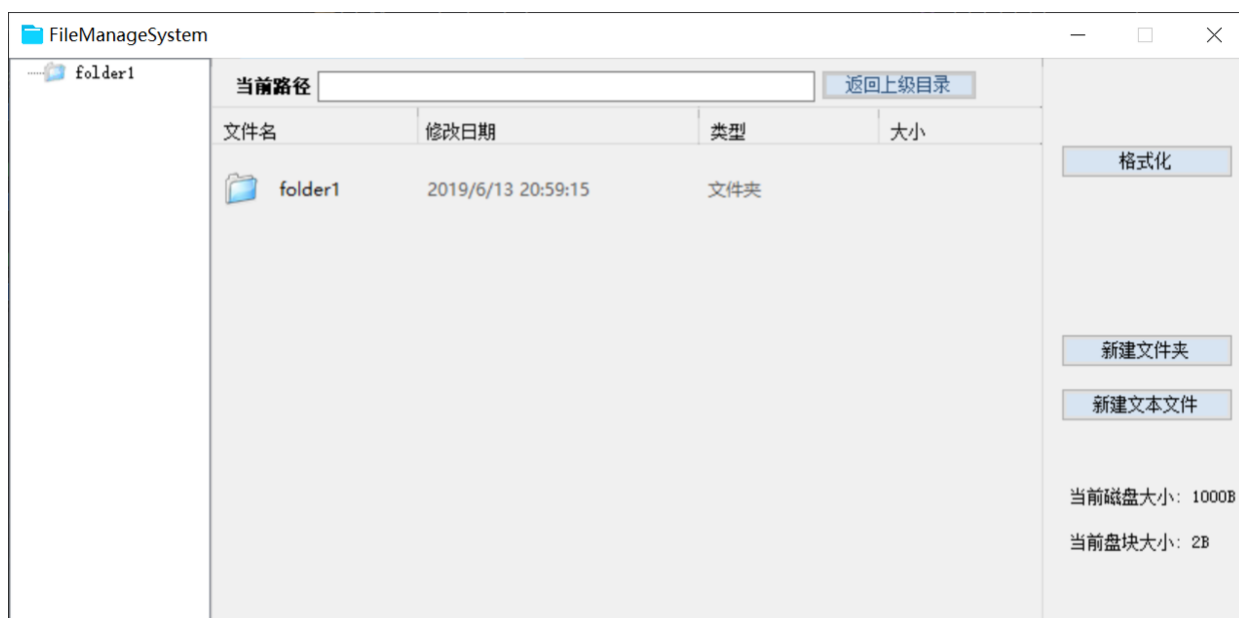
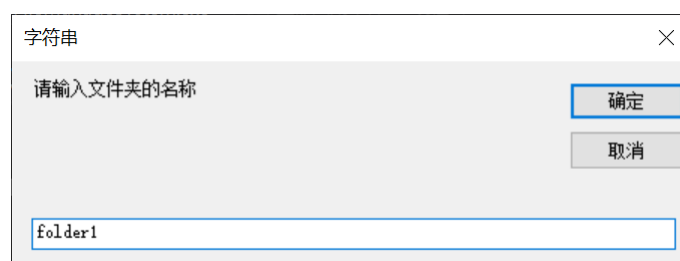
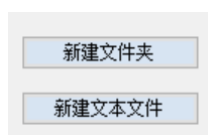
```
        writer.WriteLine(MyDisk.memory[i].Replace("\r\n", "()"));
    }
    else
    {
        if (MyDisk.memory[i].IndexOf('\r') >= 0)
        {
            writer.WriteLine(MyDisk.memory[i].Replace('\r', '('));
        }
        else if (MyDisk.memory[i].IndexOf('\n') >= 0)
        {
            writer.WriteLine(MyDisk.memory[i].Replace('\n', '('));
        }
        else if (MyDisk.memory[i] != "")
            writer.WriteLine(MyDisk.memory[i]);
        else
            writer.WriteLine("#");
    }
}
writer.Close();
}
```

功能实现截屏展示

帮助信息界面



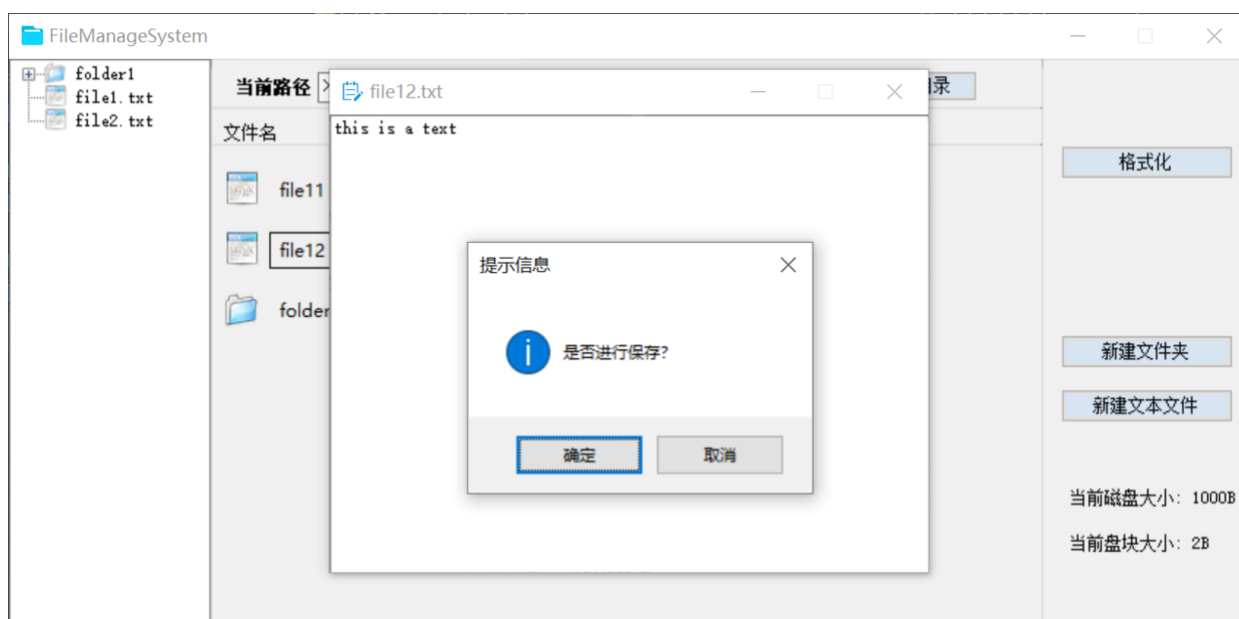
新建文件/文件夹




打开/删除文件项



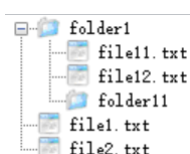
编辑文本文件



查看文件信息

	file	2019/6/13 21:15:52	文件	150B
---	------	--------------------	----	------

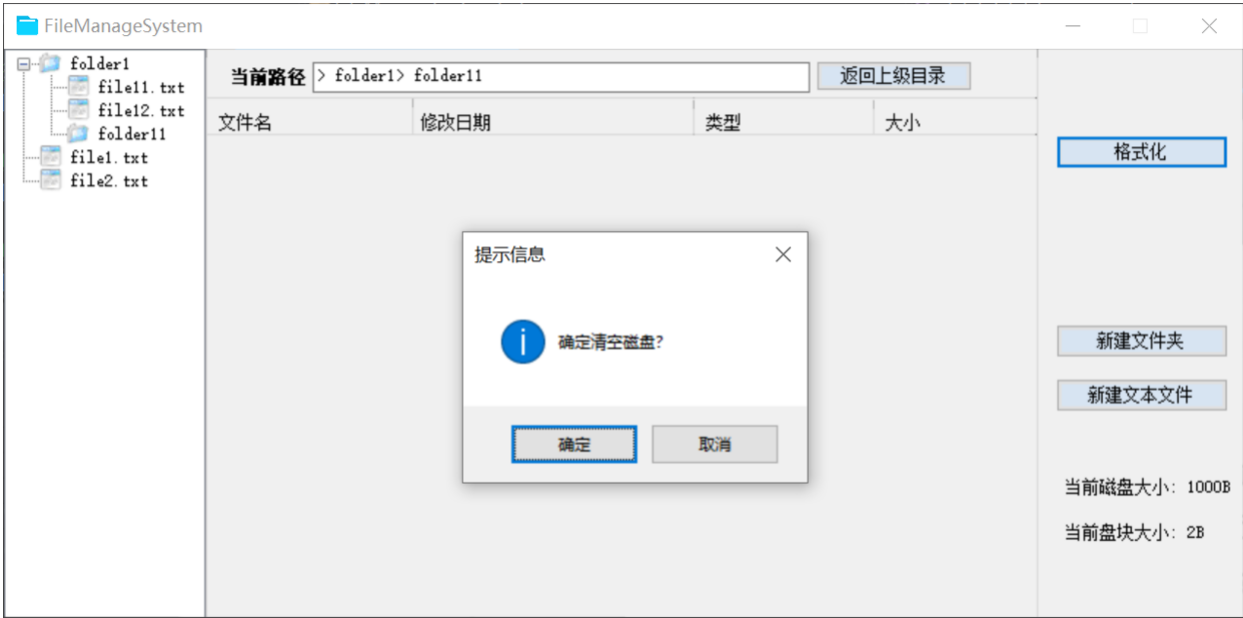
展开目录树



返回上级目录

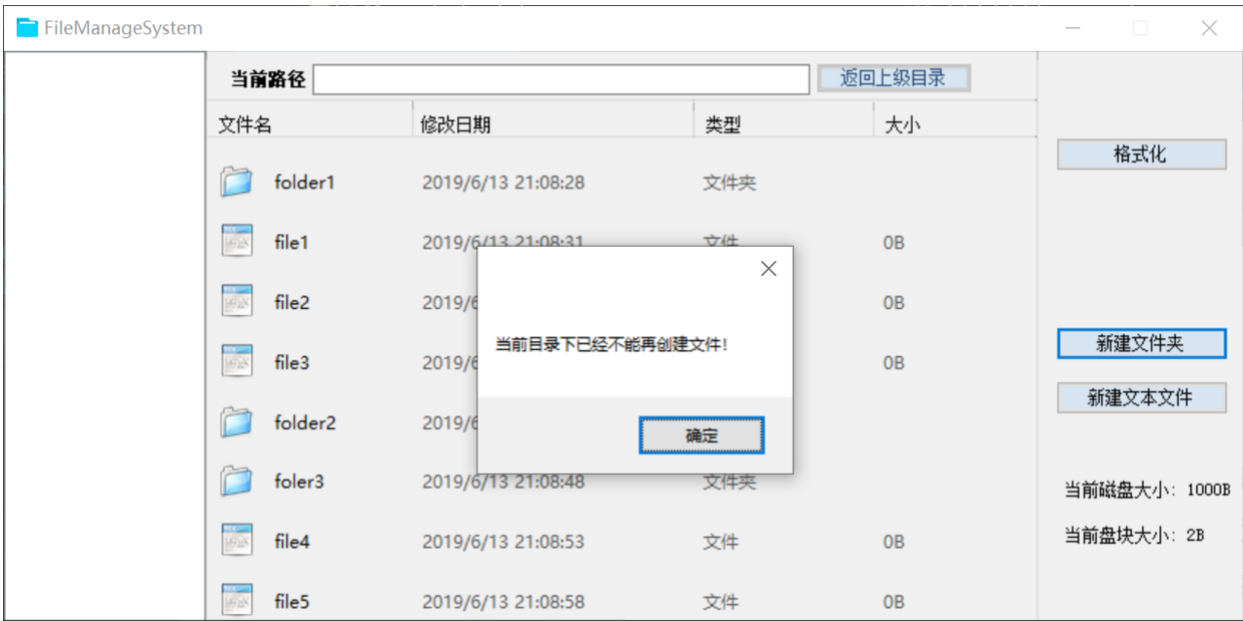


格式化磁盘



创建最大数量的文件项

!!!受展示限制, 本文件系统模拟器在一个目录下最多可创建8个子项目, 超过8个时会受到系统提醒(只是受展示方式限制, 物理和逻辑上的存储理论上都允许创建无限多的子项目).



退出系统重新进入后恢复目录

BitMapInfo.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

-1
-1
3
4
5
6
-2
-1
-1
-1
-1
-1
-1
-1
-1

Windows (CRLF)	第 1 行, 第 1 列	100%
----------------	--------------	------

CategoryInfo.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
root
folder1
1
2019/6/13 23:22:04
0
-1
#
root
file1
0
2019/6/13 23:22:08
0
0
#
root
file2
```

Windows (CRLF)	第 1 行, 第 1 列	100%
----------------	--------------	------

MyDiskInfo.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#
#
#  i
#  s
#  a
#  te
#  xt
#
#
#
#
#
#
#
#
#
#
```

Windows (CRLF)	第 1 行, 第 1 列	100%
----------------	--------------	------

作者

学号 1754060

姓名 张喆

指导老师 王冬青老师

上课时间 周三/周五 上午一二节

联系方式 email: doubleZ0108@gmail.com