# 操作系统第三次项目——文件系统模拟

## *谭梓煊(1853434) 2020.7.10*



# 0.项目概述 -- 完成情况

- [x] **基本文件系统** -- 按名存取，使用多级索引表
- [x] **磁盘格式化/保存到文件** -- 数据持久化
- [x] **创建/删除/重命名** -- 自动处理文件(夹)重名问题
- [x] **复制/粘贴/剪切** -- 内置剪贴板，递归复制文件(夹)
- [x] **前进/后退/返回上一级** -- 自动记录浏览历史，支持前进、后退
- [x] **按名查找** -- 可以搜索当前目录下的文件(夹)
- [x] **GUI文件浏览器** -- 所见即所得，实时显示系统可用空间和剩余空间

- [x] **文件编辑器** -- 可更换字体字号，显示当前编辑位置，自动提示保存

# 1.项目背景

## 1.1 相关背景

文件系统是操作系统中统一管理信息资源的一种软件，管理文件的存储、检索、更新，提供安全可靠的共享和保护手段，并且方便用户使用。对于操作系统而言，文件系统是必不可少，本次项目要求实现一个模拟的文件系统。

## 1.2 项目需求

- 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，以便下次可以将其恢复到内存中来。

## 1.3 项目目的

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

# 2.需求分析

### 具体要求：

文件存储空间管理可采取显式链接（如FAT）或者其他方法；

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。

### 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
- ……

# 3.项目设计

## 3.1 文件逻辑表示

建立`FCB`文件控制块类，每一个`FCB`对象表示一个文件或目录，`FCB`类中存储着文件的名称、类型、大小、数据分配表、父目录指针、兄弟指针等信息，我们通过`FCB`来管理文件；

所有的`FCB`通过双向长子-兄弟表示法组成一棵多叉树结构，多叉树的每个节点都是一个`FCB`对象。

## 3.2 文件存储管理

按照项目需求，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。

### 3.2.1 块存储单元

在内存上存储数据，建立了`Block`类作为存储数据的基本单位，每个Block可以存储**64**字节的数据，并且定义了设置和获得数据的接口。

### 3.2.2 空闲空间管理

本项目采用位图管理空闲空间，建立了`BitMap`类，专门负责空闲空间分配和回收；

位图内部开辟了10*1024个`Block`空间，作为系统的总存储空间；

空闲空间的分配按照循环遍历的原则，先找到的空位先进行分配。

### 3.2.3 文件分配表结构

本项目采用了多级索引表管理文件存储的物理结构；

建立`IndexTable`类作为主索引表，`PrimaryIndex`类作为一级索引表，`SecondaryIndex`类作为二级索引表，每一级索引表中都包含着索引数组`index[]`，利用该数组顺序记录`BitMap`分配的空闲空间索引，每一个`FCB`对象都包含一个索引表，用于记录该文件存储的blocks的索引。

每张索引表包括10个指向基本物理块的指针，1个一级间接索引表和1张二级间接索引表，每张一级间接索引表可指向128个基本物理块，每张二级间接索引表可指向128个一级间接索引表。

经过多级索引表的表示，该文件系统支持的最大文件为`(10+128+128*128)*64B=1032KB`；

## 3.3 目录表示

目录与普通文件一致，采用一个`FCB`结构表示。与普通文件不同的是，目录不包含文件分配表；

由于本项目采用树形逻辑表示，因此可以很方便地创建文件和子文件夹；

## 3.4 内存数据持久化

利用C#的序列化功能，将`BitMap`，`Directory`，`FCB`，`IndexTable`，`Block`类均声明为`Serializable`；

在关闭程序时，将该文件管理系统每个类的数据序列化后存储到硬盘；

启动程序时通过反序列化将数据从硬盘读回内存中，如果失败则会创建新的文件系统；

# 4.关键代码

## 4.1 `FCB`类

```
namespace TinyFileSystem
{
    // File Control Block
    // 文件控制块，包含文件基本信息，用于表示文件或目录
    // 不包含块分配表，放在File中
    // 使用双向长子-兄弟表示法表示树形结构，
    [Serializable]
```

```
public class FCB
{
    public enum FileType { directory, regular };
    public static int fileCounter = 0; // 文件总数

    public int fileNo; // 文件号
    public string name; // 文件名
    public FileType type; // 文件类型
    public int size; // 文件大小(字节)
    public DateTime createTime; // 创建时间
    public IndexTable blockTable; // 块分配图

    public FCB child = null, parent = null; // 指向父目录和子目录的指针
    public FCB next = null, pre = null; // 指向左右兄弟的指针

    public string Path {
        get
        {
            string path = this.name;
            FCB fcb = this.parent;
            while (fcb != null)
            {
                path = fcb.name + (fcb.parent == null ? "://" : "/") + path;
                fcb = fcb.parent;
            }
            return path;
        }
    }

    public FCB(string fileName, FileType fileType)
    {
        this.fileNo = fileCounter++;
        this.name = fileName;
        this.type = fileType;
        this.size = 0;
        this.createTime = DateTime.Now;

        if (fileType == FileType.regular)
        {
            this.blockTable = new IndexTable();
        }
    }

    public void AppendChild(FCB child)
    {
        if (this.child == null)
        {
            this.child = child;
            child.parent = this;
        }
        else
        {
            FCB temp = this.child;
            while (temp.next != null)
            {
                temp = temp.next;
            }
            temp.next = child;
```

```
                child.pre = temp;
                child.parent = this;
            }
        }

        // 从链表中删除自己
        public void Remove()
        {
            if (parent != null && parent.child == this)
            {
                parent.child = next;
            }
            else if (pre != null)
            {
                pre.next = next;
            }
        }
    }
}
```

## 4.2 Block类

```
namespace TinyFileSystem
{
    // 数据块，长度64个字节
    [Serializable]
    public class Block
    {
        public static readonly int blockSize = 64;
        private readonly char[] data = new char[blockSize];
        private int size = 0;

        public Block()
        {
        }

        public void SetData(string data)
        {
            size = (data.Length > blockSize) ? blockSize : data.Length;
            for(int i = 0; i < size; i++)
            {
                this.data[i] = data[i];
            }
        }

        public string GetData()
        {
            return new string(data);
        }
    }
}
```

## 4.3 BitMap类

```
namespace TinyFileSystem
```

```
{
    // 位图，管理数据块
    [Serializable]
    public class BitMap
    {
        public int size = 0; // 当前使用容量
        public static int capacity = 10 * 1024; // 位图最大容量
        private readonly bool[] bitMap = new bool[capacity];
        private readonly Block[] blocks = new Block[capacity];

        public BitMap()
        {
            for (int i = 0; i < capacity; i++)
            {
                bitMap[i] = true;
            }
        }

        //Get a block's data
        public string GetBlock(int i)
        {
            return blocks[i].GetData();
        }

        // 申请一个新块，并设置数据，返回块号
        // 找不到新块时返回-1
        public int NewBlock(string data)
        {
            size %= capacity;
            int tempPointer = size;
            while (true)
            {
                if (bitMap[tempPointer])
                {
                    blocks[tempPointer] = new Block();
                    blocks[tempPointer].SetData(data);
                    size = tempPointer + 1;
                    return tempPointer;
                }
                else
                {
                    tempPointer += 1;
                    tempPointer %= capacity;
                }
                if (tempPointer == size)
                {
                    break;
                }
            }
            return -1;
        }

        public void FreeBlock(int index)
        {
            bitMap[index] = true;
        }

        public void FreeBlock(List<int> indexs)
```

```
        {
            foreach(int i in indexs)
            {
                bitMap[i] = true;
            }
        }

        // 将所有内容创建成一个数据块列表，返回这个列表的blockTable
        public IndexTable CreateBlockTable(string data)
        {
            IndexTable table = new IndexTable();
            while (data.Count() > Block.blockSize)
            {
                table.AddIndex(NewBlock(data.Substring(0, Block.blockSize)));
                data = data.Remove(0, Block.blockSize);
            }
            table.AddIndex(NewBlock(data));

            return table;
        }
    }
}
```

## 4.4 IndexTable类

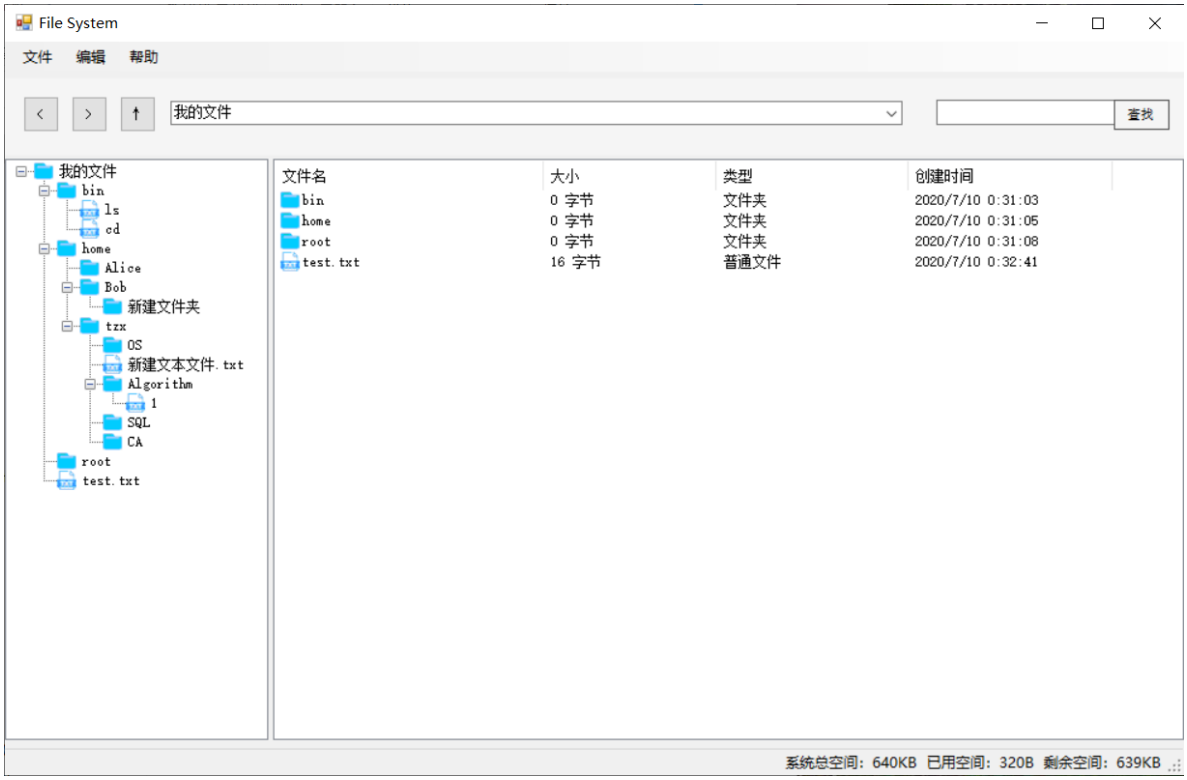
```
namespace TinyFileSystem
{
    // 多级索引结构
    // 最多支持10+128+128*128个数据块
    [Serializable]
    public class IndexTable
    {
        public static int capacity = 10;
        private readonly int[] index = new int[capacity];
        private int size = 0;
        private PrimaryIndex pIndex;
        private SecondaryIndex sIndex;

        public IndexTable()
        {
        }

        public bool Isfull()
        {
            return size >= capacity;
        }

        public bool AddIndex(int block)
        {
            if (!Isfull())
            {
                index[size] = block;
                size++;
                if(size == capacity)
                {
                    pIndex = new PrimaryIndex();
```

```
            }
        }
        else if (!pIndex.Isfull())
        {
            pIndex.AddIndex(block);
            if (pIndex.Isfull())
            {
                sIndex = new SecondaryIndex();
            }
        }
        else if(!sIndex.Isfull())
        {
            sIndex.AddIndex(block);
        }
        else
        {
            return false;
        }
        return true;
    }

    public List<int> GetBlockList()
    {
        List<int> blockList = new List<int>();

        for(int i = 0; i < size; i++)
        {
            blockList.Add(index[i]);
        }
        if(size == 100)
        {
            for(int j = 0;j < pIndex.size; j++)
            {
                blockList.Add(pIndex.index[j]);
            }
        }
        if (pIndex != null && pIndex.Isfull())
        {
            foreach (PrimaryIndex pIndex in sIndex.pIndex)
            {
                for(int k = 0; k < pIndex.size; k++)
                {
                    blockList.Add(pIndex.index[k]);
                }
            }
        }

        return blockList;
    }
}

[Serializable]
public class PrimaryIndex
{
    public static int capacity = 128;
    public int[] index = new int[capacity];
    public int size = 0;
```

```
        public PrimaryIndex()
        {
        }

        public void AddIndex(int data)
        {
            index[size] = data;
            size++;
        }

        public bool Isfull()
        {
            return size >= capacity;
        }
    }

    [Serializable]
    public class SecondaryIndex
    {
        public static int capacity = 128;
        public List<PrimaryIndex> pIndex = new List<PrimaryIndex>{ new
PrimaryIndex() };
        public int size = 0;

        public SecondaryIndex()
        {
        }

        public bool Isfull()
        {
            return size >= capacity;
        }

        public void AddIndex(int data)
        {
            PrimaryIndex temp = pIndex[size];
            if (temp.Isfull())
            {
                pIndex.Add(new PrimaryIndex());
                size++;
                temp = pIndex[size];
            }
            temp.AddIndex(data);
        }
    }
}
```

# 5.程序演示

## 5.1 界面说明




最上方为菜单栏功能区，功能如下

- 文件
  - 新建
    - 文件 -- 在当前目录下创建一个新文件
    - 文件夹 -- 在当前目录下创建一个新文件夹
  - 删除 -- 删除选中的文件/文件夹
  - 打开 -- 编辑选中的文件/进入选中的文件夹
  - 格式化 -- 格式化文件系统
  - 保存 -- 将文件系统同步写入真实文件中
- 编辑 -- 和现有文件系统功能类似
  - 剪切
  - 复制
  - 粘贴
- 帮助
  - 关于 -- 显示"关于"窗口
  - 使用帮助 -- 显示"使用帮助"窗口

上方左侧的三个按钮分别为`后退`，`前进`，`返回上一层目录`，在文件浏览过程中，系统会自动文件记录浏览历史，可以通过后退前进键快速跳转；

上方中间的路径栏会显示当前目录的路径，以`我的文件://`开始；

上方右侧为文件搜索栏，可以按名搜索当前目录下的文件和文件夹；

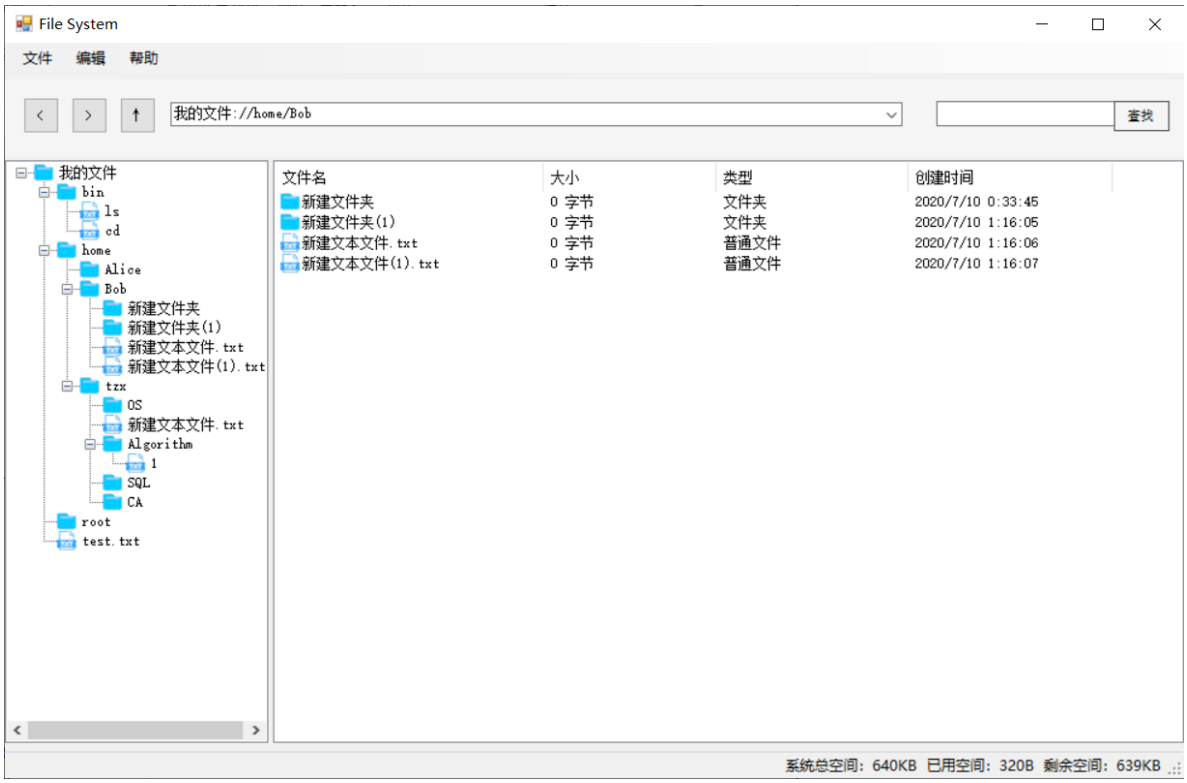
左侧为文件系统树状结构视图，单击文件夹可以进入目录，双击普通文件可以打开编辑窗口；

右侧为当前目录的详细视图，显示当前文件夹中的文件信息列表，双击条目可以执行切换路径、编辑文件操作；右键点击会弹出便捷菜单，与顶部菜单栏类似；

底部为文件系统状态栏，显示文件系统可用空间与剩余空间等信息；

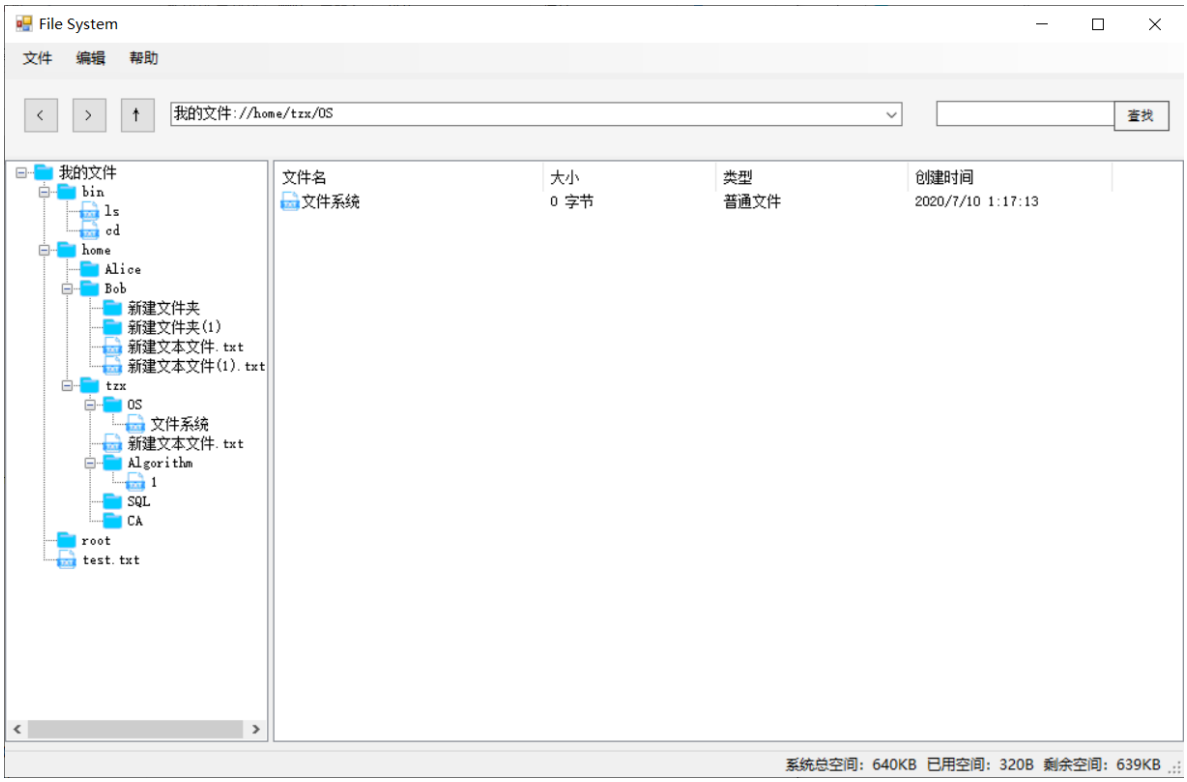
## 5.2 新建文件与文件夹

该文件系统具有重名检查功能，会自动添加后缀序号避免重名；




## 5.3 进入子文件夹

双击文件夹或选中后选择“打开”均可进入子文件夹，此时地址栏地址会变化，当前目录文件视图也会刷新



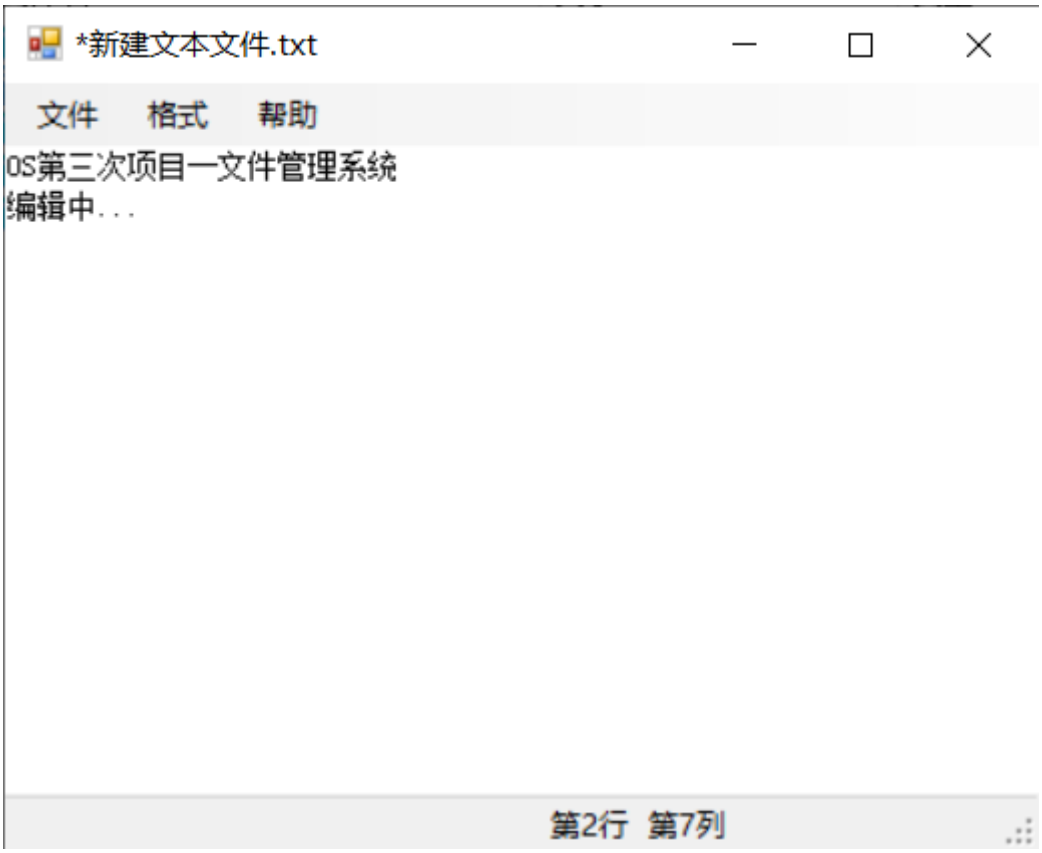

## 5.4 文本文件的操作

双击文件或选中后选择“打开”均可打开文本编辑器，可以对普通文件以文本方式进行编辑；

通过上方菜单栏可以修改字体字号，保存修改，退出等等；

编辑时，底部状态栏会实时反馈当前编辑位置；

关闭编辑器时，若文件尚未保存，系统会提示用户是否需要保存，并更新文件大小信息；




## 5.5 格式化

点击格式化可以格式化当前文件系统；

## 5.6 保存

退出文件系统时会提示是否保存系统信息，点击确定后会将当前的数据序列化后存储到硬盘中，在菜单栏中可以读取本地保存的文件系统数据；

# 6.附录

## 6.1 文件说明

```
|  README.md
|  项目文档.pdf
|
├─exe -- 可执行文件
|      bitMap.dat
|      catalogTable.dat
|      catalogTree.dat
|      TinyFileSystem.exe
|      TinyFileSystem.exe.config
|      TinyFileSystem.pdb
|
├─img -- 文档图片
|      1.png
|      2.png
|      3.png
|      4.png
|
```

```
└src -- 项目源代码
    |  FileSystem.sln
    |
    └Filesystem
        |  App.config
        |  Class1.cs
        |  Editor.cs
        |  Editor.Designer.cs
        |  Editor.resx
        |  FileSystem.csproj
        |  MainWindows.cs
        |  MainWindows.Designer.cs
        |  MainWindows.resx
        |  Program.cs
        |
        ├─Kernel -- 文件系统核心部分
        |      BitMap.cs
        |      Block.cs
        |      FCB.cs
        |      FCBTable.cs
        |      IndexTable.cs
        |
        ├─Properties
        |      AssemblyInfo.cs
        |      Resources.Designer.cs
        |      Resources.resx
        |      Settings.Designer.cs
        |      Settings.settings
        |
        └─Resource -- 项目资源文件
                file18.png
                folder18.png
```

## 6.2 开发环境

- 操作系统：windows 10 64-bit
- 开发语言：C#.NET
- 开发环境：Visual Studio 2019