# 文件管理系统: 项目文档

**Operating System Project**

**2352471 刘震**

# 1. 项目概览

## 1.1 项目介绍

本项目是一个基于C#和Windows Presentation Foundation (WPF)开发的桌面应用程序，旨在模拟一个简单的操作系统的文件管理系统

它通过图形化用户界面(GUI)直观地展示了文件系统的内部工作原理，其代码核心围绕操作系统课程中的文件系统的理论知识构建

### 1.1.1 操作系统原理模拟

项目重点模拟了以下三个核心的操作系统概念：

- **文件控制块 (File Control Block, FCB)**: 系统中的每个文件和目录都由一个FCB对象表示。FCB是文件系统用于管理文件的最重要的数据结构，它包含了文件的所有元数据，如文件名、大小、创建/修改时间、物理位置等。
- **位图 (Bitmap) 磁盘分配算法**: 为了管理存储空间，项目实现了一个位图。位图中的每一位(bit)对应一个物理磁盘块，通过位的状态（0或1）来跟踪该块是空闲还是已被占用。这种方法对于查找和分配空闲空间非常高效。
- **树形目录结构**: 系统实现了一个层次化的文件结构，允许目录的无限嵌套，这与现代主流操作系统（如Windows和Linux）的目录组织方式一致。

### 1.1.2 开发工具与技术

- **.NET框架**: .NET 6.0
- **开发语言**: C# 10
- **用户界面**: Windows Presentation Foundation (WPF)

## 1.2 文件结构

```
FileManagerSystem/
|
├── Report.pdf        # 项目文档
├── FileManager.exe   # 项目可执行文件
├── README.md         # 项目运行说明
├── images/           # 项目运行图片文件
├── src/              # 项目完整代码
|   ├── FileManagerSystem.csproj
|   ├── App.xaml
|   ├── App.xaml.cs
|
|   ├── Models/
|   |   ├── FCB.cs
```

```
|   |   └── BitMap.cs
|
|   ├── Services/
|   |   └── FileSystemService.cs
|
|   ├── ViewModels/
|   |   ├── MainViewModel.cs
|   |   └── FileItemViewModel.cs
|
|   ├── Views/
|   |   ├── MainWindow.xaml
|   |   ├── MainWindow.xaml.cs
|   |   └── Dialogs/
|   |       ├── InputDialog.xaml
|   |       ├── InputDialog.xaml.cs
|   |       ├── TextEditorDialog.xaml
|   |       ├── TextEditorDialog.xaml.cs
|   |       ├── PropertiesDialog.xaml
|   |       └── PropertiesDialog.xaml.cs
|
|   ├── Commands/
|   |   └── RelayCommand.cs
|
|   ├── Converters/
|   |   └── BoolToTypeConverter.cs
|
```

## 1.3 运行说明

本项目需要.NET 6.0 SDK环境。可以通过以下步骤在命令行中编译和运行程序。

1. **克隆或下载项目到本地。**
2. **打开终端（如PowerShell或CMD），并导航到项目根目录。**

```
cd src
```

3. **还原.NET依赖项。**

```
dotnet restore ./project3.sln
```

4. **编译项目。**

```
dotnet build ./project3.sln
```

5. **运行应用程序。**

```
dotnet run
```

# 2. 类的介绍

## 2.1 `Models.FCB`

文件控制块类，操作系统中FCB的完整实现。它定义了文件或目录所必需的基础属性，包括文件名、大小、创建时间、修改时间、访问时间、起始物理块号以及所占用的所有物理块列表。同时包含了文本内容存储功能（TextContent属性），用于在内存中直接存储`.txt`文件的文本内容，以及文件类型判断和描述的相关方法。

## 2.2 `Models.BitMap`

位图算法的实现类。内部使用一个`BitArray`来模拟物理磁盘的块分配表。它提供了分配和释放单个或多个磁盘块的核心方法，并能计算当前磁盘的使用率。

## 2.3 `Services.FileSystemService`

文件系统的核心服务类，封装了所有与文件操作相关的业务逻辑。它管理着全局的FCB表和磁盘位图实例。所有对文件的创建、删除、重命名和内容修改等操作都通过此类完成。

## 2.4 `ViewModels.MainViewModel`

主窗口`MainWindow`的视图模型。它遵循MVVM设计模式，负责处理主窗口的UI逻辑，如文件列表的展示、导航历史记录、UI状态更新以及用户命令的响应。

## 2.5 `ViewModels.FileItemViewModel`

文件列表中单个项目（文件或文件夹）的视图模型。它包装了`FCB`对象，并为UI提供了格式化的数据，如将文件大小（字节）转换为KB字符串。

## 2.6 `Views.MainWindow`

应用程序的主窗口，定义了文件管理器的整体布局，包括菜单栏、工具栏、地址栏、文件列表视图和状态栏。

## 2.7 `Views.Dialogs.TextEditorDialog`

一个独立的窗口，用作简单的文本编辑器。当用户打开`.txt`文件时，此窗口会显示，并提供基本的文本编辑和保存功能。

## 2.8 `Views.Dialogs.InputDialog`

一个通用的输入对话框，用于在创建新文件/文件夹或重命名现有项时获取用户输入的名称。

## 2.9 `Views.Dialogs.PropertiesDialog`

用于显示文件或文件夹详细属性的对话框。它会展示选中项的FCB中存储的所有元数据。

### 2.10 `Commands.RelayCommand`

MVVM模式中`ICommand`接口的一个通用实现。它使得在ViewModel中定义的普通方法可以轻松地绑定到View中的控件事件上（如按钮点击）。

### 2.11 `Converters.BoolToTypeConverter`

一个WPF值转换器。它将`FCB`的`IsDirectory`布尔属性转换为对应的文件或文件夹图标（用Emoji字符表示），实现了UI上的视觉区分。

## 3. 核心代码分析

### 3.1 文件控制块 (FCB) 设计

FCB是文件系统的基石。`FCB`类定义了所有文件和目录共享的属性，并包含了文本内容存储功能，支持完整的文件管理操作。

#### 3.1.1 `FCB.cs` 核心定义

```
// Models/FCB.cs

public class FCB
{
    // 文件名
    public string FileName { get; set; }
    // 完整路径
    public string FullPath { get; set; }
    // 父目录路径
    public string ParentPath { get; set; }
    // 是否为目录
    public bool IsDirectory { get; set; }
    // 文件大小（字节）
    public long Size { get; set; }
    // 创建、修改、访问时间戳
    public DateTime CreatedTime { get; set; }
    public DateTime ModifiedTime { get; set; }
    public DateTime AccessedTime { get; set; }
    // 物理存储信息
    public int StartBlock { get; set; }
    public List<int> AllocatedBlocks { get; set; }
    // 目录内容
    public List<FCB> Children { get; private set; }
    // 父目录引用
    public FCB? Parent { get; set; }

    public FCB()
    {
        // ... 初始化 ...
    }
}
```

**逻辑说明**:

- `FullPath`作为系统中每个文件或目录的唯一标识符。
- `AllocatedBlocks`列表存储了该文件占用的所有物理磁盘块的索引，这对于非连续分配至关重要。
- `Children`和`Parent`属性共同构成了内存中的目录树结构。

### 3.1.2 FCB扩展功能

FCB类除了基础的文件属性外，还包含了以下扩展功能：

```
// 文本内容存储（用于.txt文件）
public string TextContent { get; set; } = string.Empty;
public string MimeType { get; set; } = "text/plain";
public string Encoding { get; set; } = "UTF-8";

// 判断此文件是否为可编辑的文本文档
public bool IsTextEditable()
{
    if (IsDirectory) return false;
    var extension = System.IO.Path.GetExtension(FileName).ToLowerInvariant();
    return extension == ".txt";
}

// 获取文件类型描述
public string GetFileTypeDescription()
{
    if (IsDirectory) return "文件夹";
    var extension = System.IO.Path.GetExtension(FileName).ToLowerInvariant();
    return extension switch
    {
        ".txt" => "文本文档",
        _ => "文件"
    };
}
```

**逻辑说明**:

- FCB作为一个完整的文件控制块，包含了`TextContent`属性用于存储文本内容。在我们的模拟系统中，文件内容不直接写入模拟的"磁盘块"，而是存储在此属性中。这极大地简化了文件读写操作的实现，使我们能专注于文件元数据和磁盘空间管理的模拟。

## 3.2 位图 (Bitmap) 磁盘分配算法

`BitMap`类负责模拟物理磁盘的空间管理。它提供分配和释放块的核心功能。

```
// Models/BitMap.cs

public class BitMap
{
    private readonly BitArray _bitArray;
    public int TotalBlocks { get; } = 1024;
    private const int BLOCK_SIZE = 1024; // 1 KB
```

```
    // ...

    // 分配指定数量的非连续磁盘块
    public List<int> AllocateBlocksNonContiguous(int blockCount)
    {
        if (FreeBlocks < blockCount)
        {
            return new List<int>(); // 空间不足
        }

        var allocated = new List<int>();
        for (int i = 0; i < TotalBlocks && allocated.Count < blockCount; i++)
        {
            if (!_bitArray[i]) // 如果块是空闲的 (false)
            {
                _bitArray[i] = true; // 标记为已占用 (true)
                allocated.Add(i);
            }
        }
        return allocated;
    }

    // 释放一组磁盘块
    public void DeallocateBlocks(List<int> blocks)
    {
        foreach (var blockIndex in blocks)
        {
            if (blockIndex >= 0 && blockIndex < TotalBlocks)
            {
                _bitArray[blockIndex] = false; // 标记为空闲
            }
        }
    }
}
```

**逻辑说明**:

- `_bitArray`是核心数据结构，其索引`i`代表第`i`个磁盘块。`_bitArray[i] = true`表示已占用，`false`表示空闲。
- `AllocateBlocksNonContiguous`方法从头开始扫描位图，寻找`blockCount`个状态为`false`的位，并将它们的状态设置为`true`。它返回所有成功分配的块的索引列表。这是对非连续分配策略的直接模拟。
- `DeallocateBlocks`方法接收一个块索引列表，并将它们在位图中对应的位设置回`false`，从而"释放"这些块。

## 3.3 文件系统核心操作

`FileSystemService`是所有操作的中心。以`CreateFile`为例，可以清晰地看到FCB和位图是如何协同工作的。

```
// Services/FileSystemService.cs

```

```
public class FileSystemService
{
    private readonly Dictionary<string, FCB> _fcbTable;
    private readonly BitMap _bitMap;
    // ...

    public bool CreateFile(string fileName, string parentPath, string content = "")
    {
        string fullPath = $"{parentPath}\\{fileName}".Replace("\\\\", "\\");
        // 1. 检查文件是否已存在
        if (string.IsNullOrEmpty(fileName) || _fcbTable.ContainsKey(fullPath))
            return false;

        // 2. 找到父目录FCB
        var parentFCB = _fcbTable.ContainsKey(parentPath) ? _fcbTable[parentPath] :
null;
        if (parentFCB == null || !parentFCB.IsDirectory)
            return false;

        // 3. 创建新的文件FCB实例
        var newFile = new FCB(fileName, false, parentPath);
        newFile.Size = content.Length;
        if (fileName.EndsWith(".txt", StringComparison.OrdinalIgnoreCase))
        {
            newFile.TextContent = content;
        }

        // 4. 根据内容大小计算所需磁盘块数
        int blocksNeeded = Math.Max(1, (int)Math.Ceiling((double)content.Length /
BLOCK_SIZE));

        // 5. 使用位图分配磁盘块
        var blocks = _bitMap.AllocateBlocksNonContiguous(blocksNeeded);
        if (blocks.Count == 0 && blocksNeeded > 0)
        {
            return false; // 磁盘空间不足
        }

        // 6. 更新FCB的物理存储信息
        newFile.AllocatedBlocks = blocks;
        newFile.StartBlock = blocks.FirstOrDefault(-1);

        // 7. 将新文件添加到文件系统记录中
        parentFCB.AddChild(newFile);
        _fcbTable[newFile.FullPath] = newFile;

        return true;
    }
}
```

**逻辑说明**:

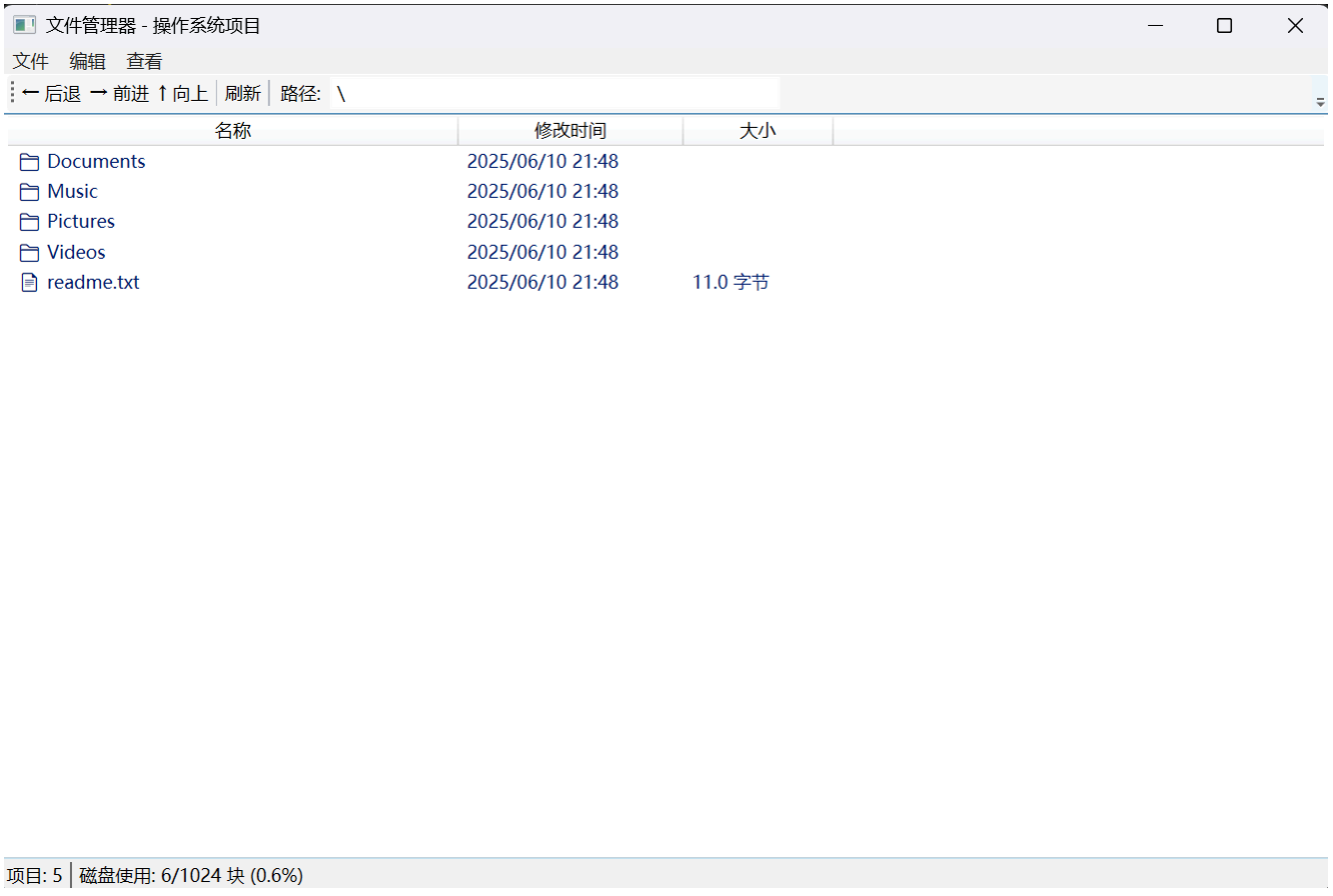
1. **验证**: 首先进行有效性检查，确保文件名有效且不存在同名文件。

2. **定位**: 查找父目录的FCB。所有文件都必须存在于一个目录中。
3. **创建FCB**: 实例化一个新的`FCB`对象，并设置其基本属性。
4. **计算空间**: 根据文件内容（或默认为空）计算需要多少个磁盘块。即使文件为空，也至少分配一个块来存储其元数据。
5. **分配空间**: 调用`_bitMap`服务来获取所需数量的空闲块。
6. **更新元数据**: 将分配到的块信息记录回新创建的FCB中。
7. **注册文件**: 最后，将新的FCB添加到父目录的子项列表和全局的`_fcbTable`哈希表中，完成文件的创建过程。`_fcbTable`使用完整路径作为键，提供了对系统中任何文件或目录的O(1)时间复杂度的快速访问。

# 4. 运行展示

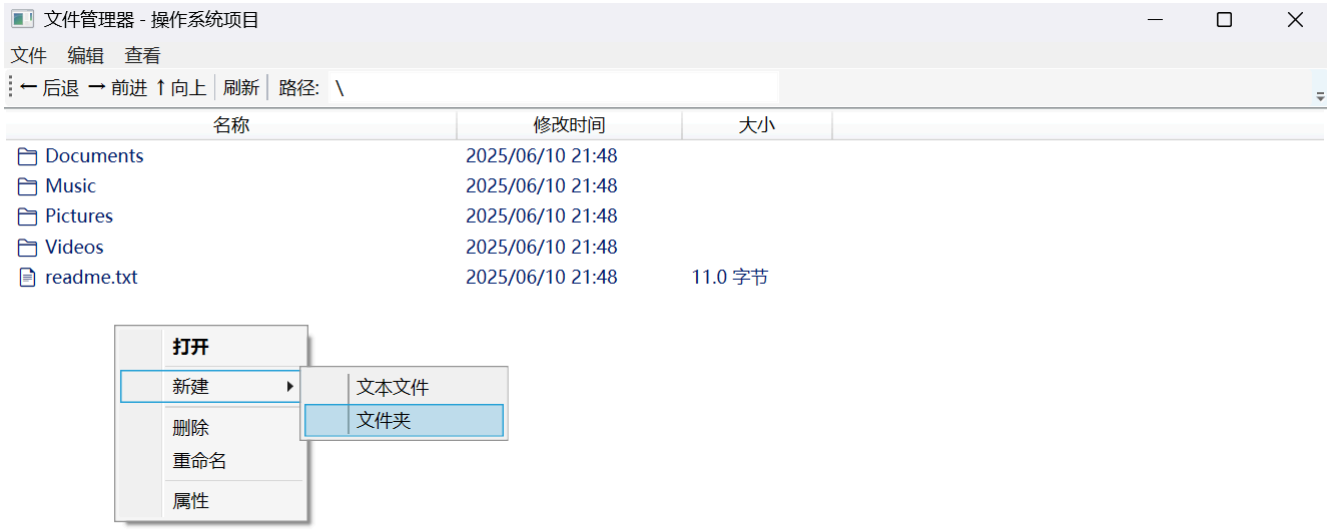
## 4.1 主界面展示

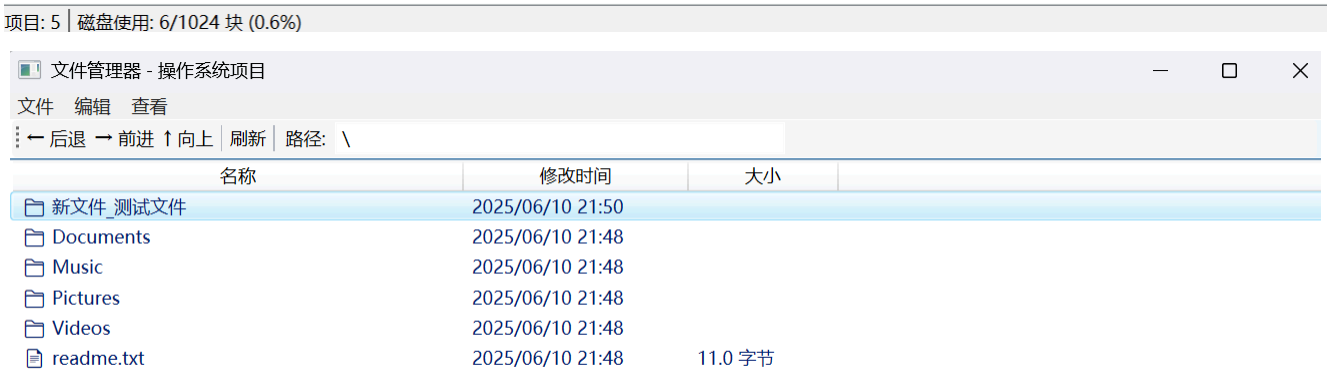
可以清晰看见文件的名称、修改时间、大小等属性，同时可以自由的在目录中进行后退、前进等操作



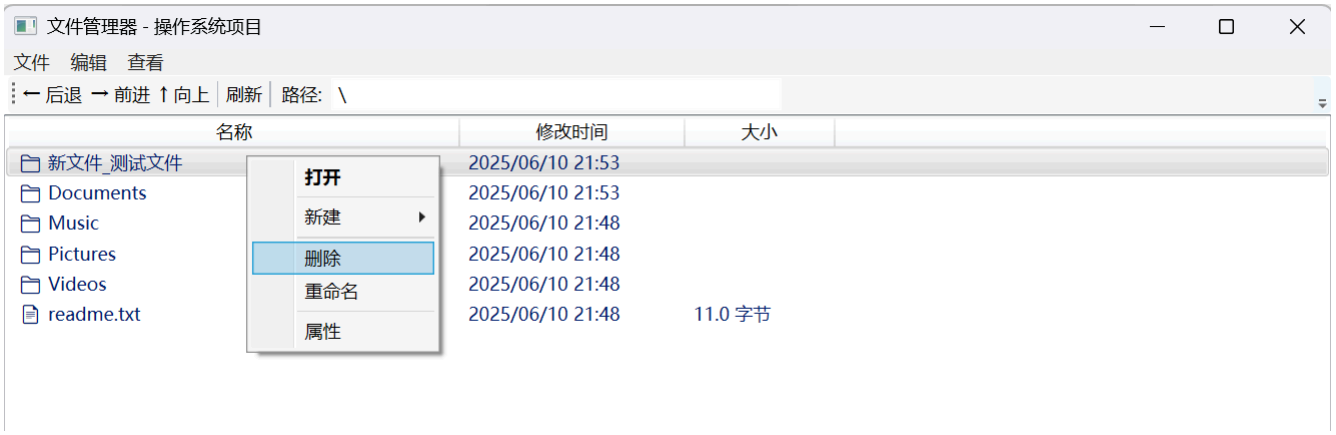

# 4.2 文件基本操作

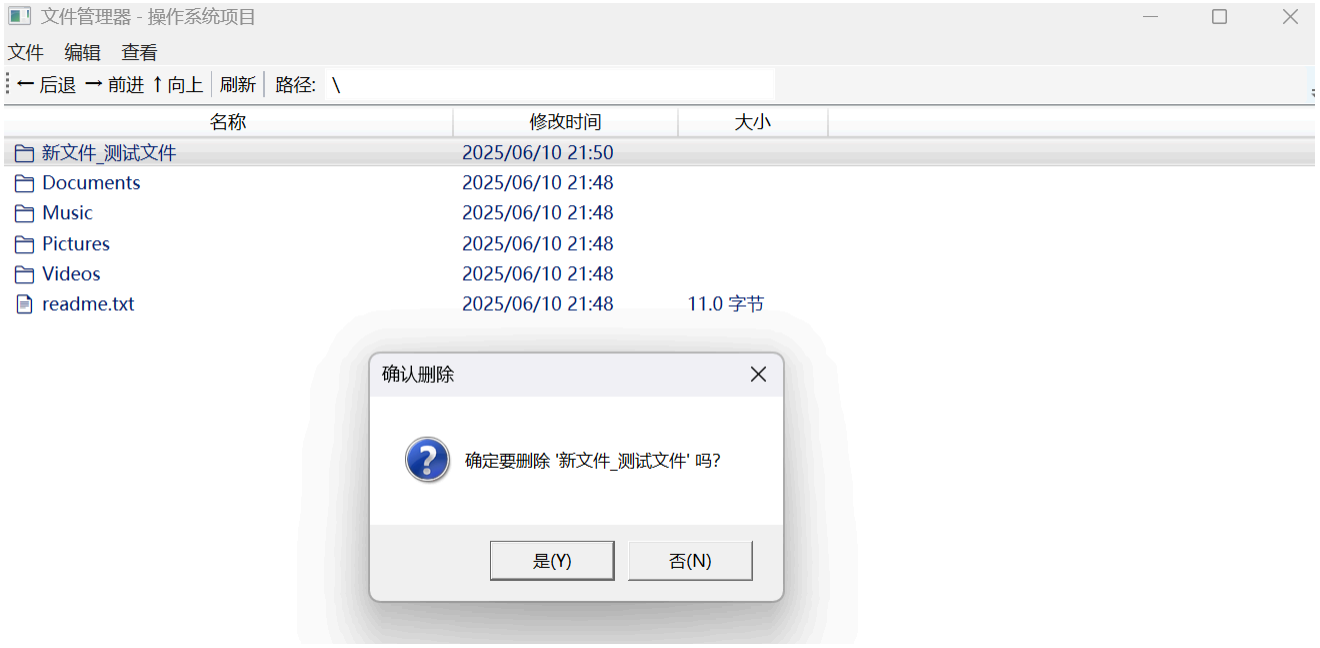
文件的新建：


文件管理器 - 操作系统项目
文件 编辑 查看
后退 前进 向上 刷新 路径: \
名称 修改时间 大小
Documents 2025/06/10 21:48
Music 2025/06/10 21:48
Pictures 2025/06/10 21:48
Videos 2025/06/10 21:48
readme.txt 2025/06/10 21:48 11.0 字节
打开
新建
删除
重命名
属性
文本文件
文件夹

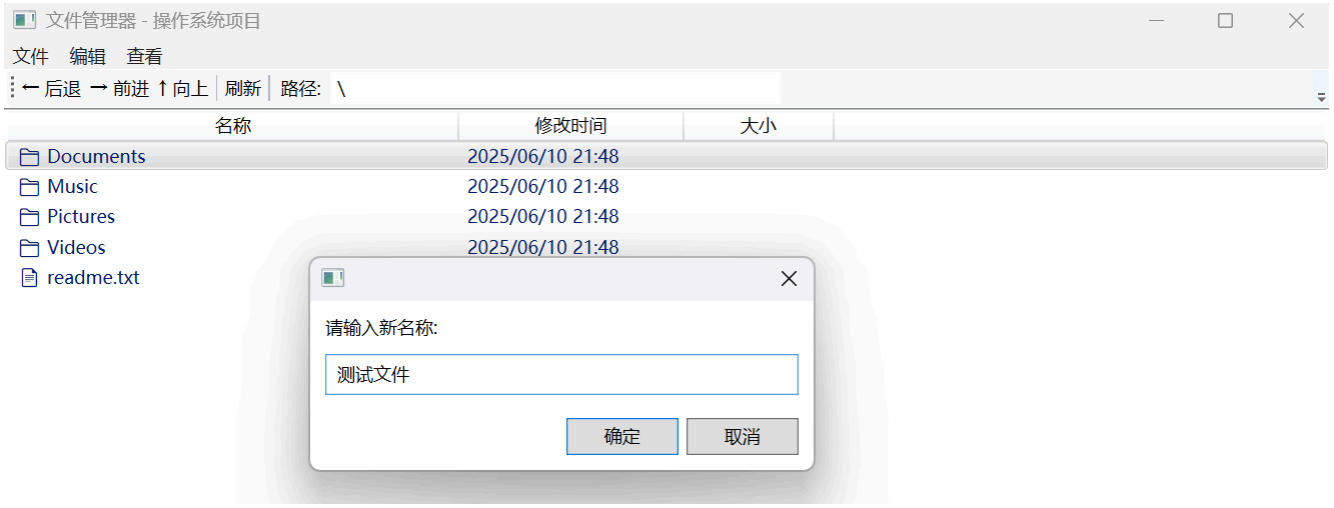

项目: 5 | 磁盘使用: 6/1024 块 (0.6%)
文件管理器 - 操作系统项目
文件 编辑 查看
后退 前进 向上 刷新 路径: \
名称 修改时间 大小
新文件_测试文件 2025/06/10 21:50
Documents 2025/06/10 21:48
Music 2025/06/10 21:48
Pictures 2025/06/10 21:48
Videos 2025/06/10 21:48
readme.txt 2025/06/10 21:48 11.0 字节

文件的删除：


文件管理器 - 操作系统项目
文件 编辑 查看
后退 前进 向上 刷新 路径: \
名称 修改时间 大小
新文件_测试文件 2025/06/10 21:53
Documents 2025/06/10 21:53
Music 2025/06/10 21:48
Pictures 2025/06/10 21:48
Videos 2025/06/10 21:48
readme.txt 2025/06/10 21:48 11.0 字节
打开
新建
删除
重命名
属性



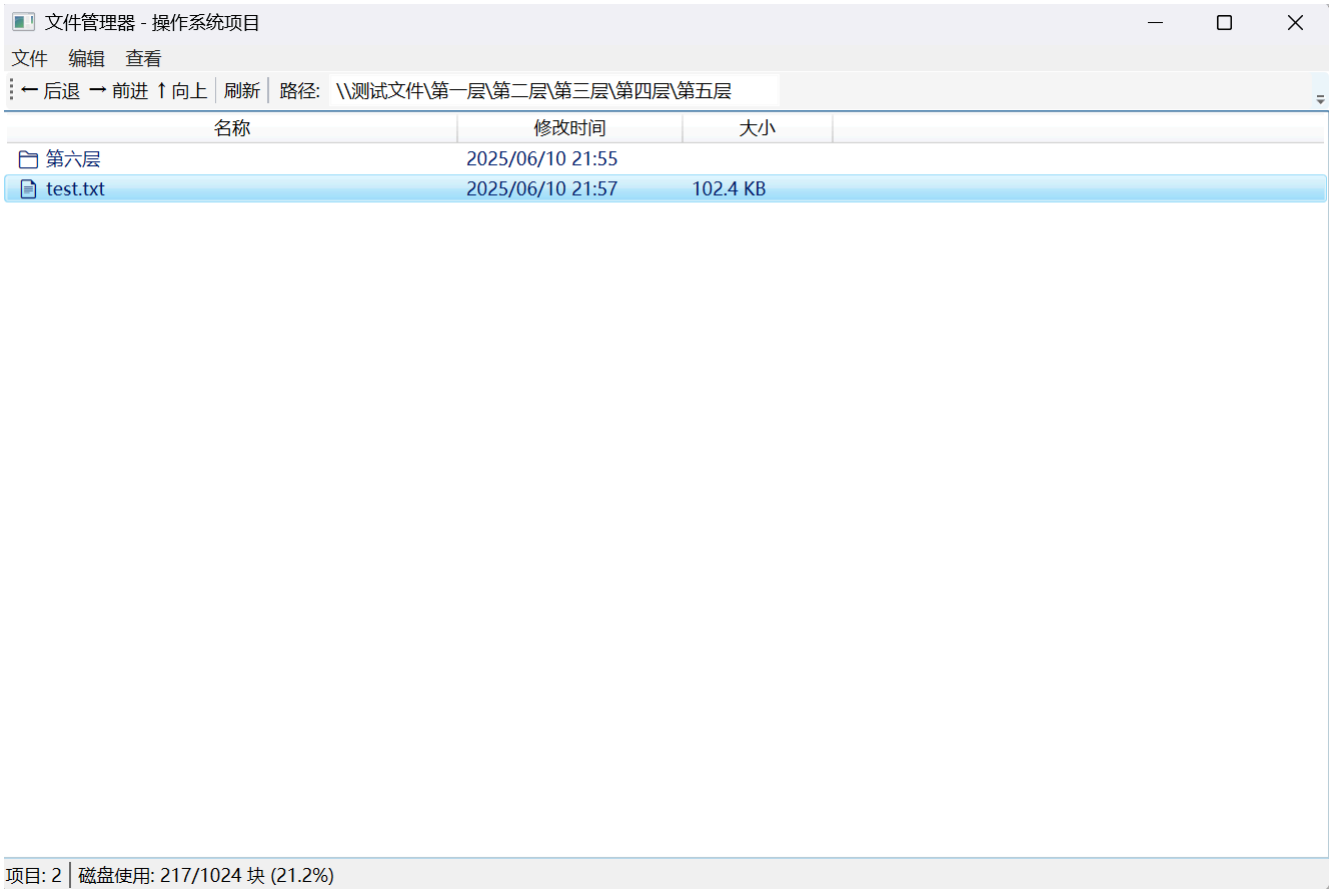

文件的重命名：





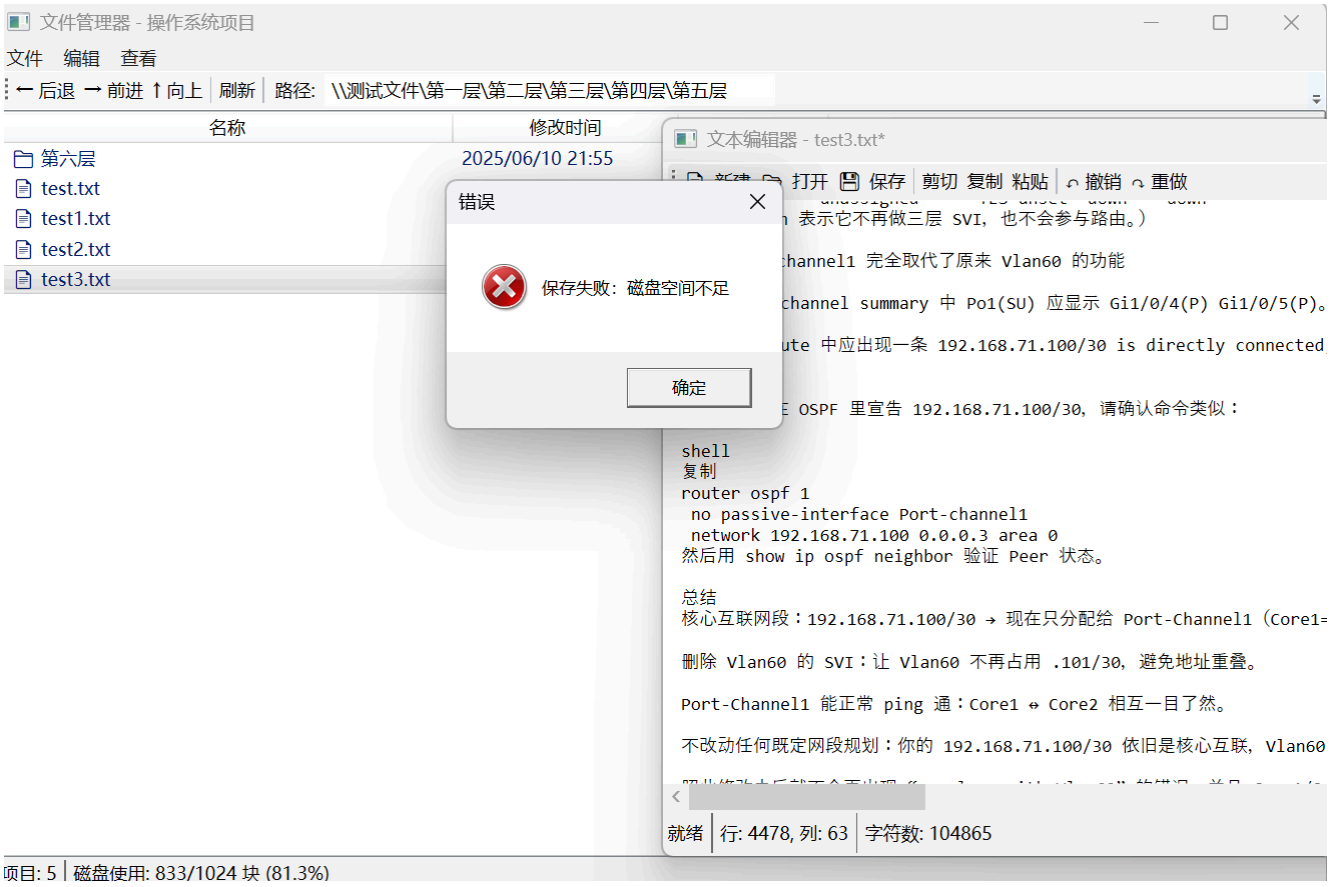
## 4.3 文件的嵌套

文件成功嵌套到了六层以上：




## 4.4 异常处理、报错机制

如果文件已经占用所有磁盘块空间（找不到足够多的空闲块），则会报错



# 5. 项目总结

本项目成功地使用现代软件工程技术（C#、WPF、MVVM）实现了一个模拟文件管理系统，有效地将操作系统的核心理论知识与实践相结合。

通过亲手构建文件控制块、实现位图分配算法和维护目录树结构，开发者可以深刻理解文件系统并非一个黑盒，而是由一系列定义明确的数据结构和算法构成的精密系统。项目不仅提供了文件和目录的创建、删除、重命名等基本功能，还实现了文本文件的内容编辑和保存，形成了一个功能闭环，让模拟体验更加完整。