

**操作系统进程管理项目**

题目 文件系统管理项目

姓 名 刘淑仪

学 号 2251730

学 院 软件学院

专 业 软件工程

教 师 张惠娟

二〇二四 年 六 月 十四 日

**文件系统管理项目**

刘淑仪 2251730 软件学院 软件工程

**一、项目简介**

**（一）项目目的**

1. 理解文件存储空间的管理；

2. 掌握文件的物理结构、目录结构和文件操作；

3. 实现简单文件系统管理；

4. 加深文件系统实现过程的理解；

**（二）项目需求**

1. 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;
2. 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。
3. 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；
4. 空闲空间管理可采用位图或者其他方法；
5. 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。
6. 能够格式化、创建子目录、删除子目录、显示目录、更改当前目录
7. 能够创建文件、打开文件、关闭文件、写文件、读文件、删除文件

**（三）项目开发环境**

系统：Windows 11 家庭中文版

IDE：Rider 2024.1、Visual Studio 2022

语言：C#

**二、文件系统管理的原理和实现过程**

该项目的原理基本上是模拟一个简易的文件系统，通过一个图形用户界面（GUI）进行文件和目录的管理。项目中涵盖了文件和目录的创建、修改、删除、复制、粘贴等操作，同时还支持目录的导航历史管理，允许用户进行撤销和重做（后退和前进）操作。

1. 界面设计：

项目利用 Windows Forms 提供图形用户界面，包括树形视图（TreeView）和列表视图（ListView），分别用于显示目录结构和文件详细信息。

界面上还有文本框显示当前路径，以及多个按钮用于执行各种文件操作（如新建、重命名、删除、复制、粘贴等）。

2. 数据结构：

目录（Catalog）：用于存储目录信息，包括目录下的文件和子目录列表（nodelist），目录名称（name），路径（path），以及创建和修改时间等。

文件（File）：表示单个文件，包含文件名、路径、大小、创建和修改时间以及一个数据块列表（blocklist）用于存储文件内容。

节点（Node）：抽象了文件和目录的概念，每个节点可以是一个文件或一个目录。

位图（BitMap）：用于管理数据块的分配情况，通过一个布尔数组（bitMap）标记哪些数据块是空闲的，哪些是已占用的。

3. 功能实现：

文件操作：支持创建、重命名、删除、读取、写入文件。文件内容的读取和写入操作通过数据块进行，每个数据块固定大小（如512字节），并通过位图管理空闲和占用的数据块。

目录操作：支持创建和删除目录，以及浏览目录内容。目录的创建和删除会更新父目录的信息。

视图更新：每当进行文件或目录操作后，界面上的视图会相应更新，反映最新的文件系统状态。

序列化和反序列化：使用二进制格式化器（BinaryFormatter）将整个文件系统的状态（包括所有目录和文件的信息）保存到磁盘，并在程序启动时加载这些数据。

4. 事件处理：

文件和目录的操作通过点击按钮或菜单项触发，事件处理函数负责执行具体的操作逻辑（如调用相关的创建、删除、重命名等方法）。

文件编辑通过另一个表单（TxtEditor）处理，支持文本文件的内容编辑和保存。

**三、项目架构**

该文件管理器项目的架构主要围绕着模拟一个简单的虚拟文件系统，通过图形用户界面（GUI）让用户能够执行文件和目录的操作。

1. 用户界面层 (Presentation Layer)：

1. Windows Forms：使用 Windows Forms 构建的 GUI 提供了用户交互界面。界面包括树形视图来展示目录结构，列表视图来显示文件和目录的详细信息，以及多个按钮和菜单项用于执行文件操作（如新建、编辑、删除、复制、粘贴等）。
2. 事件处理：包括单步执行、连续执行和复位功能。按钮点击后触发相应的方法进行操作。

2. 业务逻辑层 (Business Logic Layer)：

1. 文件操作：核心的文件操作逻辑，包括创建、读取、更新、删除（CRUD）文件和目录。
2. 路径管理：处理文件和目录的路径逻辑，包括路径的解析、修改和存储。
3. 错误处理：进行错误检测和处理，确保在执行文件操作时，任何异常都能被妥善管理。

3. 数据访问层 (Data Access Layer)

1. 序列化与反序列化：使用 .NET 的 BinaryFormatter 对象进行对象的序列化和反序列化，以持久化文件系统的状态到磁盘或从磁盘恢复。
2. 位图管理：使用自定义的 BitMap 类来管理文件数据的存储，标记哪些数据块已被使用，哪些仍然是空闲的。

4. 数据模型层 (Data Model Layer)

1. 文件（File）：表示文件的类，包含文件名、文件类型、大小、创建和更新时间以及文件内容的数据块列表。
2. 目录（Catalog）：表示目录的类，包含目录名、路径、创建和更新时间，以及包含的文件和子目录的列表。
3. 节点（Node）：通用的节点类，用于表示文件系统中的一个实体，可以是文件或目录。
4. 数据块（Block）：表示文件内容的基本存储单位。

5. 辅助功能层 (Auxiliary Functions)

1. 文本编辑器：为编辑文本文件提供的简易文本编辑器，包含基本的文本操作和保存功能。
2. 历史记录管理：支持导航历史的功能，允许用户撤销和重做导航操作。
3. 复制粘贴功能：实现文件和目录的复制粘贴操作，包括处理重名等问题。

6. 持久化层 (Persistence Layer)

文件存储：文件系统的状态通过序列化保存在本地磁盘上，确保数据在程序关闭后仍然可用。

**四、界面设计**

**（一） 双面板布局**

1. 左侧面板：采用树形视图（TreeView）显示目录结构，使用户能够一目了然地看到文件和目录的层次关系，方便导航。
2. 右侧面板：列表视图（ListView）用来显示选中目录中的内容，包括文件和子目录的名称、修改日期、类型和大小。

**（二） 工具条**

包括后退按钮和一些可能的功能按钮（如新建文件夹、文件等），这些都是进行文件操作的快捷方式。

**（三） 状态栏**

在界面底部，展示当前目录的完整路径，帮助用户确认他们正在操作的文件路径。。

**（四） 简洁的设计**

1. 界面采用了简洁明了的设计风格，使用的图标和文字大小适中，易于操作，没有过多的视觉干扰元素。
2. 色彩使用较为简单，主要以白色背景搭配深色文字，清晰显示信息。

**（五） 功能分类明确**

1. 用户可以很容易地识别出每个部分的功能，如文件夹和文件的图标不同，使得视觉上一目了然。
2. 文件和文件夹的详细信息排列整齐，包括修改时间、类型和大小，这对于文件管理非常关键。

**（六） 响应式交互**

1. 点击树形视图中的任一节点会在右侧列表视图中显示该节点的内容，实现了良好的交互性。
2. 可以预见的是，这些界面元素（如按钮和菜单项）很可能支持点击事件，执行相关的文件操作。

**（七） 整体界面预览**

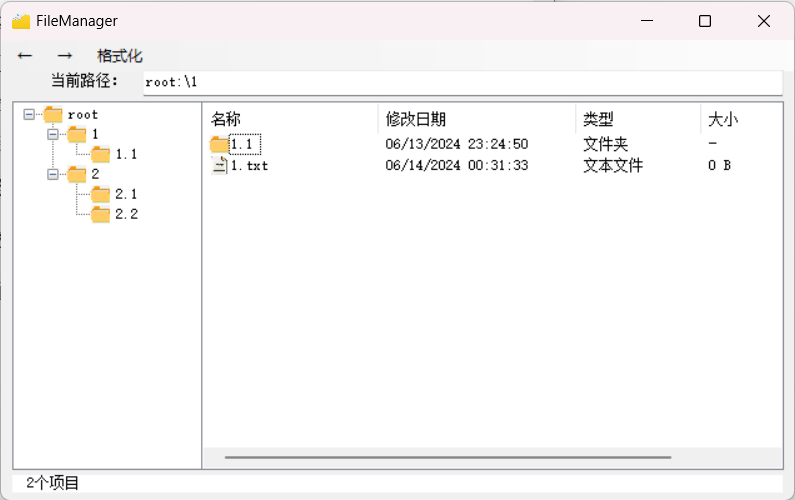


图1 整体界面预览

**五、核心算法**

**（一） 位图管理 (BitMap)**

这是文件系统中管理磁盘空间的关键算法。位图（BitMap）类通过一个布尔数组来跟踪哪些数据块是被占用的，哪些是空闲的。核心功能包括：

1. 查找空闲块：遍历位图，找到第一个标记为true（表示空闲）的索引。

2. 设置占用：在文件写入数据块时，将对应的位图索引设置为false，表示该块已被占用。

3. 设置空闲：删除文件或清空数据块时，将对应的位图索引设置为true。

**（二） 文件写入和读取**

这部分处理如何将文件数据写入到磁盘的数据块中，以及如何从数据块中读取文件数据。

1. 写入文件：将文件内容按照块大小（例如512字节）分割，每一部分存储到一个数据块中。如果数据块不足以存放全部数据，则继续分配下一个空闲块。这涉及到位图的使用来找到空闲块。

2. 读取文件：根据文件的数据块列表，逐个读取数据块的内容并合并，最终构成完整的文件内容。

**（三）** **序列化和反序列化**

为了持久化文件系统状态，使用.NET的BinaryFormatter进行对象的序列化和反序列化。

1. 序列化：在应用关闭或需要保存当前状态时，将所有目录和文件对象序列化到文件中。

2. 反序列化：启动应用时，从文件中读取序列化数据，恢复文件系统的状态。

**（四）** **界面更新算法**

管理如何根据文件系统的状态更新用户界面元素，例如树视图和列表视图。

1. 更新树视图：递归遍历目录结构，为每个目录和子目录创建树节点。

2. 更新列表视图：显示当前目录的内容，包括所有文件和子目录的详细信息。

**（五）** **导航和历史记录管理**

处理用户在文件系统中的前进和后退操作。

前进和后退：通过维护一个历史记录栈，用户每次导航时将当前目录状态压入栈中。后退操作通过弹出栈来恢复到前一个目录状态。

**（六）** **核心数据结构与组件**

1. 数据结构

1) FCB 结构

public int size; *// 大小*public String name; *// 文件名*public DateTime createdTime; *// 创建时间*public DateTime updatedTime; *// 修改时间*public List<Block> blocklist; *// 文件指针*public String path;

2) BitMap 结构

public const int **BYTE\_SIZE** = 8;  
public const int **MAX\_CAPCITY** = 100 \* 100;  
public const int **BYTENUMBER** = 100 \* 100 / 8;  
public Block[] blocks = new Block[**MAX\_CAPCITY**];  
public bool[] bitMap = new bool[**MAX\_CAPCITY**];

3) Catalog 结构

public List<Node> nodelist;  
public int childrenNum;  
public String name;  
public String path;  
public int fileSize;  
public DateTime createdTime;  
public DateTime updatedTime;  
public Catalog parenCatalog;

2. 关键窗体和组件：

|  |  |  |
| --- | --- | --- |
| 窗体、组件或函数 | 功能 | 截图 |
| FileManager | 主界面构建和初始化参数 |  |
| FileCreator | 创建新文件或重命名现有文件，并通过界面交互接收用户输入,更新上级目录修改时间 |  |
| FileAttributes | 显示文件或文件夹的详细属性，包括名称、类型、路径、大小和创建及更新时间。 |  |
| TxtEditor | 编辑和保存文本文件内容，并在编辑完成后可执行回调函数更新相关状态。 |  |
| contextMenuScript1 | 选择对文件的操作 |  |
| menuScript1 | 返回上一次操作、撤销上一次操作以及格式化 |  |
| checkName() | 检查是否重名 |  |
| updateTreeView() | 更新目录表视图 |  |
| updateListView() | 更新文件列表视图 | 略 |
| FileManager\_FormClosing() | 保存至本地 | 略 |

**六、项目整体运行测试**

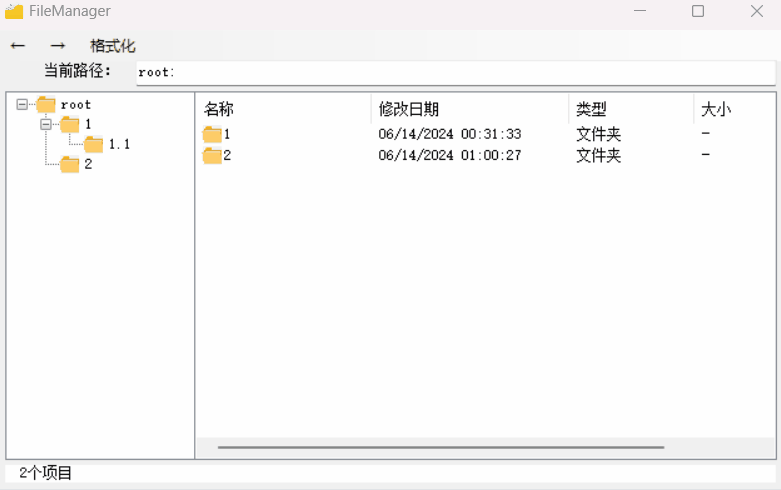


图2 整体运行测试图示例图

**七、项目总结**

**（一）项目改进建议**

1. 用户交互：增强功能性、提升响应性和优化用户体验。通过引入搜索功能、多选操作、异步加载以提高响应性、文件预览、安全的数据处理如回收站功能、支持拖放操作、提供多种视图选项以及自定义设置，可以显著提高用户的操作便利性和满意度。这些改进将使文件管理器不仅功能全面，而且用户友好，适应更广泛的使用场景和用户需求。
2. 算法设计：在算法设计方面，文件管理器项目的改进可以集中在增强其核心功能的效率和可维护性上。首先，通过引入模块化和解耦，将大型功能拆分成更易管理的小模块，每个模块专注于单一职责，这样可以提高代码的可读性和易维护性。其次，使用合适的设计模式如观察者模式和命令模式，可以优化数据和视图的更新机制，以及统一处理文件操作，从而提高系统的灵活性和扩展性。此外，加强错误处理和日志记录，不仅增强系统的健壮性，还便于问题的追踪和调试。性能优化也是关键，通过改进数据结构选择和算法逻辑，尤其是在文件的读写和位图管理上，可以显著提升应用的执行效率。最后，引入异步编程减少界面延迟，保持界面响应性，同时实施单元测试和集成测试确保代码质量，通过这些策略，项目的整体工程质量和用户体验都将得到提升。
3. 性能优化：

1) 优化数据结构和存储：针对文件和目录的存储，选择更高效的数据结构，如使用哈希表或平衡树（例如 AVL 树或红黑树）来管理文件节点，可以加快搜索、插入和删除操作的速度。

2) 异步和多线程处理：对于耗时的文件操作（如大文件的读写、复杂的目录操作等），应采用异步编程模式，使用多线程来处理，避免界面阻塞，提升响应速度和用户体验。

3) 利用缓存机制：对于频繁访问的文件和数据，实现缓存策略，减少对磁盘的直接读写次数，从而提高数据访问速度。

4) 减少不必要的界面刷新：在界面上显示文件系统的信息时，合理控制刷新频率和范围，避免全局刷新，只更新必要的部分，减轻渲染负担。

5) 批处理文件操作：在执行大量文件操作（如批量删除、移动等）时，采用批处理技术，合并操作命令，减少对文件系统的总体负担。

6) 资源管理和释放：确保所有文件操作后，及时释放不再使用的资源，如文件句柄、内存等，避免内存泄漏和资源占用。

**（二）总结**

本文件管理器项目的亮点在于其提供了一个直观且功能丰富的用户界面，支持基础的文件操作和视图更新，以及通过序列化实现的数据持久化。然而，项目在处理大量数据或复杂操作时可能存在性能瓶颈，界面响应速度需进一步优化。此外，界面虽功能全面但缺乏一些高级功能，如文件搜索和内容预览。为此，建议进一步拓展功能，例如加入异步处理机制提高响应速度，引入文件预览和高效的搜索功能，以及通过实现更高级的数据结构和缓存机制来提升整体性能和用户体验。这些改进将使得文件管理器更加强大，更能满足专业用户的需求。