文件管理系统

2350939 卜天

项目简介

本项目是一个基于 FAT 文件系统的文件管理系统，采用 Web 界面实现文件和目录的创建、删除、重命名、编辑等操作。系统模拟了真实操作系统中的文件管理机制，包括 FAT 表管理、磁盘块分配、目录树结构等核心算法。

本项目已被部署到作者本人的 Ubuntu 服务器，可通过http://124.223.93.75:5001直接访问体验。

技术栈

- **后端**: Flask
- **前端**: Bootstrap 5 + JavaScript
- **图标**: Bootstrap Icons
- **存储**: FAT 表和多级目录结构
- **数据持久化**: Python pickle 序列化

核心算法实现

1. 磁盘存储结构

物理块设计

```
BLOCKSIZE = 512  # 每个物理块大小
BLOCKNUM = 512   # 磁盘中物理块个数
```

系统将磁盘划分为 512 个物理块，每个块大小为 512 字节。每个物理块包含：

- `blockIndex`：块编号
- `data`：存储的数据内容

块操作算法

- **写入算法**: 数据超过块大小时自动截断 ’返回剩余数据
- **追加算法**: 计算剩余空间 ’优先填满当前块
- **清空算法**: 重置块数据为空字符串

2. FAT 文件分配表

FAT 表结构

FAT 表是一个长度为 BLOCKNUM 的数组 ’每个元素表示对应块的状态 ：

- `-2` : 空闲块
- `-1` : 文件结束块
- `正整数` : 指向下一个块的索引

核心算法

空闲块查找算法:

```
def findBlank(self):
    for i in range(BLOCKNUM):
        if self.fat[i] == -2:
            return i
    return -1
```

文件写入算法:

1. 查找空闲块
2. 建立链表结构连接多个块
3. 最后一个块标记为-1
4. 返回起始块索引

文件删除算法:

1. 从起始块开始遍历链表
2. 清空每个块的数据
3. 将 FAT 表项重置为-2
4. 释放所有相关块

文件更新算法:

1. 先删除原有数据
2. 重新分配空间写入新数据
3. 返回新的起始块索引

3. 文件控制块(FCB)

FCB 包含文件的元数据信息 ：

- `name`：文件名
- `createTime`：创建时间
- `updateTime`：最后修改时间
- `start`：文件数据起始块索引

4. 目录树结构

目录节点设计

```
class CatalogNode:
    def __init__(self, name, isFile, fat, disk, createTime, parent=None, data=""):
        self.name = name          # 节点名称
        self.isFile = isFile      # 是否为文件
        self.parent = parent      # 父节点引用
        self.createTime = createTime
        self.updateTime = createTime

        if not self.isFile:
            self.children = []    # 子节点列表
        else:
            self.data = FCB(...)  # 文件控制块
```

路径导航算法

```
def get_current_node(self):
    current_node = self.catalog[0]  # 从根节点开始
    for path_part in self.current_path[1:]:  # 逐级遍历路径
        for child in current_node.children:
            if child.name == path_part and not child.isFile:
                current_node = child
                break
    return current_node
```

5. 文件系统管理算法

文件创建算法

1. 检查当前目录下是否存在同名文件
2. 创建新的 CatalogNode 节点
3. 如果是文件 ’创建对应的 FCB
4. 将节点添加到父目录的 children 列表
5. 持久化文件系统状态

文件删除算法

1. 在当前目录查找目标文件/文件夹
2. 如果是文件 ’调用 FCB 的 delete 方法释放磁盘空间
3. 如果是文件夹 ’递归删除所有子项
4. 从父目录的 children 列表中移除节点

递归删除算法

```
def _delete_folder_recursive(self, folder_node):
    for child in folder_node.children:
        if child.isFile:
            child.data.delete(self.fat, self.disk)  # 释放文件占用的磁盘空间
        else:
            self._delete_folder_recursive(child)    # 递归删除子文件夹
```

6. 数据持久化机制

系统使用 Python 的 pickle 模块实现数据持久化 ：

- `catalog`: 存储目录树结构
- `fat`: 存储 FAT 文件分配表
- `disk`: 存储所有磁盘块数据

每次文件系统操作后都会调用`save_file_system()`方法保存状态 。

功能特性

核心功能

- **文件管理**: 创建 `删除 `重命名文件和文件夹
- **文件编辑**: 内置文本编辑器 ’支持实时保存
- **目录导航**: 多级目录结构浏览
- **文件属性**: 查看文件创建时间 `修改时间等信息
- **磁盘格式化**: 清空所有数据重新开始

界面特性

- **响应式设计**: 支持桌面和移动设备
- **现代化 UI**: 基于 Bootstrap 5 的美观界面
- **右键菜单**: 类似桌面系统的操作体验
- **快捷键支持**: Delete 删除 `F2 重命名 `Ctrl+S 保存
- **实时反馈**: 操作结果即时显示

增强功能

- **文件编辑器增强**: 字符统计 `行数统计 `自动保存提醒
- **拖拽支持**: 计划中的功能
- **搜索功能**: 计划中的功能
- **文件预览**: 计划中的功能

使用指南

基本操作

1. **创建文件/文件夹**
   - 点击工具栏的"新建文件夹"或"新建文件"按钮
   - 或在空白区域右键选择创建选项
2. **打开文件/文件夹**
   - 双击文件打开编辑器
   - 双击文件夹进入目录
3. **文件操作**
   - 右键文件/文件夹显示操作菜单
   - 支持重命名 `删除 `查看属性
4. **导航**
   - 使用面包屑导航快速跳转
   - 点击"返回"按钮回到上级目录

快捷键

- `Delete`: 删除选中的文件/文件夹
- `F2`: 重命名选中的文件/文件夹
- `Ctrl+S` / `Command+S`: 在文件编辑器中保存文件

文件编辑器

- **实时统计**: 显示字符数 `行数
- **保存状态**: 实时显示保存状态
- **快捷保存**: Ctrl+S 快速保存
- **关闭提醒**: 未保存时关闭会提醒

项目结构

```
FileManagement/
├── app.py                    # Flask主应用，包含FileSystemManager类
├── File.py                   # 核心数据结构定义
│   ├── Block                 # 磁盘物理块类
│   ├── FAT                   # 文件分配表类
│   ├── FCB                   # 文件控制块类
│   └── CatalogNode           # 目录节点类
├── requirements.txt          # Python依赖包
├── README.md                 # 项目文档
├── templates/                # HTML模板文件
│   ├── base.html            # 基础模板
│   ├── index.html           # 文件管理主界面
│   └── edit_file.html       # 文件编辑器界面
├── catalog                   # 序列化的目录树数据
├── disk                      # 序列化的磁盘块数据
└── fat                       # 序列化的FAT表数据
```

系统架构设计

分层架构

```
┌─────────────────────────────────────────┐
│            Web界面层                    │
│    (Bootstrap + JavaScript)             │
├─────────────────────────────────────────┤
│            Flask路由层                  │
│      (RESTful API接口)                  │
├─────────────────────────────────────────┤
│         文件系统管理层                  │
│     (FileSystemManager)                 │
├─────────────────────────────────────────┤
│          数据结构层                     │
│  (CatalogNode, FCB, FAT, Block)         │
├─────────────────────────────────────────┤
│          持久化存储层                   │
│      (pickle序列化文件)                 │
└─────────────────────────────────────────┘
```

核心类关系图

```
FileSystemManager
    ├── catalog: List[CatalogNode]  # 目录树根节点列表
    ├── fat: FAT                    # 文件分配表
    └── disk: List[Block]           # 磁盘块列表

CatalogNode
    ├── name: str                   # 节点名称
    ├── isFile: bool                # 是否为文件
    ├── parent: CatalogNode         # 父节点引用
    ├── children: List[CatalogNode] # 子节点列表(目录)
    └── data: FCB                   # 文件控制块(文件)

FCB (文件控制块)
    ├── name: str                   # 文件名
    ├── createTime: time            # 创建时间
    ├── updateTime: time            # 修改时间
    └── start: int                  # 起始块索引

FAT (文件分配表)
    └── fat: List[int]              # FAT表数组

Block (磁盘块)
    ├── blockIndex: int             # 块索引
    └── data: str                   # 块数据
```

API 接口设计

方法	路径	功能	参数
GET	`/`	主页面	-
POST	`/navigate`	目录导航	`{"path": ["root", "folder1"]}`
POST	`/create_file`	创建文件	`{"name": "filename.txt"}`
POST	`/create_folder`	创建文件夹	`{"name": "foldername"}`
POST	`/delete`	删除文件/	`{"name": "itemname"}`

方法	路径	功能	参数
		文件夹	
POST	`/rename`	重命名	`{"old_name": "old", "new_name": "new"}`
GET	`/edit_file/<filename>`	文件编辑页面	-
POST	`/save_file`	保存文件	`{"filename": "file.txt", "content": "..."}`
GET	`/file_info/<filename>`	获取文件信息	-
POST	`/format`	格式化磁盘	-

安装与运行

环境要求

- Python 3.7+
- Flask 2.0+

安装步骤

1. 克隆项目到本地

```
git clone <repository-url>
cd FileManagement
```

2. 安装依赖

```
pip install -r requirements.txt
```

3. 运行应用

```
python app.py
```

4. 访问应用

打开浏览器访问 `http://localhost:5001`

首次运行

首次运行时，系统会自动创建以下文件：

- `catalog`：目录结构数据
- `fat`：FAT 表数据
- `disk`：磁盘块数据

技术特点

算法优势

1. **高效的空间管理**
   - 采用 FAT 表进行磁盘空间分配
   - 支持文件的动态扩展和收缩
   - 自动回收删除文件的磁盘空间
2. **灵活的目录结构**
   - 树形目录结构支持任意深度嵌套
   - 父子节点双向引用便于快速导航
   - 支持同名文件在不同目录下共存
3. **可靠的数据持久化**
   - 使用 pickle 序列化保证数据完整性
   - 每次操作后立即保存，防止数据丢失
   - 支持系统重启后数据恢复

性能特性

- **时间复杂度**:
  - 文件查找: O(n) - n 为当前目录文件数
  - 文件创建: O(1)
  - 文件删除: O(m) - m 为文件占用的块数
  - 目录遍历: O(d) - d 为目录深度
- **空间复杂度**:
  - FAT 表: O(BLOCKNUM) = O(512)

- 目录树: O(文件总数)
- 磁盘块: O(BLOCKNUM × BLOCKSIZE)

扩展性设计

系统采用模块化设计，便于功能扩展：

1. **存储层扩展**
   - 可替换 pickle 为数据库存储
   - 支持增加磁盘块大小和数量
   - 可实现磁盘碎片整理功能
2. **功能层扩展**
   - 可添加文件权限管理
   - 支持文件压缩和加密
   - 可实现文件版本控制
3. **界面层扩展**
   - 支持拖拽操作
   - 可添加文件搜索功能
   - 支持批量操作

项目总结

本文件管理系统成功实现了 FAT 文件系统的核心功能，包括：

- **完整的文件系统架构**: 从底层磁盘块到上层用户界面的完整实现
- **高效的算法设计**: FAT 表管理、目录树遍历、空间分配等核心算法
- **良好的用户体验**: 现代化 Web 界面，支持常用快捷键和右键菜单
- **可靠的数据管理**: 完整的 CRUD 操作和数据持久化机制

该项目展示了操作系统文件管理的核心原理，为理解真实文件系统的工作机制提供了良好的学习平台。