**简易文件管理系统设计文档**

**--2352835 夏弘泰**

**一、项目简介**

本项目是一个使用 Python 编写的**模拟文件系统管理程序**，通过图形用户界面（GUI）对虚拟磁盘进行操作。该系统模拟了基本文件系统的磁盘管理机制，使用链式结构和位图方式实现文件存储管理，并通过 Tkinter 提供可视化界面支持多级目录结构展示、文件读写、目录操作等。

**项目特点：**

内存模拟磁盘 + 位图管理空闲块；

支持多级目录；

支持文件内容读写；

图形化操作界面；

支持数据持久化保存。

**二、设计思路**

**1. 系统结构**

整体采用**MVC 思路分离逻辑**：

| **组件** | **功能描述** |
| --- | --- |
| FileSystem | 模拟文件系统核心逻辑（数据结构 + 操作接口） |
| FileSystemGUI | 图形界面交互与调用 FileSystem 的功能 |
| disk.data | 使用 pickle 序列化存储虚拟磁盘数据 |

**2. 存储结构设计**

**块结构**：模拟磁盘共 1024 个块，每块 64 字节；

**位图**：标记哪些块被占用（1）或空闲（0）；

**链式存储**：文件由多个块通过“指向下一个块”的方式连接；

**目录结构**：采用多层嵌套字典 {name: (type, content/block\_index)} 实现；

**当前路径**：使用栈结构记录用户当前路径，便于进入/返回目录。

**3. UI 设计思路**

使用 Tkinter 的 Treeview 展示目录结构；

支持点击展开目录、右键菜单操作；

添加状态信息输出区；

支持弹出文件内容编辑窗口。

**三、功能实现**

**1. 文件系统管理（FileSystem.py）**

格式化磁盘：清空位图、初始化根目录；

创建/删除目录；

创建/删除文件；

文件写入：分配多个磁盘块、写入内容；

文件读取：根据链表读取所有内容；

多级目录跳转（进入、返回）；

自动保存与加载磁盘状态。

**2. 图形界面交互（main.py）**

展示多级目录树结构；

图标区分文件/目录；

右键菜单支持打开、删除；

提供操作按钮实现各种功能调用；

支持刷新视图、弹出内容编辑窗口；

操作状态展示输出区。

**四、核心代码分析**

**1. 位图 + 链式分配实现**

# 分配空闲块

def allocate\_block(self):

for i in range(self.total\_blocks):

if self.bitmap[i] == 0:

self.bitmap[i] = 1

return i

raise Exception("No free blocks available")

def write\_file(self, name, content):

...

for i in range(len(content\_blocks)):

block = content\_blocks[i]

next\_block = content\_blocks[i + 1] if i + 1 < len(content\_blocks) else -1

self.disk[block] = (content[i\*64:(i+1)\*64], next\_block)

**2. 多级目录结构设计**

# 当前目录是字典嵌套结构

self.root = {}

self.current\_path = [self.root]

# 进入子目录

def enter\_directory(self, name):

current = self.current\_path[-1]

...

self.current\_path.append(current[name][1])

**3. 图形界面构建（Tkinter + Treeview）**

self.tree = ttk.Treeview(self.left\_frame, columns=('Type',), show='tree')

self.tree.pack(fill=tk.BOTH, expand=True)

# 插入节点

def refresh\_tree(self):

self.tree.delete(\*self.tree.get\_children())

self.insert\_tree\_nodes('', self.fs.root)

def insert\_tree\_nodes(self, parent, directory):

for name, (type\_, \_) in directory.items():

node = self.tree.insert(parent, 'end', text=name, image=self.folder\_icon if type\_ == 'directory' else self.file\_icon)

if type\_ == 'directory':

self.insert\_tree\_nodes(node, directory[name][1])

**4. 文件修改窗口**

def modify\_file(self):

...

edit\_window = tk.Toplevel(self.root)

text\_widget = tk.Text(edit\_window, height=20, width=60)

text\_widget.insert(tk.END, content)

...

def save\_content():

new\_content = text\_widget.get("1.0", tk.END).strip()

self.fs.modify\_file(filename, new\_content)

**五、项目运行说明**

**环境要求**

* Python 3.8+
* 第三方库：
  + Pillow（用于加载图标）
  + Tkinter（标准 GUI 库）

**依赖库准备**

pip install pillow

**启动方式**

python main.py

**使用流程**

启动后默认加载或初始化磁盘；

左侧树形结构浏览文件系统；

点击按钮或右键操作管理文件和目录；

所有更改自动保存至 disk.data。

**六、运行截图**

