

文件管理模拟系统 设计方案报告

目录

- [开发环境](#)
- [项目结构](#)
- [界面](#)
- [操作说明](#)
- [实现的功能](#)
- [系统分析](#)
 - [实现的类](#)
 - [FCB类](#)
 - [Node类](#)
 - [Category类](#)
- [系统实现](#)
 - [为文件内容分配磁盘空间](#)
 - [获取文件内容](#)
 - [删除文件内容](#)
 - [更新文件内容](#)
 - [检查同目录下是否有同名文件](#)
 - [保存文件内容](#)
 - [创建目录树](#)
 - [磁盘文件的写入和读取](#)
 - [目录文件的写入和读取](#)

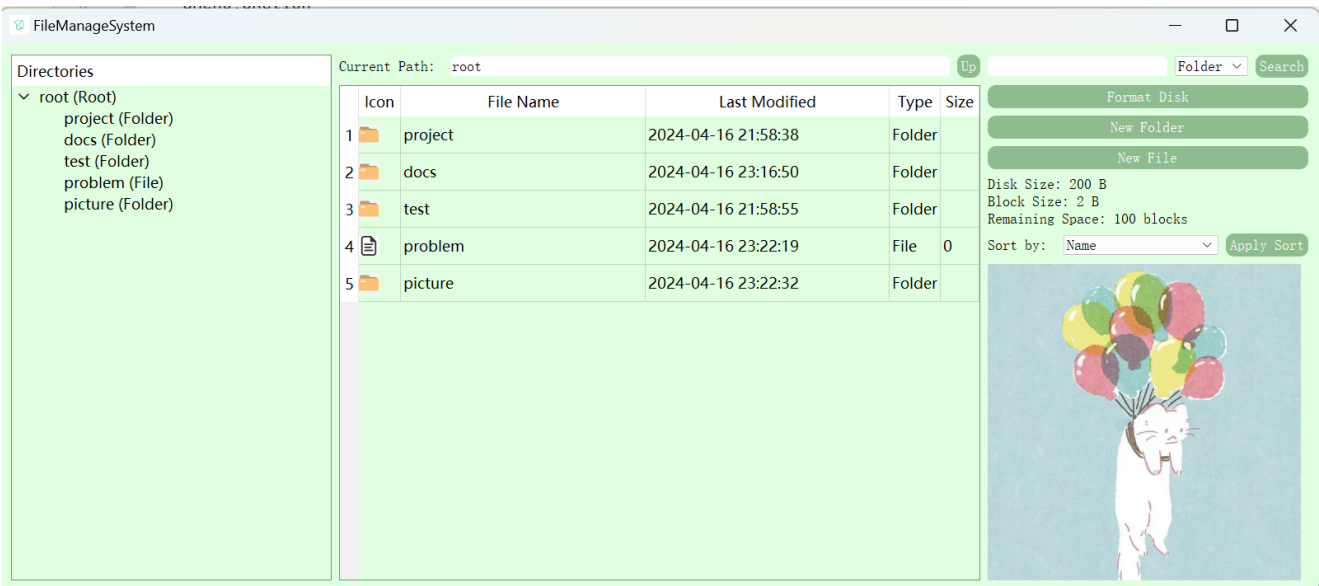
开发环境

- 开发环境：Windows 11
- 开发软件：Pycharm
- 开发语言：Python 3.11
- 主要引用模块：
 - PyQt5
 - sys
 - os

项目结构

```
1 FileManageSystem/  
2     BitMapInfo.txt  
3     Category.py  
4     CategoryInfo.txt  
5     FCB.py  
6     main.py  
7     MyDiskInfo.txt  
8     project_structure.txt  
9     virtualDisk.py  
10    运行时截图.png  
11    docs/  
12        文件管理模拟系统 设计方案报告.md  
13    res/  
14        cat.jpg  
15        file.png  
16        folder.jpg  
17        icon.jpg
```

界面



操作说明

- 点击右侧按钮可以新建文件，文件夹，格式化磁盘；
- 当用鼠标右键点击空白区域时，可以新建文件或文件夹；
- 当用鼠标右键点击文件或文件夹名称时，可以打开或者删除文件和文件夹；
- 点击文件或文件夹名可以打开文件编辑框或者打开文件夹；
- 当退出文件编辑框时可以选择是否保存修改；
- 界面右侧上方可以输入名称，选择类型，然后在当前目录下进行搜索；
- 可以按文件类型，文件名和修改日期对文件夹和文件进行排序。

实现的功能

- 当前目录下文件和文件夹信息的显示
- 文件和文件夹的创建与删除
- 文件夹的打开，文件的编辑
- 格式化磁盘
- 文件和文件夹的搜索
- 树状目录结构示意图
- 按文件名，修改实现，文件类型排列文件和文件夹
- 返回上级目录
- 当前路径的显示

系统分析

实现的类

- FCB类：文件控制块，记录文件名称，文件类型，修改日期，文件大小，在磁盘中的起始存储位置；
- Node类：存储子节点，记录父节点，映射为文件之间的关系；
- Category类：存储整个文件系统的目录信息，其中的root记录了根节点，提供了一些方法：
 - free_category(self, p_node)：释放指定节点的目录；
 - search(self, p_node, file_name, file_type)：在指定节点下搜索文件或文件夹；
 - search_in_current_directory(self, p_node, file_name, file_type)：只在指定目录下搜索文件或文件夹；
 - create_file(self, parent_node, fcb)：创建文件或文件夹；
 - check_same_name(self, p_node, name, file_type)：判断在同一目录下是否存在同名文件或文件夹。
- VirtualDisk类：模拟对磁盘的操作，记录了磁盘大小，存储块大小，存储块数量，剩余存储块数量，内存和位图，提供了一些方法：
 - get_block_size(self, size)：得到指定大小所要占用的存储块数量；
 - give_space(self, fcb, content)：为特定fcb分配存储content的空间，将内容写入磁盘中，并对位图做出修改；
 - get_file_content(self, fcb)：返回指定fcb在存储在磁盘中的内容；
 - delete_file_content(self, start, size)：删除指定起始位置和大小存储在磁盘上的内容；
 - file_update(self, old_start, old_size, new_fcb, new_content)：更新文件内容。
- MainWindow类：主窗口，维护了主要的程序逻辑；
- HelpDialog类：操作帮助窗口，在初次打开文件管理系统时弹出，给用户提供操作帮助；
- CreateDialog类：新建文件和文件夹窗口，在新建文件和文件夹时弹出；
- NoteForm类：编辑文本文件的界面，在编辑文本文件时弹出；

显式链接法

- 本文件系统中, 文件存储空间管理使用显示链接的方法, 文件中的内容存放在磁盘不同的块中, 每次创建文件时为文件分配数量合适的空闲块。每次写文件时按顺序将文件内容写在相应块中; 删除文件时将原先有内容的位置置为空即可。

位图、FAT表

- 磁盘空闲空间管理在位图的基础上进行改造, 将存放磁盘上文件位置信息的FAT表与传统的位图进行结合, 磁盘空闲的位置使用EMPTY = -1标识, 放有文件的盘块存放文件所在的下一个盘块的位置, 文件存放结束的盘块位置使用END = -2标识。

系统实现

为文件内容分配磁盘空间

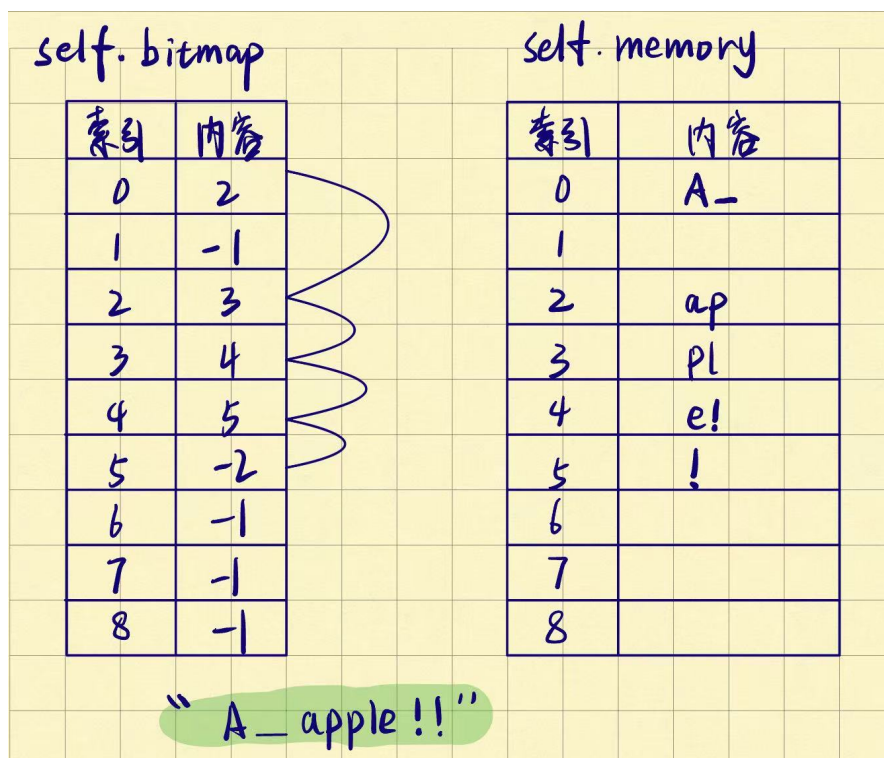
```
1  def give_space(self, fcb, content):
2      blocks = []
3      index = 0
4      while index < len(content):
5          # 特别处理换行符, 保证 '\r\n' 不被拆分
6          if content[index:index + 2] == '\r\n' and self.block_size == 2 and
index + 2 <= len(content):
7              blocks.append(content[index:index + 2])
8              index += 2
9          elif index + self.block_size <= len(content):
10             blocks.append(content[index:index + self.block_size])
11             index += self.block_size
12         else:
13             # 添加最后一个可能的小于 block_size 的块
14             blocks.append(content[index:])
15             break
16
17     if not blocks: # 如果 blocks 为空 (content 为空或其他情况)
18         return True
19
20     if len(blocks) <= self.remain:
21         # 找到文件开始存放的位置
22         start = -1
23         for i in range(self.block_num):
24             if self.bit_map[i] == self.EMPTY:
25                 self.remain -= 1
26                 start = i
27                 fcb.start = i
28                 self.memory[i] = blocks[0]
29                 break
30
31         if start == -1: # 如果没有找到空间
32             return False
33
34         # 从该位置往后开始存放内容
35         j = 1
36         i = start + 1
37         while j < len(blocks) and i < self.block_num:
```

```

38         if self.bit_map[i] == self.EMPTY:
39             self.remain -= 1
40             self.bit_map[start] = i # 以链接的方式存储每位数据
41             start = i
42             self.memory[i] = blocks[j]
43             j += 1 # 处理下一个块
44             i += 1
45
46         if j == len(blocks):
47             self.bit_map[start] = self.END # 标记文件尾
48
49         return True
50     else:
51         return False

```

首先将content进行分割，确保文本按照给定的块大小被分割，同时不会将 \r\n 换行符拆分到不同的块中。然后遍历磁盘，找到第一个空闲的存储块，将它的位置记录到fcb.start中，并往里面写入内容，同时将剩余的存储块数量减一。接着从该位置往后开始存放内容，在位图中以链接的方式存储块之间的联系，位图中的每一位存储下一个块的位置，文件尾用self.END表示。



获取文件内容

```

1  def get_file_content(self, fcb):
2      if fcb.start == self.EMPTY:
3          return ""
4      else:
5          content = ""
6          start = fcb.start
7          blocks = self.get_block_size(fcb.size)
8
9          count = 0
10         i = start
11         while i < self.block_num and count < blocks:
12             content += self.memory[i]
13             i = self.bit_map[i]
14             count += 1

```

```
15  
16         return content
```

根据位图中存储的信息将不同存储块里的内容拼接在一起。

删除文件内容

```
1 def delete_file_content(self, start, size):  
2     if start == self.EMPTY or start >= self.block_num:  
3         return # If start position is invalid or file is empty, return  
         immediately  
4  
5     blocks = self.get_block_size(size)  
6  
7     count = 0  
8     i = start  
9     while i < self.block_num and count < blocks:  
10         next_index = self.bit_map[i] # Get next index before clearing  
11         self.memory[i] = ""  
12         self.bit_map[i] = self.EMPTY  
13         self.remain += 1  
14  
15         if next_index == self.END:  
16             break # If this was the last block, exit the loop  
17  
18         i = next_index  
19         count += 1
```

更新文件内容

```
1 def file_update(self, old_start, old_size, new_fcb, new_content):  
2     self.delete_file_content(old_start, old_size)  
3     return self.give_space(new_fcb, new_content)
```

检查同目录下是否有同名文件

```
1 def check_same_name(self, p_node, name, file_type):  
2     if p_node is None:  
3         return True  
4     # 只检查给定节点（即父节点）的直接子节点  
5     for child in p_node.children:  
6         if child.fcb.file_name == name and child.fcb.file_type == file_type:  
7             return False # 找到一个同名同类型的直接子节点，返回 False  
8     return True # 在同级目录中没有找到同名同类型的文件，返回 True
```

保存文件内容

```
1     def save_content(self):  
2         content = self.textEdit.toPlainText()  
3         fcb = self.main_form.category.search(self.main_form.current_node,  
self.filename, FCB.TXTFILE).fcb  
4         old_size = fcb.size  
5         new_size = len(content)  
6         current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
7  
8         # Update file size and modification time  
9         fcb.size = new_size  
10        fcb.last_modify = current_time
```

```

11
12     # Attempt to update the file content on the disk
13     if not self.main_form.disk.file_update(fcb.start, old_size, fcb,
content):
14         QMessageBox.critical(self, 'Error', 'Failed to save file on disk.')
15     else:
16         QMessageBox.information(self, 'Success', 'File saved successfully.')
17
18     # Update the modification time for all parent nodes
19     node = self.main_form.category.search(self.main_form.current_node,
self.filename, FCB.TXTFILE)
20     while node.parent:
21         node.parent.fcb.last_modify = current_time
22         node = node.parent
23
24     self.main_form.update_disk_info()
25     # Assume a method to update the UI to reflect changes
26     self.main_form.display_file_folder_info(fcb.file_name, fcb.last_modify,
fcb.file_type, fcb.size)
27     #self.main_form.file_form_init(self.main_form.current_node) # Refresh
the view to show updated times
28

```

创建目录树

```

1     def setup_tree(self):
2         # 清除现有的树结构
3         self.tree.clear()
4         # 创建一个递归函数来填充树视图
5         def add_items(parent_item, node):
6             # 根据当前节点的信息创建一个新的树项目
7             if node.fcb.file_type == FCB.FOLDER:
8                 item = QTreeWidgetItem(parent_item, [node.fcb.file_name + "
(Folder)"])
9             else:
10                 item = QTreeWidgetItem(parent_item, [node.fcb.file_name + "
(File)"])
11
12             # 递归地为每个子节点添加树项目
13             for child in node.children:
14                 add_items(item, child)
15
16         # 检查根节点是否存在
17         if self.category.root is not None:
18             # 创建根节点对应的树项目
19             root_item = QTreeWidgetItem(self.tree,
[self.category.root.fcb.file_name + " (Root)"])
20             self.tree.addTopLevelItem(root_item)
21
22         # 为根节点的每个子节点添加树项目
23         for child in self.category.root.children:
24             add_items(root_item, child)
25
26         # 展开根节点，以便默认显示所有子节点
27         root_item.setExpanded(True)
28     else:
29         print("No root node is defined in the category.")

```

磁盘文件的写入和读取

```
1     def read_my_disk(self):
2         path = os.path.join(os.getcwd(), "MyDiskInfo.txt")
3         if os.path.exists(path):
4             with open(path, 'r', encoding='utf-8') as reader:
5                 # 首先读取磁盘的剩余容量信息
6                 remain_line = reader.readline().strip()
7                 if remain_line.startswith("Remaining Blocks:"):
8                     self.disk.remain = int(remain_line.split(":")[1].strip())
9
10                for i in range(self.disk.block_num):
11                    line = reader.readline()
12                    #if line == '\n': # 检查是否是空行，只有换行符的行
13                        #continue
14
15                    # Decode the line, handling all types of newlines
16                    line = line.rstrip("\n") # Remove only the newline at the
17end                line = line.replace("|||", "\r\n").replace("|r|",
18"\r").replace("|n|", "\n")
19
20                self.disk.memory[i] = line
21
22    def write_my_disk(self):
23        path = os.path.join(os.getcwd(), "MyDiskInfo.txt")
24        if os.path.exists(path):
25            os.remove(path)
26
27        with open(path, 'w', encoding='utf-8') as writer:
28            # 写入磁盘的剩余容量
29            writer.write(f"Remaining Blocks: {self.disk.remain}\n")
30
31            for data in self.disk.memory:
32                # 输出即将被编码的原始数据
33                print("Original data:", repr(data))
34
35                # Encode the line, handling all types of newlines
36                encoded_data = data.replace("\r\n", "|||").replace("\r",
37"|r|").replace("\n", "|n|")
38
39                # 打印编码后的数据以确认转换正确
40                print("Encoded data:", repr(encoded_data))
41
42            writer.write(encoded_data + '\n') # 写入转换后的数据加上行分隔符
```

对特殊字符, "\r\n", "\r", "\n"进行特殊编码处理, 在读入时使用line = reader.readline()保证读取整行数据, 包括换行符, 再用line = line.rstrip("\n") 移除行末换行符。

目录文件的写入和读取

```
1     def read_category(self):
2         with open("CategoryInfo.txt", 'r') as file:
3             lines = file.readlines()
4             root_node = None
5             parent_stack = []
```



```

6         current_node_info = {}
7
8     for line in lines:
9         line = line.strip()
10        if "Node Start" in line:
11            current_node_info = {}
12        elif "Node End" in line:
13            fcb = FCB(current_node_info['File Name'],
14                      int(current_node_info['File Type']),
15                      current_node_info['Last Modified'],
16                      int(current_node_info['File Size']),
17                      int(current_node_info['Start Position']))
18            new_node = Category.Node(fcb)
19            if parent_stack:
20                parent_stack[-1].add_child(new_node)
21            else:
22                root_node = new_node # 标记根节点
23                parent_stack.append(new_node) # 添加当前节点到栈，用作后续子节
点的父节点
24
25        elif "Parent End" in line and parent_stack:
26            parent_stack.pop() # 当一个节点的所有子节点都被处理完毕，从栈中
移除该节点
27
28        else:
29            if line:
30                parts = line.split(": ", 1)
31                if len(parts) == 2:
32                    key, value = parts
33                    current_node_info[key.strip()] = value.strip()
34
35        if not root_node:
36            default_fcb = FCB("root", FCB.FOLDER, "", 0)
37            root_node = Category.Node(default_fcb)
38
39        self.category.root = root_node
40        self.root_node = root_node
41        self.current_node = root_node
42        self.file_form_init(self.category.root)
43
44    def write_category(self):
45        with open("CategoryInfo.txt", 'w') as file:
46            def write_node(node, parent_name=""):
47                file.write("Node Start\n")
48                file.write(f"Parent Name: {parent_name}\n")
49                file.write(f"File Name: {node.fcb.file_name}\n")
50                file.write(f"File Type: {node.fcb.file_type}\n")
51                file.write(f>Last Modified: {node.fcb.last_modify}\n")
52                file.write(f"File Size: {node.fcb.size}\n")
53                file.write(f"Start Position: {node.fcb.start}\n")
54                file.write("Node End\n")
55                for child in node.children:
56                    write_node(child, node.fcb.file_name)
57                file.write("Parent End\n") # 标记父节点的结束
58
59        if self.category.root:
60            write_node(self.category.root)

```

将目录结构和文件信息写入目录文件中，如果目录结构如下：

Directories

```

  root (Root)
    docs (Folder)
    res (Folder)
      test (Folder)
      project (Folder)
    yik (Folder)
      code (Folder)
```

那么目录文件的内容为:

```

1 Node Start
2 Parent Name:
3 File Name: root
4 File Type: 1
5 Last Modified: 2024-04-17 23:25:17
6 File Size: 0
7 Start Position: -1
8 Node End
9 Node Start
10 Parent Name: root
11 File Name: docs
12 File Type: 1
13 Last Modified: 2024-04-17 23:24:37
14 File Size: 0
15 Start Position: -1
16 Node End
17 Parent End
18 Node Start
19 Parent Name: root
20 File Name: res
21 File Type: 1
22 Last Modified: 2024-04-17 23:25:04
23 File Size: 0
24 Start Position: -1
25 Node End
26 Node Start
27 Parent Name: res
28 File Name: test
29 File Type: 1
30 Last Modified: 2024-04-17 23:24:58
31 File Size: 0
32 Start Position: -1
33 Node End
34 Parent End
35 Node Start
36 Parent Name: res
37 File Name: project
38 File Type: 1
39 Last Modified: 2024-04-17 23:25:04
40 File Size: 0
41 Start Position: -1
42 Node End
43 Parent End
44 Parent End
45 Node Start
46 Parent Name: root
```

```
47 File Name: yik
48 File Type: 1
49 Last Modified: 2024-04-17 23:25:17
50 File Size: 0
51 Start Position: -1
52 Node End
53 Node Start
54 Parent Name: yik
55 File Name: code
56 File Type: 1
57 Last Modified: 2024-04-17 23:25:17
58 File Size: 0
59 Start Position: -1
60 Node End
61 Parent End
62 Parent End
63 Parent End
```

为了方便说明原理，对上面文件内容添加缩进：

```
1 Node Start
2 Parent Name:
3 File Name: root
4 File Type: 1
5 Last Modified: 2024-04-17 23:25:17
6 File Size: 0
7 Start Position: -1
8 Node End
9
10 Node Start
11 Parent Name: root
12 File Name: docs
13 File Type: 1
14 Last Modified: 2024-04-17 23:24:37
15 File Size: 0
16 Start Position: -1
17 Node End
18 Parent End
19
20 Node Start
21 Parent Name: root
22 File Name: res
23 File Type: 1
24 Last Modified: 2024-04-17 23:25:04
25 File Size: 0
26 Start Position: -1
27 Node End
28
29 Node Start
30 Parent Name: res
31 File Name: test
32 File Type: 1
33 Last Modified: 2024-04-17 23:24:58
34 File Size: 0
35 Start Position: -1
36 Node End
37 Parent End
38
```

```
39         Node Start
40         Parent Name: res
41         File Name: project
42         File Type: 1
43         Last Modified: 2024-04-17 23:25:04
44         File Size: 0
45         Start Position: -1
46         Node End
47         Parent End
48     Parent End
49
50     Node Start
51     Parent Name: root
52     File Name: yik
53     File Type: 1
54     Last Modified: 2024-04-17 23:25:17
55     File Size: 0
56     Start Position: -1
57     Node End
58
59         Node Start
60         Parent Name: yik
61         File Name: code
62         File Type: 1
63         Last Modified: 2024-04-17 23:25:17
64         File Size: 0
65         Start Position: -1
66         Node End
67         Parent End
68     Parent End
69 Parent End
70
```

可以看出，该文件具有以下特点：

- 每个节点以Node Start标记开始，以Node End标记结束；
- 当一个节点的子节点全部被列出时，输出Parent End；

这些特点也可以通过write_category(self)函数获得；

接下来说明函数read_category(self)的原理：

- 首先读入目录文件中的所有数据；
- 逐行处理读入的数据：
 - 如果该行包括"Node Start"，将current_node_info清空，准备读入新节点的数据；
 - 如果是节点的其它信息，则对数据进行分割，获得key和value；
 - 如果该行包括"Node End"，说明一个节点的信息已经全部读入，创建该节点，并把它作为子节点添加到当前的父节点（栈顶元素）下，如果此时栈为空，说明该节点就是根节点。最后，将该节点入栈，作为后续节点的根节点；
 - 如果该行包括"Parent End"，说明当前父节点的所有子节点都已经处理完，弹出栈顶元素；
- 如果没有根节点信息，创建一个默认根节点；
- 设置目录的根节点；