项目说明文档

操作系统

——文件管理系统

作 者 姓 名： 崔宸睿

学 号： 2152614

指 导 教 师： 张惠娟

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

1 项目说明

本项目的开发技术为python3.9+PyQt，使用PyCharm Community Edition 2022.3.3作为开发工具。

启动项目需要在控制台中安装以下python库：

pip install PyQt5

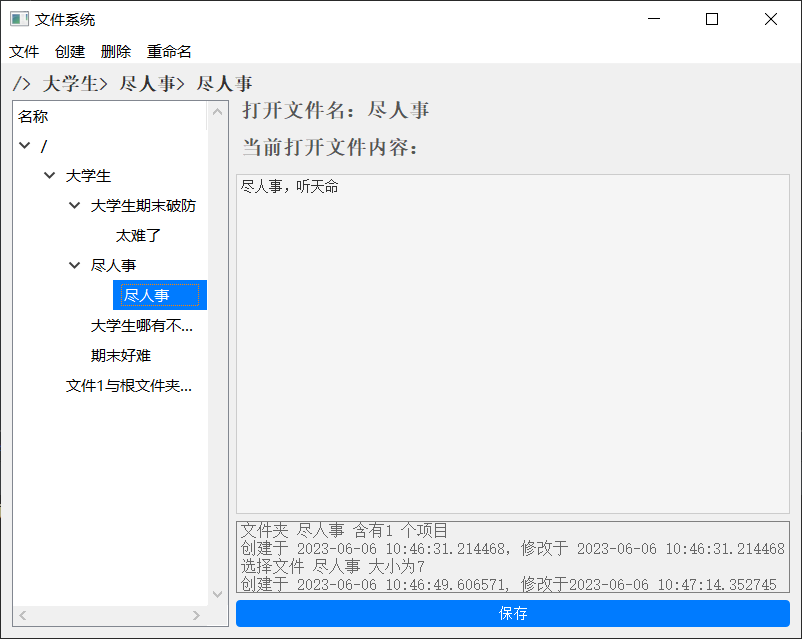
pip install qt-material

pip install bitarray

安装完成后运行文件FileSystemMainUI.py即可，打开界面后可以看到文件管理系统可视化界面。

也可以直接运行用pyInstaller打包后的文件FileSystem.exe。不过仍然建议配置好环境后在python环境中运行。

2 项目界面展示



项目界面的左侧为文件树，用户可以直接在左侧通过点击的方式选中相应文件夹或文件，若用户选中的是文件，则可以在右侧灰色大框中看到当前打开文件的内容，并且可以在右侧框中直接编辑，用户选中的文件和文件夹的属性也在下方显示，点击下方蓝色按钮可以保存文件内容。用户在左侧文件树中右击可以对选中的对象执行新建，重命名，删除等操作，也可以在选中某一对象后在上方菜单栏进行相关操作。操作演示视频请见[演示视频.mp4.](演示视频.mp4)

3 项目文件结构及重要类设计

项目主要由两个文件组成：

|  |  |
| --- | --- |
| 文件名 | 说明 |
| FileSystemMainUI.py | 项目启动文件，封装了页面和对应的操作 |
| FileSystemClass.py | 封装了文件系统的主要类 |

3.1文件控制块类：FCB

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| filename | 文件名 |
| creatTime | 创建时间 |
| modifyTime | 修改时间 |
| length | 文件长度，具体为文件中字符串的长度 |
| startAddress | 文件在模拟磁盘中的起始地址 |

3.2 FAT链表类：FAT

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| blockNum | 总块数 |
| table | 链接数组，记录每一个块在顺序上下一个块的链接地址 |

在我的文件系统中，采用链接结构来管理文件。为此专门开辟了一个数组table来记录存储各个块之间的链接。也就是说，第i块的下一块即为table[i]，同时，宏定义了FAT\_FREE和FAT\_END标记，分别在FAT表中代表某一块未被使用和某一块是一个文件内容的结束。

3.3 FAT空闲空间管理类：freeBlock

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| bitmap | 用位图的方式管理空闲空间，0代表某一块未被占用，1代表该块被占用，位图信息被存储在bitmap中。 |

3.4 文件夹节点类：folderNode

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| folderName | 文件夹名称 |
| createTime | 文件夹创建时间 |
| modifyTime | 文件夹修改时间 |
| folderChildren | 记录该文件夹中作为文件夹的孩子结点 |
| FCBChildren | 记录该文件夹中作为真实文件的孩子结点 |

与FCB类相对，FCB对应的是真实文件节点，而folderName记录的是文件夹节点，在folderName类中，folderChildren数组中全部为folderNode类对象，FCBChildren数组中全部为FCB类对象。该类中还有一个renewSize类方法，用于获得该文件夹节点当前的大小。

3.5文件树类：fileTree

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| root | 初始化整个文件树的根节点 |

3.6磁盘块类：hardDisk

初始化磁盘空间，建立BLOCK\_NUM个磁盘块，内容全部置空。

**3.7文件系统类：fileSystem**

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| \_fileTree | 文件树 |
| \_freeBlock | 文件系统空闲空间管理 |
| \_hardDisk | 磁盘块管理 |
| \_fat | 文件系统的FAT链表类 |

当用户第一次运行程序时，程序若未找到本地存储的文件LocalFile，则会自行创建一系列文件，形成一个文件树，若程序找到本地存储的文件LocalFile，则会通过pickle库读取LocalFile中存储的文件树信息，保证用户看到的是上次保存在本地磁盘中的文件。在文件系统关闭时，文件系统会询问用户是否将本次操作保存到磁盘，若选择“是”，则系统会调用类方法saveLocally，使用pickle库将文件保存到本地。

文件系统类中还封装了很多类方法，简要说明如下：

|  |  |
| --- | --- |
| 类方法名称 | 作用 |
| findFreeBlock | 寻找空闲块 |
| createDir | 在文件系统中建立文件目录 |
| createFCB | 在文件系统中某个目录下建立文件 |
| cleanDir | 清空文件夹内容，将文件夹所有子节点置空 |
| deleteFCB | 删除文件，并修改空闲空间标记 |
| deleteFolder | 彻底删除文件夹 |
| format | 格式化，将文件系统全部重置 |
| openFCB | 根据FCB中的起始地址在磁盘块中查找文件具体内容并返回 |
| writeFCB | 写文件，更新文件内容 |
| saveLocally | 将文件保存到本地 |

**3.8 UI界面显示类：FileSystemMainUI**

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| file\_system | 将前端界面和文件系统绑定 |
| curDir | 当前文件目录 |
| selectFile | 当前被用户选中的文件 |
| selectDir | 当前被用户选中的文件夹 |
| root | 伪根节点，用于在前端界面中模拟根节点 |

该类封装在文件FileSystemMainUI.py中，主要负责前端页面初始化和更新，并处理用户的操作。

界面显示类中还封装了很多类方法，简要说明如下：

|  |  |
| --- | --- |
| 类方法名称 | 作用 |
| setupUI | 绘制前端UI界面，初始化UI |
| buildQTreeModel | 构建适用于QTreeView的树模型，方便展示文件树 |
| clickItem | 鼠标点击左侧条目，展开选中的文件夹，打开所选的文件 |
| rightClickItem | 鼠标右击时间，弹出对文件或文件夹进行操作的菜单 |

其余类方法均是将前端用户的点击事件与后端文件系统相应的类方法绑定起来，比如删除，新建等功能，不再赘述。

4重要算法设计

4.1 写文件 filesystem: writeFCB函数

writeFCB函数有2个参数：content和fcb，分别代表待写入的内容和写入文件的fcb信息块。在我的文件系统设计中，一个块的大小为4字节，整个文件系统共有2^10个块，采用链接的方式管理文件的物理结构。

进入函数后，首先更新文件FCB的length，modifyTime等基本信息，然后调用findFreeBlock函数寻找空闲块nextBlockIndex，并更新文件起始地址；若未找到空闲块，则返回“没有足够空间”并退出。在找到空闲块后，在content中取一个文件块大小的内容，存入文件块self.\_hardDisk[nextBlockIndex]中，同时将存入文件块中的内容从content中删去，由于有新块被占用，所以要更新空闲块位图和FAT表。若content仍有内容，则继续执行上述算法，若content为空，则退出循环，写文件操作结束，代码如下：

def writeFCB(self, content, fcb:FCB):  
 fcb.length = len(content)  
 fcb.modifyTime = datetime.now()  
 curBlockIndex = FAT\_END  
 while content != "":  
 nextBlockIndex = self.findFreeBlock()  
 if nextBlockIndex == -1:  
 print("没有足够空间")  
 raise AssertionError("没有足够空间")  
 if curBlockIndex == FAT\_END:  
 fcb.startAddress = nextBlockIndex  
 else:  
 self.\_fat.table[curBlockIndex] = nextBlockIndex  
 self.\_hardDisk[nextBlockIndex] = content[:BLOCK\_SIZE]  
 content = content[BLOCK\_SIZE:]  
 self.\_freeBlock.bitmap[nextBlockIndex] = 1  
 curBlockIndex = nextBlockIndex  
 self.\_fat.table[curBlockIndex] = FAT\_END

**4.2删除操作 fileSystem: cleanDir, deleteFCB, recursiveDeleteFCB和deleteFolder函数**

删除操作共包含上述4个函数。第一个cleanDir函数负责清空文件夹内容，有一个传入参数，为待清空文件夹节点folderNode。清空文件夹节点需要清空FCBChildren数组内的所有文件节点和folderChildren内的所有文件夹，这就分别需要调用deleteFCB函数和cleanDir函数。

deleteFCB函数负责从fileSystem中删除具体的文件，传入参数为待删除文件节点fcb。需要将fcb文件对应的磁盘块中的内容全部删掉，这就需要利用FAT的table表。根据fcb中的起始地址对应的磁盘块ptr开始，将self.\_hardDisk[ptr]置空，然后将ptr=\_fat.table[ptr]将指针索引到下个磁盘块，再次执行清空操作，直到ptr为空。清除文件内容后，需要调用recursiveDeleteFCB函数将文件从父节点中删除，代码如下：

def deleteFCB(self, fcb : FCB):

ptr = fcb.startAddress

if ptr is not None:

while ptr != FAT\_END:

self.\_hardDisk[ptr] = ""

self.\_freeBlock.bitmap[ptr] = 0

nextPos = self.\_fat.table[ptr]

self.\_fat.table[ptr] = 0

ptr = nextPos

self.recursiveDeleteFCB(fcb, self.\_fileTree.root)

recursiveDeleteFCB函数有两个参数，分别是待删除文件节点fcb和遍历起点folder\_node.该函数从根节点开始，在每个文件夹节点中遍历FCBChildren数组，寻找待删除文件节点fcb，若找到，则删除节点fcb并返回，若未找到，则向深一层文件夹节点中遍历，直到搜索到待删除文件节点位置，代码如下：

def recursiveDeleteFCB(self, fcb:FCB, folder\_node:folderNode):  
 if fcb in folder\_node.FCBChildren:  
 folder\_node.FCBChildren.remove(fcb)  
 return  
 else:  
 for node in folder\_node.folderChildren:  
 self.recursiveDeleteFCB(fcb, node)

还有一个用于删除文件夹的函数deleteFolder，有一个参数dir用于记录待删除的文件夹节点。首先调用cleanDir函数清除文件夹 dir 及其子文件夹的内容，然后使用队列辅助广度优先搜索，从根节点开始搜索dir所在的子文件夹列表，搜索到后将dir从相应的子文件夹列表中移除，删除结束。代码如下：

def deleteFolder(self, dir:folderNode):

self.cleanDir(dir)

ptr = self.\_fileTree.root

q = [ptr]

while dir not in ptr.folderChildren and q!=[]:

ptr = q.pop(0)

q.extend(ptr.folderChildren)

ptr.folderChildren.remove(dir)

5 总结和展望

通过这次项目，我对文件系统有了更加深刻的了解，对文件的目录结构和文件的物理结构有了新的认识，同时，也锻炼了我python的代码能力，进一步强化了我对PyQt的使用能力。在这次项目中，为了能将模拟文件系统的数据存到磁盘中，我了解到了可以使用python的pickle库，它能将python中几乎所有的数据类型（列表，字典，集合，类等）序列化，这为我把文件系统的数据存到磁盘中起到了非常大的帮助。