操作系统第三次作业-文件管理系统

1. 介绍

本次作业是在内存中模拟实现一个文件管理系统，退出此文件系统后，会将该文件系统的内容保存到磁盘中，下次使用此文件系统时可以将上次的内容恢复到内存中去。

本项目实现了一个基本完整的文件系统，实现的功能点有：

- 格式化
- 新建子目录
- 删除子目录
- 显示当前目录
- 新建文件
- 删除文件
- 更改当前目录
- 打开文件
- 关闭文件
- 写文件
- 读文件
- 文件与目录重命名
- 统计文件大小

同时提供了一个简洁的用户交互界面，以及一个简单的文本编辑器可以对文件进行查看与编辑操作。

开发环境

- 语言：`python3`
- 界面：`PyQt5`
- 操作系统：`Kubuntu 23.04`

3. 系统设计

3.1 文件系统内核

3.1.1 块类设计

本文件系统对空间进行分块管理，一块的大小为4kB,块类设计如下：

```
class Block:
    def __init__(self):
        self.block_size = 1024*4
        self.data = bytearray(self.block_size)

    def write(self, data: bytearray):
        self.data = data

    def read(self) -> bytearray:
        return self.data
```

3.1.2 Inode类设计

`Inode`类保存每个文件相关信息，例如单个文件块索引位置，文件创建修改与访问时间，文件大小等信息。设计如下：

```python
class Inode:
    def __init__(self):
        self.file_size = 0
        self.file_blocks_index = []
        self.ctime = datetime.now()
        self.mtime = datetime.now()
        self.atime = datetime.now()

    def add_block(self, block_index: int):
        self.file_blocks_index.append(block_index)

    def remove_block(self, block_index: int):
        self.file_blocks_index.remove(block_index)
```

3.1.3 文件类设计

文件类包含一个对应的 `Inode` 对象和文件名以及类型标识符，具有 `read`，`write`，与 `clear` 方法，`read`，`write` 方法读写对应文件内容，`clear` 方法情空文件内容，同时标记位图释放空闲块。

```python
class File:
    def __init__(self, name):
        self.name = name
        self.inode = Inode()
        self.type = "file"

    def read(self, fs: FileSystem) -> bytearray:
        data = bytearray()
        self.inode.atime = datetime.now()
        for block_index in self.inode.file_blocks_index:
            data += fs.space[block_index].read()
        return data
```

```
13
14      def write(self, data: bytearray, fs: FileSystem) -
    > bool:
15          valid_block_nums = fs.get_valid_block_nums()
16          block_count = len(data) // (1024*4) + 1
17          if block_count > valid_block_nums:
18              print("No more space available")
19              return False
20          self.clear(fs)
21          self.inode.file_size = len(data)
22          fs.used_size += self.inode.file_size
23          for i in range(block_count):
24              j = 0
25              for j in range(fs.file_block_nums):
26                  if fs.valid_blocks[j] == 0:
27                      fs.valid_blocks[j] = 1
28                      block = fs.space[j]
29                      if i == 0:
30                          self.inode.mtime =
    datetime.now()
31                          self.inode.atime =
    datetime.now()
32                      block.write(
33                          data[i * 1024*4: min((i + 1) *
    1024*4, len(data))])
34                      self.inode.add_block(j)
35                      break
36              if j == fs.file_block_nums-1:
37                  print("No more space available")
38                  return False
39          return True
40
41      def clear(self, fs: FileSystem):
42          """释放文件占用block
43          """
44          fs.used_size -= self.inode.file_size
```

```
            self.inode.ctime = datetime.now()
            self.inode.mtime = datetime.now()
            self.inode.atime = datetime.now()
            for i in self.inode.file_blocks_index:
                fs.valid_blocks[i] = 0
            self.inode.file_blocks_index = []
```

3.1.4 目录类设计

目录类包含目录名，子目录列表，子文件列表，父目录等相关信息，提供添加删除文件，添加删除子目录，列出目录内容等方法。

```
class Directory:
    def __init__(self, name, parent):
        self.name = name
        self.parent = parent
        self.files = []
        self.subdirectories = []
        self.type = "directory"

    def add_file(self, file):
        self.files.append(file)

    def remove_file(self, file, fs: FileSystem):
        file.clear(fs)
        self.files.remove(file)

    def get_file(self, name):
        for file in self.files:
            if file.name == name:
                return file

    def add_subdirectory(self, directory):
        self.subdirectories.append(directory)

```

```
    def remove_subdirectory(self, directory, fs: FileSystem):
        if directory in self.subdirectories:
            for file in directory.files:
                directory.remove_file(file, fs)
            self.subdirectories.remove(directory)
            directory.parent = None
            directory.remove_all_subdirectories(fs)

    def remove_all_subdirectories(self, fs: FileSystem):
        for directory in self.subdirectories:
            for file in directory.files:
                directory.remove_file(file, fs)
            directory.parent = None
            directory.remove_all_subdirectories(fs)
        self.subdirectories = []

    def get_subdirectory(self, name):
        for directory in self.subdirectories:
            if directory.name == name:
                return directory

    def list_contents(self):
        dir_contents = []
        file_contents = []
        for directory in self.subdirectories:
            dir_contents.append(directory)
        for file in self.files:
            file_contents.append(file)
        return dir_contents, file_contents
```

3.1.5 文件系统类设计

文件系统类对外层提供所有文件系统的相关接口，同时存储当前所处目录位置，文件系统整体空间，以及位图管理文件系统空闲空间。

```
class FileSystem:
    def __init__(self):
        self.root = Directory("/", None)
        self.current_directory = self.root
        self.file_block_nums = 2560*4  # 块数量
        self.valid_blocks = bytearray(
            self.file_block_nums)    # 位图管理空闲空间，0表示空闲，1表示已使用
        self.space = [Block() for _ in range(self.file_block_nums)]  # 文件系统整体空间
        self.used_size = 0

    def create_file(self, name):
        for file in self.current_directory.files:
            if file.name == name:
                print("File already exists")
                return False
        file = File(name)
        self.current_directory.add_file(file)
        return True

    def delete_file(self, name):
        file = self.current_directory.get_file(name)
        if file:
            self.current_directory.remove_file(file, self)
            return True
        else:
            print("File not found")
            return False
```

```
28
29      def read_file(self, name):
30          file = self.current_directory.get_file(name)
31          if file:
32              return file.read(self)
33          else:
34              return bytearray()
35
36      def write_file(self, name, data):
37          file = self.current_directory.get_file(name)
38          if file:
39              return file.write(data, self)
40          else:
41              return False
42
43      def list_directory(self):
44          return self.current_directory.list_contents()
45
46      def change_directory(self, name):
47          if name == "..":
48              parent = self.current_directory.parent
49              if parent:
50                  self.current_directory = parent
51              return
52          for dir in
    self.current_directory.subdirectories:
53              if dir.name == name:
54                  self.current_directory = dir
55                  return
56          directory = self.find_directory(self.root,
    name)
57          if directory:
58              self.current_directory = directory
59
60      def find_directory(self, directory, name):
61          if directory.name == name:
```

```
62                 return directory
63
64             for subdirectory in directory.subdirectories:
65                 result =
    self.find_directory(subdirectory, name)
66                 if result:
67                     return result
68
69             return None
70
71         def make_directory(self, name):
72             for dir in
    self.current_directory.subdirectories:
73                 if dir.name == name:
74                     print("Directory already exists")
75                     return False
76             directory = Directory(name,
    self.current_directory)
77
     self.current_directory.add_subdirectory(directory)
78             return True
79
80         def remove_directory(self, name):
81             for subdirectory in
    self.current_directory.subdirectories:
82                 if name == subdirectory.name:
83
     self.current_directory.remove_subdirectory(subdirect
    ory, self)
84                     return True
85             directory = self.find_directory(self.root,
    name)
86             parent = directory.parent
87             if directory:
88                 parent.remove_subdirectory(directory,
    self)
```

```
89                 return True
90             else:
91                 print("Directory not found")
92                 return False
93 
94         def save_to_disk(self, filename):
95             with open(filename, "wb") as f:
96                 pickle.dump(self, f)
97 
98         def get_current_path(self):
99             # 获取当前目录的路径
100            if self.current_directory.name == "/":
101                return "/"
102            path = ""
103            directory = self.current_directory
104            while directory.name != "/":
105                path = "/"+directory.name + path
106                directory = directory.parent
107            return path
108 
109        def get_file_size(self, file):
110            return file.inode.file_size
111 
112        def get_file_mtime(self, file):
113            return file.inode.mtime
114 
115        def get_dir_item_nums(self, directory):
116            return len(directory.files) +
    len(directory.subdirectories)
117 
118        def rename_file(self, old_name, new_name):
119            """重命名文件
120            Returns:
121                bool: 是否成功
122                int: 错误码, 0表示成功, 1表示文件不存在, 2表示
    新文件名与旧文件名相同, 3表示新文件名已存在, 4表示新文件名为空
```

```
123             """
124             if new_name is None:
125                 return False, 4
126             if new_name == old_name:
127                 return False, 2
128             for file in self.current_directory.files:
129                 if file.name == new_name:
130                     return False, 3
131             file =
    self.current_directory.get_file(old_name)
132             if file:
133                 file.name = new_name
134                 return True, 0
135             else:
136                 return False, 1
137 
138         def rename_directory(self, old_name, new_name):
139             """重命名目录
140             Returns:
141                 bool: 是否成功
142                 int: 错误码, 0表示成功, 1表示目录不存在, 2表示
    新目录名与旧目录名相同, 3表示新目录名已存在, 4表示新目录名为空
143             """
144             if new_name is None:
145                 return False, 4
146             if new_name == old_name:
147                 return False, 2
148             if new_name in [dir.name for dir in
    self.current_directory.subdirectories]:
149                 return False, 3
150             directory =
    self.current_directory.get_subdirectory(old_name)
151             if directory:
152                 directory.name = new_name
153                 return True, 0
154             else:
```

```
155                 return False, 1
156
157         def fformat(self):
158             for file in self.root.files:
159                 self.root.remove_file(file, self)
160             self.root.remove_all_subdirectories(self)
161
162         def get_total_and_used_space_size(self):
163             return self.file_block_nums*1024*4,
        self.used_size
164
165         def get_valid_block_nums(self) -> int:
166             result = 0
167             for block in self.valid_blocks:
168                 if block == 0:
169                     result += 1
170             return result
```

3.1.6 文件存储空间管理

文件存储空间管理采取索引表的方式，通过对文件系统空间分块读取，对块索引，来进行文件的读写，相关操作函数如下：

- `write`

```
1   def write(self, data: bytearray, fs: FileSystem) ->
    bool:
2       valid_block_nums = fs.get_valid_block_nums()
3       block_count = len(data) // (1024*4) + 1
4       if block_count > valid_block_nums:
5           print("No more space available")
6           return False
7       self.clear(fs)
8       self.inode.file_size = len(data)
9       fs.used_size += self.inode.file_size
```

```
    for i in range(block_count):
        j = 0
        for j in range(fs.file_block_nums):
            if fs.valid_blocks[j] == 0:
                fs.valid_blocks[j] = 1
                block = fs.space[j]
                if i == 0:
                    self.inode.mtime = datetime.now()
                    self.inode.atime = datetime.now()
                    block.write(
                        data[i * 1024*4: min((i + 1) * 1024*4, len(data))])
                    self.inode.add_block(j)
                    break
                    if j == fs.file_block_nums-1:
                        print("No more space available")
                        return False
                    return True
```

- `read`

```
def read(self, fs: FileSystem) -> bytearray:
    data = bytearray()
    self.inode.atime = datetime.now()
    for block_index in self.inode.file_blocks_index:
        data += fs.space[block_index].read()
        return data
```

3.1.7 空闲空间管理

空闲空间管理采用位图标记，0表示块空闲，1表示块占用，代码如下：

```
self.valid_blocks = bytearray(
            self.file_block_nums)     # 位图管理空闲空间，0表示空闲，1表示已使用
```

当对文件进行写入时，会首先计算该文件需要占用的块数，然后遍历块位图，找到空闲块进行写入，

```
for i in range(block_count):
    j = 0
    for j in range(fs.file_block_nums):
        if fs.valid_blocks[j] == 0:
            fs.valid_blocks[j] = 1
            block = fs.space[j]
            if i == 0:
                self.inode.mtime = datetime.now()
                self.inode.atime = datetime.now()
                block.write(
                    data[i * 1024*4: min((i + 1) * 1024*4, len(data))])
                self.inode.add_block(j)
                break
                if j == fs.file_block_nums-1:
                    print("No more space available")
                    return False
```

文件删除时，也会首先释放块的占用，更改位图：

```python
def clear(self, fs: FileSystem):
    """释放文件占用block
        """
    fs.used_size -= self.inode.file_size
    self.inode.ctime = datetime.now()
    self.inode.mtime = datetime.now()
    self.inode.atime = datetime.now()
    for i in self.inode.file_blocks_index:
        fs.valid_blocks[i] = 0
        self.inode.file_blocks_index = []
```

3.1.8 文件目录管理

文件目录采用多级目录结构，可以在不同级目录间进行切换，目录项中记录了文件名，文件大小以及文件索引地址等信息。

```python
class Inode:
    def __init__(self):
        self.file_size = 0
        self.file_blocks_index = []
        self.ctime = datetime.now()
        self.mtime = datetime.now()
        self.atime = datetime.now()

    def add_block(self, block_index: int):
        self.file_blocks_index.append(block_index)

    def remove_block(self, block_index: int):
        self.file_blocks_index.remove(block_index)
```

```
class Directory:
    def __init__(self, name, parent):
        self.name = name
        self.parent = parent
        self.files = []
        self.subdirectories = []
        self.type = "directory"
```

3.1.9 磁盘保存

由于本文件系统是在内存中开辟一块空间，当退出文件系统后，会调用`python`的`pickle`模块将文件系统对象序列化为`fs.pickle`文件保存到磁盘，当下次打开本文件系统时，将会从本地的`fs.pickle`文件读取文件系统对象，实现保留上一次使用的内容。

```
def save_to_disk(self, filename):
    with open(filename, "wb") as f:
        pickle.dump(self, f)
```

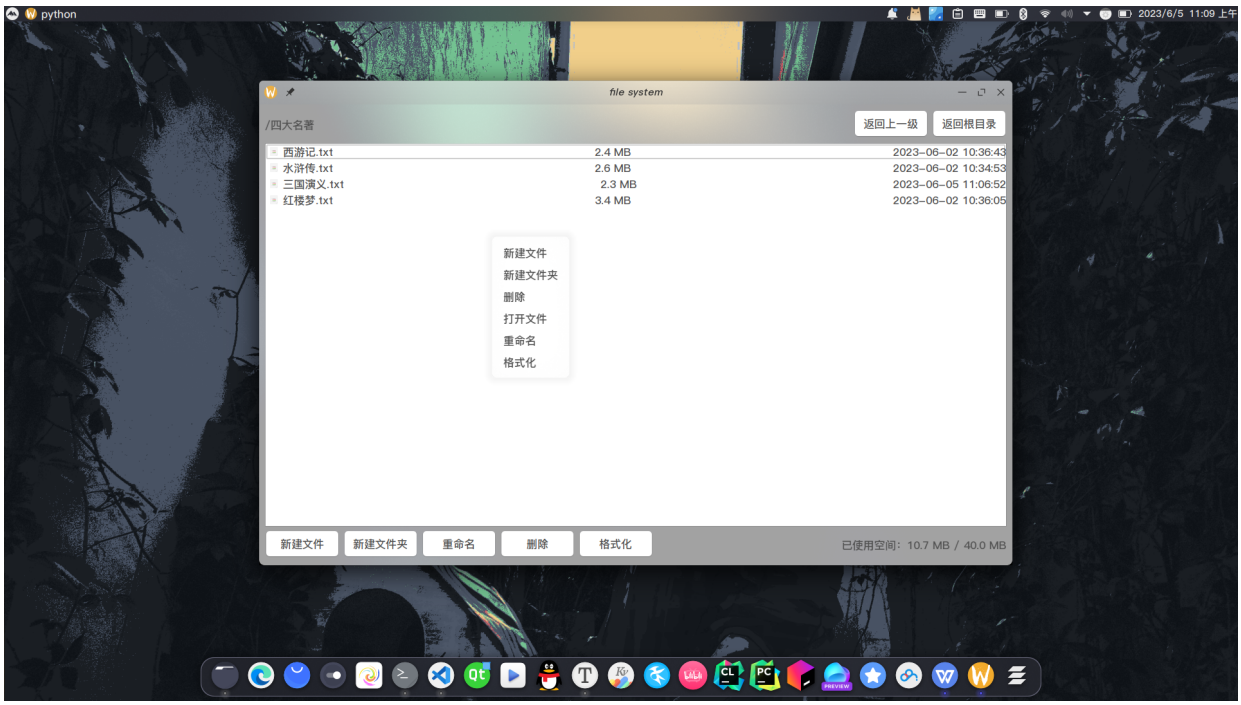
```
def load_from_disk(filename):
    with open(filename, "rb") as f:
        return pickle.load(f)
```

3.2 界面设计

3.2.1 界面展示

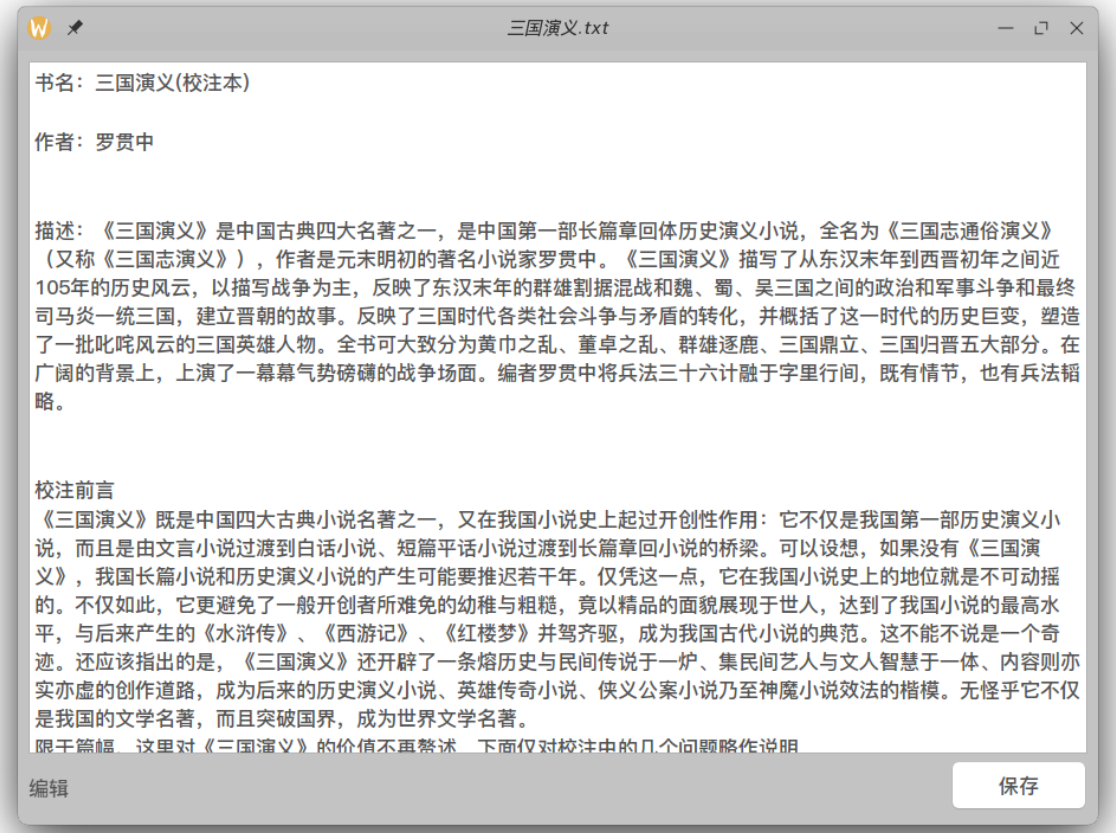
- 主界面
  1. 正常界面

2. 右键菜单

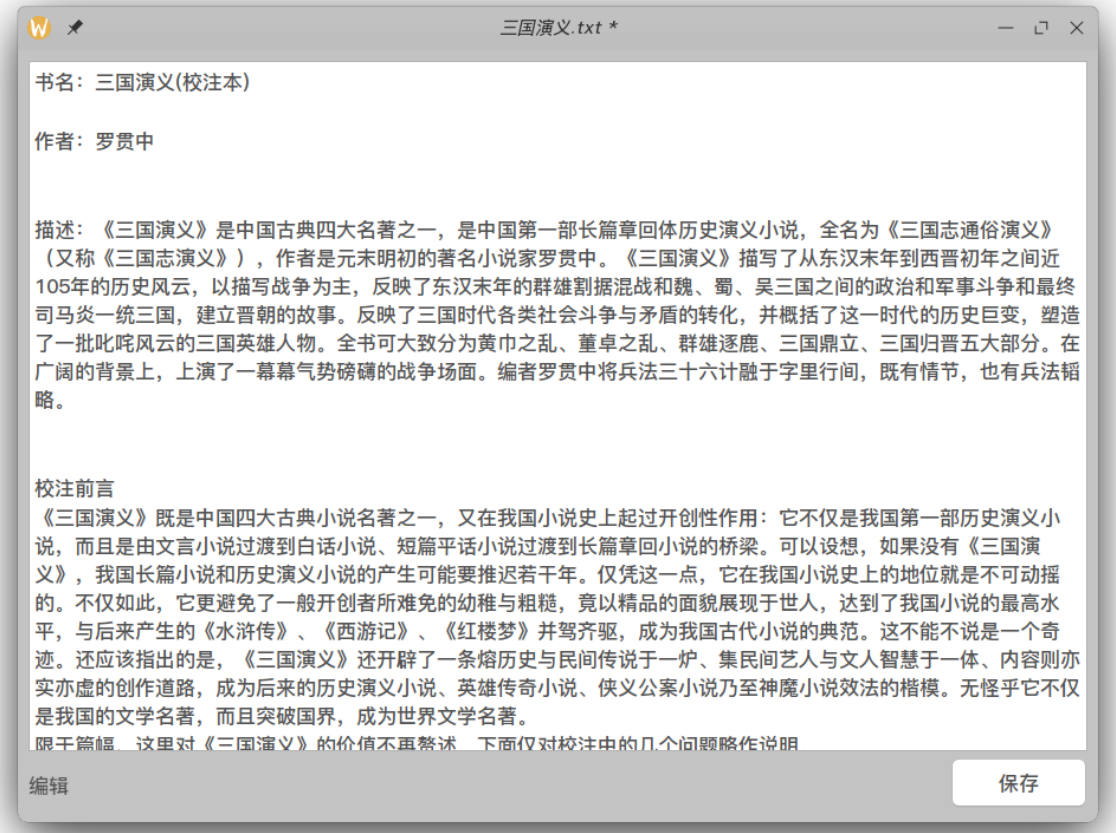


- 编辑器界面

1. 正常界面



2. 做出修改未保存，标题栏显示“*”号



3.3.2 功能提供

本UI界面提供以下几点功能：

- 新建文件
- 新建文件夹
- 重命名文件或文件夹
- 删除文件或文件夹
- 格式化文件系统
- 文件编辑(支持快捷键 `Ctrl+s` 保存)

以上功能可以通过按钮点击也可通过右键菜单实现。