操作系统——文件管理系统

1753837 陈柄畅

介绍

本项目为操作系统课程第三次作业，实现了简易的FAT文件管理系统。使用MDL作为前端库，使用Flask作为后端。

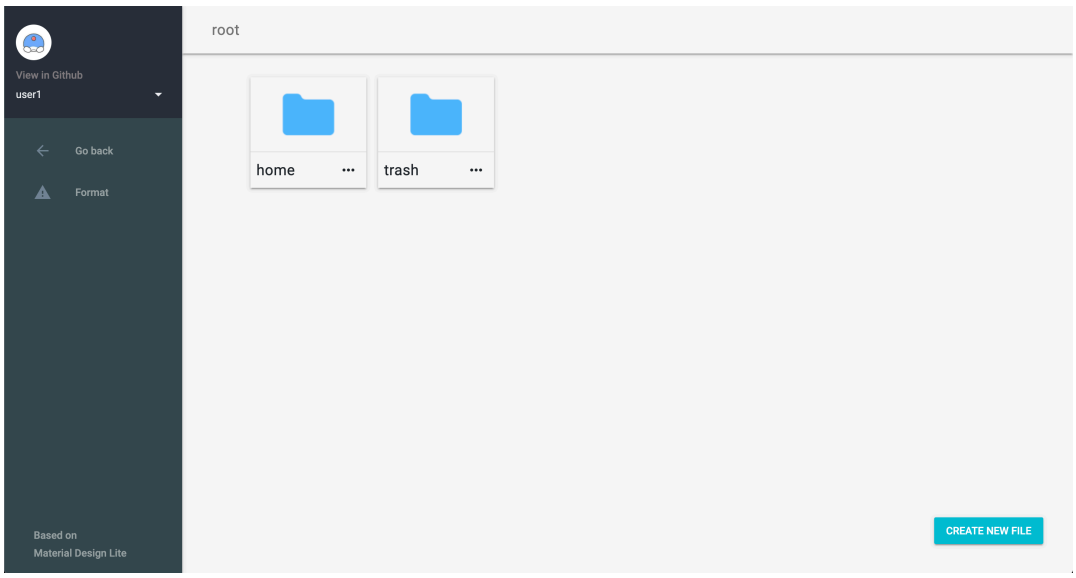
项目要求

在内存中开辟一个空间作为文件存储器，在其上实现一个简单的文件系统。退出这个文件系统时，需要该文件系统的内容保存到磁盘上，以便下次可以将其恢复到内存中来。

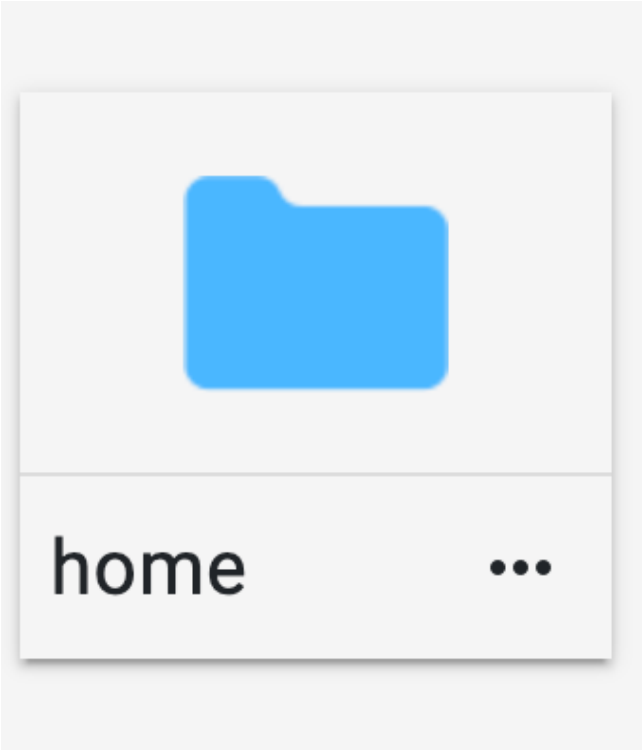
运行

1. 安装`Flask`
2. 命令行输入`python flask.py`
3. 使用浏览器访问`127.0.0.1:5000`

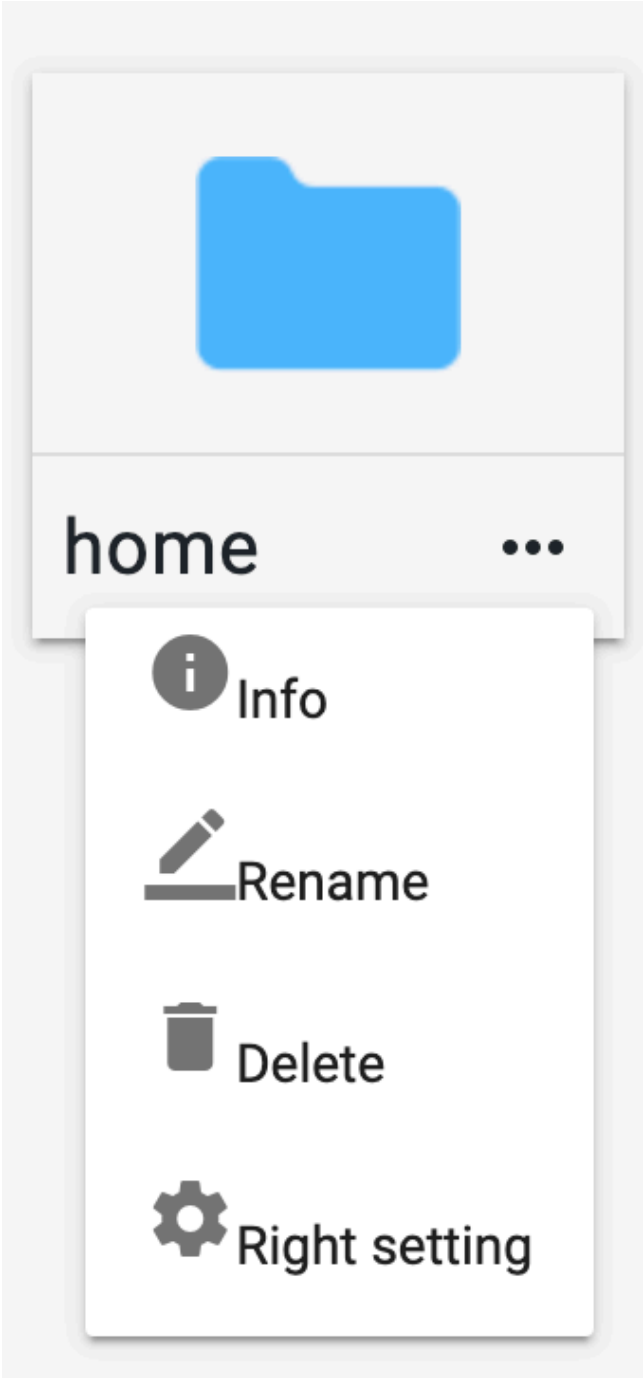
界面



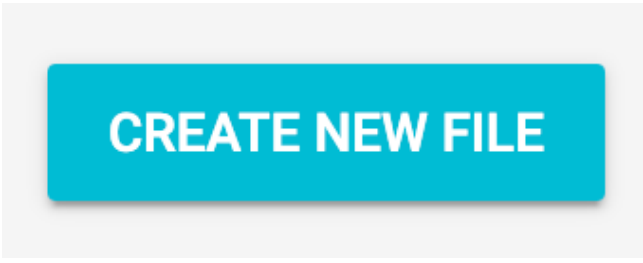
- 文件/文件夹



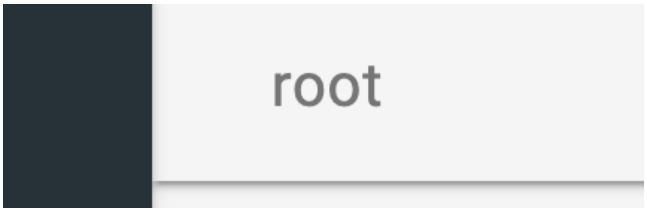
- 文件/文件夹操作



- 创建文件

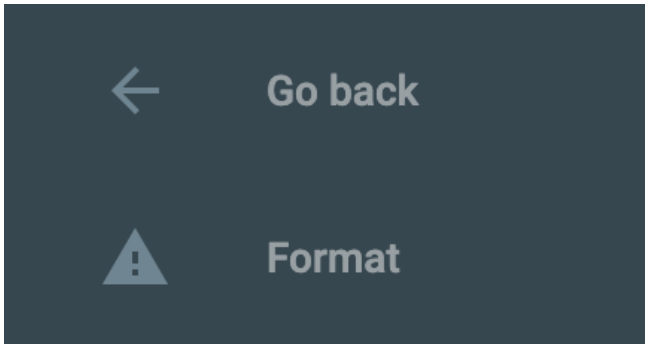


- 当前路径



- 用户切换

- 系统操作



系统概述

本项目的文件系统首先会从指定位置读取之前保存的磁盘文件(以disk.txt形式保存)，如果没有，将会初始化磁盘。接着，系统会将磁盘内容加载到内存，在内存中模拟真正的文件系统。默认的磁盘共有2048个Block，每个Block为128位(其中以字符位模拟真正磁盘系统中的二进制位)。共有512个FCB。同时，本项目使用前后端分离的架构，来模拟操作系统的用户界面与文件系统的交互。

数据结构

- FAT

  将空闲区位图与FAT表合并形成FAT类。如果当前4位为'0000'，表示空闲。如果当前4位为'——'，表示文件结束符。否则，当前4位指向下个数据块。

```
class FAT:
    def __init__(self, bitmap):
        self.bitmap = bitmap

    def getFirstFreeBlock(self):
        for i in range(len(self.bitmap) // 4):
            if self.bitmap[i * 4:i * 4 + 4] == '0000':
                return i
        return -1

    def updateBitmap(self, pos, content):
        content = '0'*(4-len(content))+content
        if pos * 4 + 4 == len(self.bitmap):
            self.bitmap = self.bitmap[0:pos*4]+content
        else:
            self.bitmap =
self.bitmap[0:pos*4]+content+self.bitmap[pos*4+4:]
```

```
    def deleteFile(self, pos):
        if pos == '-1':
            return 0
        count = 0
        while self.bitmap[pos*4:pos*4+4] != '----':
            pos = self.bitmap[pos*4:pos*4+4]
            self.updateBitmap(pos, '0000')
            count += 1
        self.updateBitmap(pos, '0000')
        return count + 1
```

- SuperBlock

  记录磁盘总数据块数，块大小，空闲数据块数，空闲FCB数。

```
class SuperBlock:
    def __init__(self, file: str):
        attr_list = file.split(" ")
        self.all_block = eval(attr_list[0])
        self.block_size = eval(attr_list[1])
        self.free_block = eval(attr_list[2])
        self.free_fcb = eval(attr_list[3])

    def writeToDisk(self):
        return str(self.all_block) + " " + str(self.block_size) + " " +
str(self.free_block) + " " + str(self.free_fcb)
```

- Dictionary

  目录表。记录FCB和FCB空闲位图。

```
class Dictionary:
    def __init__(self, fcb_bitmap, fcb_list):
        self.fcb_bitmap = fcb_bitmap
        self.fcb_list = fcb_list

    def getAndSetFirstFreeFCB(self):
        for i in range(len(self.fcb_bitmap)):
            if self.fcb_bitmap[i] == '0':
                self.updateBitmap(i,'1')
                return i
        return -1

    def updateBitmap(self, pos, value):
        if pos == len(self.fcb_bitmap) - 1:
            self.fcb_bitmap = self.fcb_bitmap[:pos] + value
        else:
```

```
            self.fcb_bitmap = self.fcb_bitmap[:pos] + value +
self.fcb_bitmap[pos + 1:]
```

- FCB

  文件控制块。记录了文件的名称，类型，位置，创建时间，修改时间，大小，第一个数据块地址，写权限用户，读权限用户，是否可以被删除，是否有效和子目录项地址。

```
class FCB:
    def __init__(self, name, file_type, pos, create_time, update_time,
size, first_block, write_user,
                 read_user, delete_able, is_able):
        self.name = name
        self.type = file_type
        self.pos = pos
        self.creat_time = create_time
        self.update_time = update_time.strip()
        self.size = size
        self.first_block = first_block
        self.write_user = write_user
        self.read_user = read_user
        self.delete_able = delete_able
        self.is_able = is_able
        self.child_list = []

    def writeToDiskFirstLine(self):
        if self.type == '-1':
            return '-1\n'
        return self.name + ' ' + self.type + ' ' + str(self.pos) + ' ' +
self.creat_time + ' ' + self.update_time + '\n'

    def writeToDiskSecondLine(self):
        if self.type == '-1':
            return '-1\n'
        return str(self.size) + ' ' + self.first_block + ' ' +
self.write_user + ' ' + self.read_user + ' ' + str(
            self.delete_able) + ' ' + str(self.is_able) + '\n'

    def checkWriteRight(self, user):
        if user in self.write_user.split('_'):
            return True
        else:
            return False

    def checkReadRight(self, user):
        if user in self.read_user.split('_'):
            return True
        else:
```

```
            return Falsec
```

实现

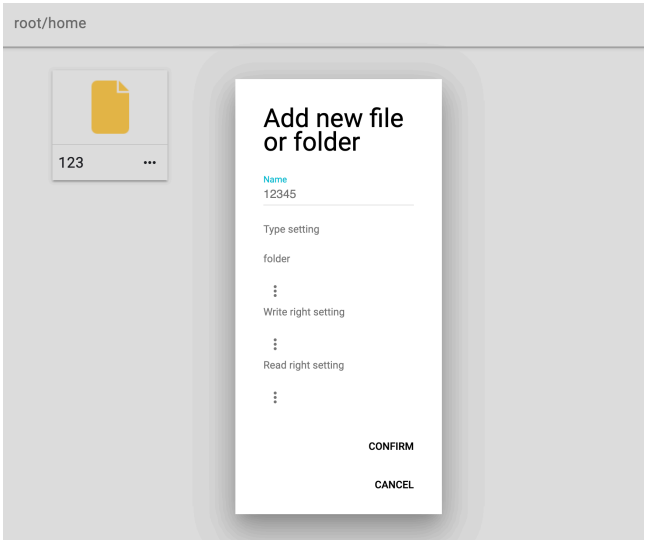
前端所有逻辑仅仅为实现GUI和与后端通信，与文件系统实现关系较弱，故不作赘述。且实现中出现的情况过多，此处就不一一截图演示。

- 格式化

  删除磁盘文件，重新初始化。

```
    def reformat(self):
        os.remove('disk.txt')
        self.initFromFile()
```

- 创建子目录，文件



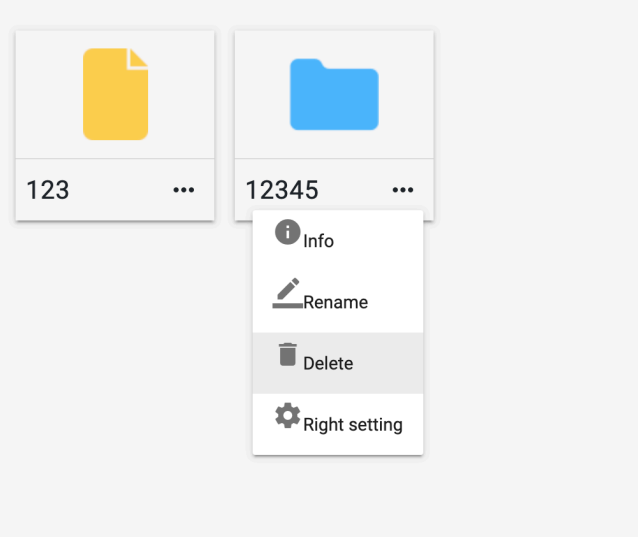
创建子目录与创建文件的方法相似，只是一个参数的不同。首先通过目录得到空闲FCB的位置，接着定位到相应的文件夹，并检查是否有同名冲突。如果没有，则更新目录，超级块，并依次向上传递更新父节点的大小。

```
@app.route('/files/<path>/<name>/<file_type>/<writer_user>/<read_user>',
methods=['POST'])
def createFile(path, name, file_type, writer_user, read_user):
    read_user = '_'.join(set(writer_user.split('_') +
read_user.split('_')))
    return str(file_system.createFile(path, name, file_type, writer_user,
read_user))
```

```
    def createFile(self, path, name, file_type, writer_user, read_user):
        fcb_address = self.dictionary.getAndSetFirstFreeFCB()
        if fcb_address < 0:
```

```
            return -2
        path_list = path.split("_")
        current = self.locateFolder(path_list)
        for child in self.dictionary.fcb_list[current].child_list:
            if self.dictionary.fcb_list[child].name == name and
self.dictionary.fcb_list[child].type == file_type:
                return -1
        fcb = FCB(name, file_type, current,
datetime.datetime.strftime(datetime.datetime.now(), '%Y-%m-%d-%X'),
                  datetime.datetime.strftime(datetime.datetime.now(), '%Y-
%m-%d-%X'), 0, '-1', writer_user, read_user,
                  1, 1)
        self.super_block.free_fcb -= 1
        self.dictionary.fcb_list[fcb_address] = fcb
        self.dictionary.fcb_list[current].child_list.append(fcb_address)
        self.updateParentSize(fcb_address, 0)
        return 1
```

- 删除子目录，文件



删除子目录与删除文件方法类似。当删除一个子目录时，文件系统会递归地调用删除子节点函数。由于文件的子节点列表为空，所以不用特殊处理，两者完全可以执行一样的函数。

```
@app.route('/delete/<path>/<user>/<file_type>', methods=['DELETE'])
def deleteFile(path, user, file_type):
    return file_system.deleteFile(path, user, file_type)
```

```
    def deleteFile(self, path, user, file_type):
        if self.getCurFile(path, file_type).checkWriteRight(user) and
self.getCurFile(path, file_type).delete_able == 1:
            loc = self.getCurFileLocation(path)
            self.updateParentSize(loc, self.dictionary.fcb_list[loc].size)
            self.deleteChild(loc)
            return "1"
        else:
            return "-1"
```

```
    def deleteChild(self, loc):
        for child in self.dictionary.fcb_list[loc].child_list:
            self.deleteChild(child)
        if self.dictionary.fcb_list[loc].delete_able == 1:
            self.dictionary.updateBitmap(loc, '0')
            # Only set it unable rather than directly delete it!
            self.dictionary.fcb_list[loc].is_able = '0'
            self.super_block.free_block +=
self.fat.deleteFile(self.dictionary.fcb_list[loc].first_block)
            parent =
self.dictionary.fcb_list[self.dictionary.fcb_list[loc].pos]
            parent.child_list.remove(loc)
            self.super_block.free_fcb += 1
```

- 显示目录

  当前端调用后端获取当前文件接口后，后端返回新目录下的所有文件。前端使用JavaScript动态增加删除HTML的DOM，显示目录。当用户没有读权限时，GUI中将不会展示文件或者文件夹。

```
@app.route('/files/<path>/<user>', methods=['GET'])
def getCurFiles(path, user):
    fcb_list = file_system.getFileChildren(path)
    result = {}
    for index, fcb in enumerate(fcb_list):
        if file_system.dictionary.fcb_list[fcb].checkReadRight(user):
            result[index] = {
                'name': file_system.dictionary.fcb_list[fcb].name,
                'type': file_system.dictionary.fcb_list[fcb].type
            }
    return jsonify(result)
```

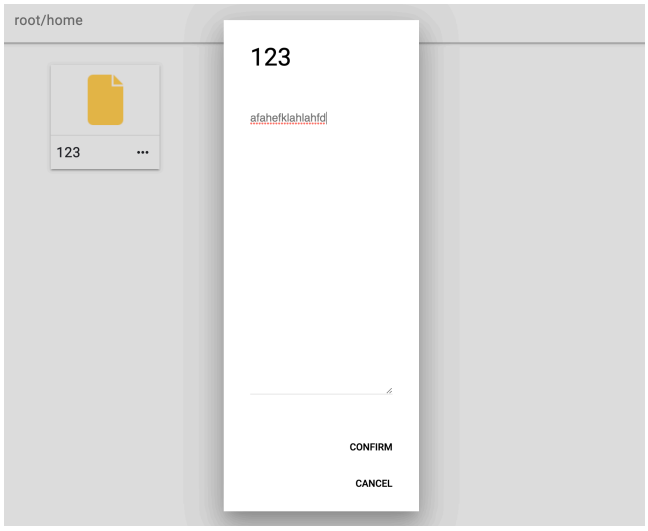
  前端修改逻辑代码过长，就不在这里展示。详见`static/js/main.js 47-201`

- 更改当前目录

  GUI共提供两种方法更改当前目录。一种是直接点击文件夹图标。前端将会将新的路径发送到后端，获取所有新目录下的文件。另一种是点击左侧边栏的返回按钮，前端将会将父节点路径发送到后端。但当用户已经处于root目录下时，就无法进行返回操作。

  代码部分与显示目录相同。

- 打开文件，读文件



用户点击文件图标后，前端将发送文件的路径及名称到后端。值得注意的是，本项目额外实现了用户权限的管理。后端，即模拟文件系统，第一时间会检查用户是否拥有读写权限。根据用户不同的权限，后端将返回不同的状态码。前端将根据不同的状态码，显示不同的界面。

```
@app.route('/content/<path>/<user>', methods=['GET'])
def getFileContent(path, user):
    return jsonify(file_system.openFileContent(path, user))
```

```
    def openFileContent(self, path, user):
        fcb = self.getCurFile(path)
        if not fcb.checkReadRight(user):
            return {'state': -1, 'content': ''}
        current = fcb.first_block
        content = ''
        if current != '-1':
            while current != "----":
                content += self.data[evalPro(current)]
                current = self.fat.bitmap[evalPro(current) *
4:evalPro(current) * 4 + 4]
        if fcb.checkWriteRight(user):
            return {'state': 2, 'content': content}
        else:
            return {'state': 1, 'content': content}
```

- 写文件

如果用户有写权限的话，点击文件图标，用户将进入文件写界面。用户选择保存文件后，文件系统将重新根据文件的新的大小重新分配数据块，同时相应地更新父节点信息，空闲位图和FAT。

```
@app.route('/content/<path>/<content>/<user>', methods=['UPDATE'])
def updateFileContent(path, content, user):
    return file_system.updateFileContent(path, content, user)
```

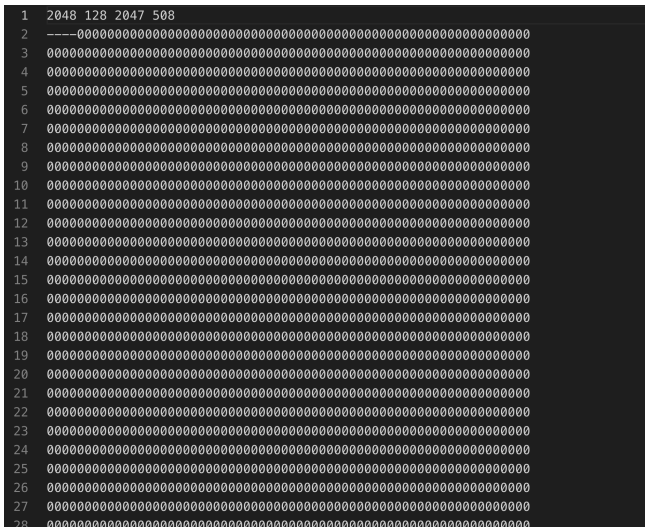
```
    def updateFileContent(self, path, content, user):
        fcb = self.getCurFile(path)
        if not fcb.checkWriteRight(user):
            return '-1'
        pre_size = fcb.size
        current = fcb.first_block
        if current != '-1':
            while self.fat.bitmap[evalPro(current) * 4:evalPro(current) *
4 + 4] != '----':
                temp = self.fat.bitmap[evalPro(current) *
4:evalPro(current) * 4 + 4]
                self.fat.updateBitmap(evalPro(current), '0000')
                current = temp
                self.super_block.free_block += 1
            self.fat.updateBitmap(evalPro(current), '0000')
            self.super_block.free_block += 1
        fcb.size = 0
        if self.super_block.free_block <= 0:
            print(len(self.fat.bitmap))
            return '-2'
        fcb.first_block = str(self.fat.getFirstFreeBlock())
        if len(content) > self.super_block.block_size:
            if self.super_block.free_block <= 0:
                fcb.update_time =
datetime.datetime.strftime(datetime.datetime.now(), '%Y-%m-%d-%X')
                print(len(self.fat.bitmap))
                return '-2'
            add_loc = self.fat.getFirstFreeBlock()
            self.data[add_loc] = content[0:self.super_block.block_size]
            self.fat.updateBitmap(add_loc, '----')
            content = content[self.super_block.block_size:]
            self.super_block.free_block -= 1
            fcb.size = 1
            while len(content) > 0:
                if self.super_block.free_block <= 0:
                    fcb.update_time =
datetime.datetime.strftime(datetime.datetime.now(), '%Y-%m-%d-%X')
                    print(len(self.fat.bitmap))
                    return '-2'
                self.fat.updateBitmap(add_loc,
str(self.fat.getFirstFreeBlock()))
                add_loc = self.fat.getFirstFreeBlock()
                if add_loc == 2047:
                    print('2047')
                self.data[add_loc] =
content[0:min(self.super_block.block_size, len(content))]
                self.fat.updateBitmap(add_loc, '----')
                self.super_block.free_block -= 1
                fcb.size += 1
```

```
                if len(content) > self.super_block.block_size:
                    content = content[self.super_block.block_size:]
                else:
                    content = ''
            self.updateParentSize(self.getCurFileLocation(path), fcb.size
- pre_size)
        else:
            self.data[evalPro(fcb.first_block)] = content
            self.fat.updateBitmap(evalPro(fcb.first_block), '----')
            self.super_block.free_block -= 1
            self.updateParentSize(self.getCurFileLocation(path), 1)
            fcb.size = 1
        fcb.update_time =
datetime.datetime.strftime(datetime.datetime.now(), '%Y-%m-%d-%X')
        return '1'
```

- 保存及复原

  本项目使用字符位模拟实际磁盘中的二进制位，关闭网页时，将会把内存中的数据写回到disk.txt文件中。当再次开启时，文件系统将根据与写相对应的读规则进行读取，恢复。

```
    def saveToDisk(self):
        os.remove('disk.txt')
        with open('disk.txt', 'w+') as file:
            file.write(self.super_block.writeToDisk() + '\n')
            for i in range(0, len(self.fat.bitmap), 64):
                file.write(self.fat.bitmap[i:i + 64] + '\n')
            for i in range(0, len(self.dictionary.fcb_bitmap), 64):
                file.write(self.dictionary.fcb_bitmap[i:i + 64] + '\n')
            for f in self.dictionary.fcb_list:
                file.write(f.writeToDiskFirstLine())
                file.write(f.writeToDiskSecondLine())
            for d in self.data:
                file.write(d + '\n')
```



此外，本项目还额外实现了更改文件名，查看文件信息，更换用户等功能。由于篇幅有限，不作详细描述。可以根据`app.py`中的接口，进一步查看相关实现。

注意

- 现阶段的fcb_bitmap暂时看来效率不高，需要遍历操作，但是如果实际操作系统中使用位操作，还是快于遍历。
- 由于存储文件系统进txt时，数据块是以行作为区分，如果文件中写入换行符，可能会导致错误，但这本身与文件系统算法关系不大。
- 对于文件或者文件夹更新自身的信息来说，只会更新其自己的update_time，对于文件，更新其内容会更新update_time，并会相应更新其父目录update_time。
- 由于存在这样一种情况，用户1有进入文件夹A的权限，但用户2没有。如果此时从用户1切换到用户2，会出现权限错误，所以当用户发生切换时，系统统一返回root目录下。
- 数据块数量和FCB数量有限，如果数据块或FCB满了，那么新输入的文件内容可能无法保存。