

OS-FileSystem

Third assignment of OS course, implementing a simulation of File System.

一、系统概述

1.1需求分析

文件系统模拟

内容

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。

技术细节

文件存储空间管理可采取显式链接（如FAT）或者其他方法。（即自选一种方法）
空闲空间管理可采用位图或者其他方法。如果采用了位图，可将位图和FAT表合二为一。
文件目录采用多级目录结构。至于是否采用索引节点结构，自选。目录项目中应包含：文件名、物理地址、长度等信息。同学可在这里增加一些其他信息

1.2系统功能

1.2.1 文件系统指令

[mkdir]:创建文件夹

[ls]:列出当前文件夹

[vi]:创建一个文本文件并写入内容，输入:q退出

[more]:查看文件内容

[cd]: cd [file_name] 进入当前目录指定文件夹，[..]回退一层，[.]保持当前位置

[rm]:删除当前目录下的一个文件或文件夹

[find]:查找当前文件夹及子文件内指定名字的文件

[clear]:清屏

[exit]:退出并持久化数据

1.3开发工具

Python开发

Pickle作为数据持久化工具

运行在Windows系统上

Pyinstaller打包

二、运行演示


初始页面



选择C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe


```
Node Manager initialized
Block Manager initialized
File Manager initialized.
*****
*****                Welcome to the GreilFS!                *****
*****                Designed by Fangpei.                    *****
*****                V1.0                                     *****
*****
GreilFS root\:_
```

查看可用命令

 C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe

```
Node Manager initialized
Block Manager initialized
File Manager initialized.
*****
*****      Welcome to the GreilFS!      *****
*****      Designed by Fangpei.          *****
*****      V1.0                          *****
*****
GreilFS root\:help
[mkdir]:Create a new directory
[ls]:List all the file
[more]:Check the details
[vi]:Create a new file
[cd]:Go to a diretory
[rm]:Delete a file or directory
[find]:Search the subfile in the current directory
[exit]:Exit and save
GreilFS root\:
```

查看目录

 C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe

```
Node Manager initialized
Block Manager initialized
File Manager initialized.
*****
*****      Welcome to the GreilFS!      *****
*****      Designed by Fangpei.          *****
*****      V1.0                          *****
*****
GreilFS root\:ls
None

GreilFS root\:_
```

创建目录




C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe

```
Node Manager initialized
Block Manager initialized
File Manager initialized.
*****
*****          Welcome to the GreilFS!          *****
*****          Designed by Fangpei.              *****
*****                      V1.0                  *****
*****
GreilFS root\:ls
None

GreilFS root\:mkdir dir_1
GreilFS root\:ls
dir_1

GreilFS root\:_
```

创建文本文件

 C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe

```
Node Manager initialized
Block Manager initialized
File Manager initialized.
*****
*****      Welcome to the GreilFS!      *****
*****      Designed by Fangpei.          *****
*****      V1.0                          *****
*****
GreilFS root\:ls
None

GreilFS root\:mkdir dir_1
GreilFS root\:ls
dir_1

GreilFS root\:vi readme.txt
这是我的文件系统，使用ufs结构
支持一些基本命令
:q
Create successfully
GreilFS root\:ls
dir_1
readme.txt

GreilFS root\:_
```

打开文本文件

```
*****
*****      Welcome to the GreilFS!      *****
*****      Designed by Fangpei.          *****
*****      V1.0                          *****
*****
GreilFS root\:ls
None

GreilFS root\:mkdir dir_1
GreilFS root\:ls
dir_1

GreilFS root\:vi readme.txt
这是我的文件系统，使用ufs结构
支持一些基本命令
:q
Create successfully
GreilFS root\:ls
dir_1
readme.txt

GreilFS root\:more readme.txt
这是我的文件系统，使用ufs结构
支持一些基本命令

GreilFS root\:
```

进入和退出目录

```
*****          Designed by Fangpei.          *****
*****          V1.0          *****
*****
GreilFS root\:ls
None

GreilFS root\:mkdir dir_1
GreilFS root\:ls
dir_1

GreilFS root\:vi readme.txt
这是我的文件系统，使用ufs结构
支持一些基本命令
:q
Create successfully
GreilFS root\:ls
dir_1
readme.txt

GreilFS root\:more readme.txt
这是我的文件系统，使用ufs结构
支持一些基本命令

GreilFS root\:cd dir_1
GreilFS root\dir_1\:ls
None

GreilFS root\dir_1\:cd ..
root\
GreilFS root\:_
```

删除文件



C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe

dir_1

GreilFS root\:vi readme.txt

这是我的文件系统，使用ufs结构
支持一些基本命令

:q

Create successfully

GreilFS root\:ls

dir_1

readme.txt

GreilFS root\:more readme.txt

这是我的文件系统，使用ufs结构
支持一些基本命令

GreilFS root\:cd dir_1

GreilFS root\dir_1\:ls

None

GreilFS root\dir_1\:cd ..

root\

GreilFS root\:ls

dir_1

readme.txt

GreilFS root\:rm readme.txt

GreilFS root\:ls

dir_1

GreilFS root\:_

查找文件



C:\课程文档\操作系统\1651718_方沛_FileSystem\main.exe

```
*****
*****      Welcome to the GreilFS!      *****
*****      Designed by Fangpei.          *****
*****      V1.0                          *****
*****
GreilFS root\:ls
None

GreilFS root\:mkdir dir_1
GreilFS root\:mkdir dir_2
GreilFS root\:ls
dir_1
dir_2

GreilFS root\:cd dir_2
GreilFS root\dir_2\:mkdir dir_2_1
GreilFS root\dir_2\:ls
dir_2_1

GreilFS root\dir_2\:cd dir_2_1
GreilFS root\dir_2\dir_2_1\:vi target.txt
这是我要找到目标文件
:q
Create successfully
GreilFS root\dir_2\dir_2_1\:ls
target.txt

GreilFS root\dir_2\dir_2_1\:cd ..
root\dir_2\
GreilFS root\dir_2\:cd ..
root\
GreilFS root\:find target.txt
search result:
  root\dir_2\dir_2_1\target.txt

GreilFS root\:_
```

三、代码设计

1 主函数

```
import os
from managers import *
from toolkit import *
if __name__ == '__main__':
    if os.path.isfile('src_sys.pkl'):
        system=permanent_load('src_sys.pkl')
    else:
        system=FileSystem()
    system.welcome()
    system.recv_input()
    permanent_store(system, 'src_sys.pkl')
```

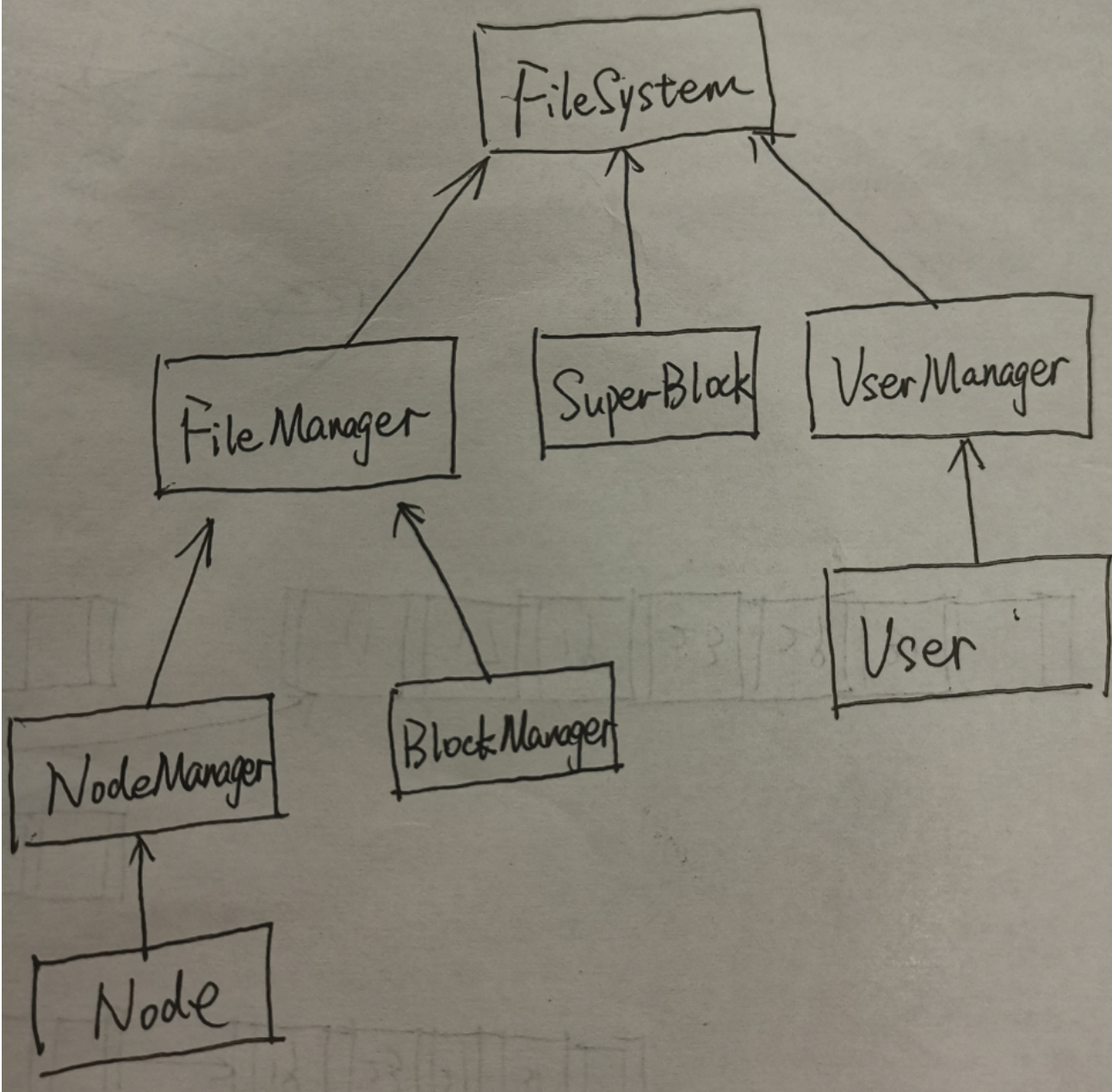
介绍:

进入前检查是否有存盘的pkl数据文件，若有则从中加载文件系统数据，没有则创建文件系统

2 类的结构

(代码太长，文档省略了内部具体功能函数实现，自顶向下进行说明)

结构.



2.1 FileSystem

```
def __init__(self):  
    self.super_block=SuperBlock()  
    self.file_manager=FileManager(self.super_block)  
    self.user_manager=UserManager()  
    self.parameter=[None,None]  
    self.operation=None  
    self.result=""  
    self.current_path="root\\"
```

```

#欢迎界面
def welcome(self):
#接收输入
def recv_input(self):
#规格化输入用于设置operation和parameter
def handle_input(self,box):
#根据输入执行对应
def answer(self):
#更新页面上显示的当前文件名
def roll_back(self)
#以下是具体的功能实现函数
def mkdir(self):
def ls(self):
def vi(self):
def more(self):
def cd(self):
def rm(self):
def rm_subfile(self,target_index):

```

介绍:

FileSystem是外部的主题函数，用于给用户提供执行命令的接口，内含负责维护用户信息的user_manager和执行文件操作的file_manager对象，以及负责传参的成员变量。

2.1.1 FileManager

```

def __init__(self,super_block):
    self.node_manager=NodeManager(super_block.bit,super_block.node_num)
    self.block_manager=BlockManager(
        super_block.bit,super_block.data_block_size,super_block.data_block_num)

    root_dir={
        '.': 'root',
        '..': 0
    }
    root_index=self.save(data=root_dir,sign='dir')
    # print("Root dir node index is ",root_index)
    print("File Manager initialized.")

    pass

#存储i节点和数据块信息
def save(self,data,sign):
#更新目录
def update_dir_file(self,location_index,new_dict,flag='add'):
#从i节点加载文件信息
def load(self,location_index,data_type):
#在rm命令中执行清除数据的函数
def delete(self,node_index):
#find命令是递归遍历子文件夹的操作
def subfile(self,index,dir_name):

```

介绍

FileSystem拥有的内部类，负责文件的所有操作，因此内部拥有node_manager和block_manager,分别负责对i节点的操作和对文件数据的操作。

2.1.1.1 NodeManager

```
def __init__(self, bit, num):
    self.map = np.zeros((bit, int(num/bit)))
    self.nodes = []
    for i in range(num):
        node = Node()
        self.nodes.append(node)
    print("Node Manager initialized")
    pass

def save(self, block_indexs, size, sign):
def allocate_nodes(self):
    # 擦除某一节点
def wipe(self, index):
    # 返回指定的索引节点
def get_node(self, index):
    # 根据i节点找到其对应的数据块
def get_block_indexs(self, index):
```

介绍:

负责i节点的增删改查，map负责记录i节点的使用情况，nodes负责记录每一个i节点

2.1.1.1.1 Node

```
class Node(object):
    def __init__(self, sign=None):
        self.file_size = 0
        self.block_num = 0
        self.sign = sign

        # 用于表示其对应的数据块节点
        self.block_index = {}

        # 采用ufs的索引结构
        for i in range(12):
            self.block_index[i] = None
        self.block_index[13] = {}

        # 文件信息更新函数
    def get_file_size(self):
    def set_file_size(self, file_size):
    def set_sign(self, sign):
```

```

#设置索引
def set_block_indexs(self,block_indexs):
#返回i节点的信息
def get_file_information(self):
#返回i节点对应的block索引
def get_block_indexs(self):

```

介绍:

超级块SuperBlock用于记录文件系统的参数,可以修改

实现inode, 内部记录了一些重要的文件信息, 这里采用ufs的多级目录结构, 只实现两级是因为两级已经可以提供1g的存储空间。

2.1.1.2 BlockManager

```

def __init__(self,bit,size,num):
    self.block_size=size
    self.map=np.zeros((bit,int(num/bit)))
    self.blocks=[b'']*num
    print("Block Manager initialized")
    pass

#保存数据块数据
def save(self,data):
#根据索引节点计算所需数据块大小并分配数据块
def allocate_blocks(self,size):
#擦除指定索引对应数据块数据
def wipe(self,indexs):
def wipe_data(self,content,indexs):
#数据块的读写
def write_data(self,data,indexs):
def read_data(self,indexs):

```

介绍

负责数据块的增删改查, map负责记录数据块的使用情况, blocks表示每一个数据块的实体。

2.1.2 UserManager

```

class UserManager(object):
    def __init__(self):
        self.user=None
        self.register()

    #用户的注册，保留了多用户的接口
    def register(self):
        #获得当前所在文件目录的node节点索引
    def get_current_dir_index(self):
        #重新设置当前所在文件目录的node节点索引
    def set_current_dir_index(self,new_index):

```

介绍:

记录用户初始化信息和当前用户所在目录

2.1.3 SuperBlock

```

class SuperBlock(object):
    def __init__(self):
        self.file_system_name = "Greilos"
        self.bit = 8
        self.file_system_size = 1024*1024*1024 #1G的文件大小
        self.block_index_size = 4 #块索引的大小
        self.node_size = 128 #每一个数据块的大小
        self.node_num = 120 #最多存储120个文件
        self.data_block_size = 8 * 1024
        self.data_block_num = 12000
        self.__address_size = 4

```

介绍:

超级块的实现，用于记录文件系统初始信息，用户可以根据电脑配置调整这些参数。

3 Toolkit

```

import pickle

def transform(data, to_type=None):
    if isinstance(data,str):
        data=bytes(data,encoding='utf-8')
    elif isinstance(data,dict):
        data=str(data)
        data=bytes(data,encoding='utf-8')

    elif isinstance(data,bytes):
        if to_type=="dir":
            data=eval(data)

```



```

        elif to_type=="text":
            #print("to text:",data)
            data=str(data,encoding='utf-8')
        else:
            data=data
    else:
        print("Data transform error!")
        return
    return data

def xy_to_index(y_length,x,y):
    return x*y_length+y

def index_to_xy(x_length,y_length,index):
    return int(index/y_length),index % y_length

def permanent_store(system,target):
    with open(target,'wb') as f:
        picklestring=pickle.dump(system,f)
        # f.write(picklestring)

def permanent_load(source):
    with open(source,'rb') as f:
        return pickle.load(f)

```

介绍:

toolkit 是一些在文件系统实现过程中用到的一些辅助函数，以及pickle数据持久化的函数实现，单独写成toolkit文件并在FileSystem实现过程中由外部引入。

四、分析与缺陷

这个项目对于文件系统有了更深的理解，包括i节点，超级块，数据块的作用和如何交互，也让我意识到我们日常使用的操作系统并没有那么简单，如删除和搜索都不是只操作当前目录即可，需要递归实现。经过重构代码设计上接口还是比较清晰的，同时对Python语法更加熟悉，尝试了使用Pickle进行数据持久化。

虽然实现了项目的要求，但是缺陷还是有很多的，由于时间问题尚未完善：

- 只支持层进式的进入和退出目录，主要由于inode只记录了三类信息
- 用户输入线程和计算线程没有分离，一旦文件存储耗时变长会造成假死，影响用户体验。
- 对用户各种错误输入没有办法完全保证系统不退出，当然在正确输入的前提下没有问题。
- 没有考虑边界情况，如block或inode全部用完等极端情况