分布式文件系统项目

姓名：邓栩瀛

班级：计算机科学与技术 1 班

学号：21307035

日期：2023.12

题目要求

设计一个分布式文件系统。该文件系统可以是 client-server 架构，也可以是 P2P 非集中式架构。 要求文件系统具有基本的访问、打开、删除、缓存等功能，同时具有一致性、 支持多用户特点。 在设计过程中能够体现在分布式课程中学习到的一些机制或者思想，例如 Paxos 共识、 缓存更新机制、 访问控制机制、并行扩展等。 实现语言不限， 要求提交代码和实验报告。

基本要求

1. 编程语言不限， 选择自己熟悉的语言，但是推荐用 Python 或者 Java语言实现
2. 文件系统中不同节点之间的通信方式采用 RPC 模式，可选择 Python 版本的 RPC、 gRPC 等
3. 文件系统具备基本的文件操作模型包括： 创建、 删除、 访问等功能
4. 作为文件系统的客户端要求具有缓存功能即文件信息首先在本地存储搜索， 作为缓存的介质可以是内存也可以是磁盘文件
5. 为了保证数据的可用性和文件系统性能，数据需要创建多个副本，且在正常情况下，多个副本不在同一物理机器，多个副本之间能够保持一致性（可选择最终一致性即延迟一致性也可以选择瞬时一致性即同时写）
6. 支持多用户即多个客户端，文件可以并行读写（即包含文件锁）
7. 对于上述基本功能， 可以在本地测试，利用多个进程模拟不同的节点，需要有相应的测试命令或者测试用例，并有截屏或者 video 支持
8. 提交源码和报告， 压缩后命名方式为： 学号_姓名_班级
9. 实验报告长度不超过 20 页

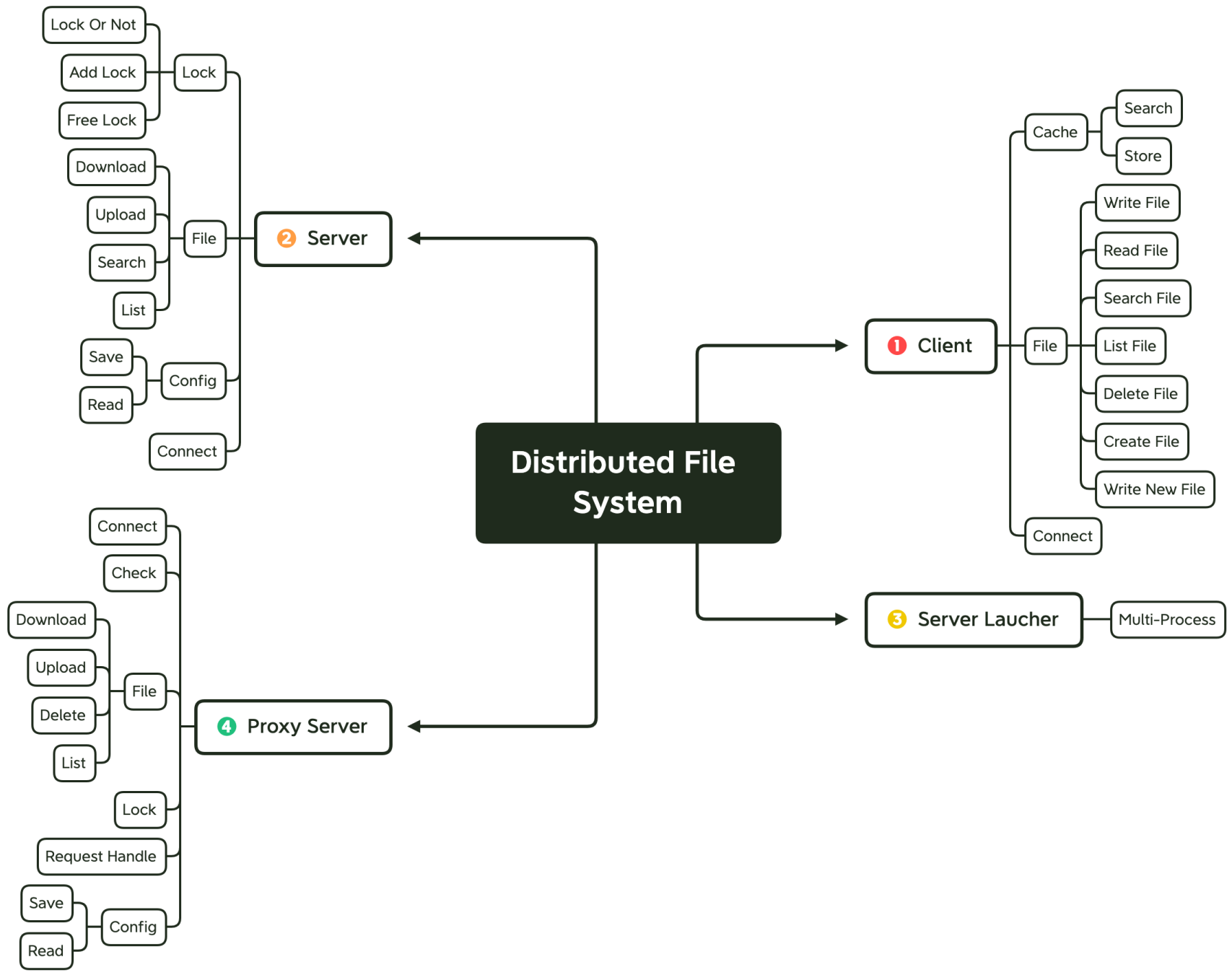
加分项

1. 加入其它高级功能如缓存更新算法
2. Paxos 共识方法或者主副本选择算法等
3. 访问权限控制
4. 其他高级功能

解决思路

1. 通信协议:使用RPC或gRPC来用于不同节点之间的通信，本实验使用Python中的rpyc库实现
2. 一致性协议:比如Paxos算法可以实现分布式一致性来保证数据的强一致性
3. 缓存机制:客户端本地缓存，集中式缓存服务等提升访问效率
4. 冗余和容错机制:数据复制、副本机制提升可用性和容错能力
5. 文件锁:为并发读写提供同步控制,解决并发冲突问题
6. 访问控制:为文件设置不同的权限，包括writable、readonly、private和full四种权限
7. 监控与经过:监控节点和系统运行状态，报告异常情况
8. 日志收集:进行日志记录，并用于后期分析

实现细节



（以下提到的代码有做删改，详细代码见源码文件）

Client

写文件，`:wq`退出，`d`删除，`r`替换，`i`插入

```
if line_input[0] == ':wq':
   break
if line_input[0] == 'i':
   text = input()
  if col == None:
      now_content.insert(row, text + '\n')
  else:
       new_line = now_content[row - 1][:col] + text + now_content[row - 1][col:]
      now_content[row - 1] = new_line
       elif line_input[0] == 'd':
            if col == None:
            now_content.pop(row - 1)
           else:
                new_line = now_content[row - 1][:col] + now_content[row - 1][col + 1:]
                now_content[row - 1] = new_line
           elif line_input[0] == 'r':
                text = input()
               if col == None:
                   now_content[row - 1] = text + '\n'
```

```
            else:
                new_line = now_content[row - 1][:col] + text + now_content[row - 1][col + 1:]
                now_content[row - 1] = new_line
```

创建文件，默认为 `writable` 模式，可以读和写，创建新文件需要判断是否存在同名文件

```
def create(self, file_name, mode='writable'):
    client_file = "%d_%s" % (self.id, file_name)
    content = self.writeNewFile(client_file)
    if content == None:
        message = 'Error: Another file already exists by this name.'
        print(message)
        self.log_file.write(message + '\n')
        return
    flag, message = self.conn.root.requestHandle(request, new_file_content=content)
    if flag == 0:
        self.log_file.write('Create %s accepted\n' % file_name)
```

读取文件

从缓存中读取文件，如果文件内容为空或者需要远程获取，则需要发送下载请求并获取文件内容；如果下载失败，将错误信息写入日志文件并打印，然后返回。接着发送释放读锁请求并获取结果，如果释放读锁失败，将错误信息写入日志文件并打印；如果释放读锁成功，将文件内容存储到缓存中。将读取成功的信息写入日志文件。

```
def read(self, owner_id, file_name, is_remote=0):
    client_file = '%d_%s' % (owner_id, file_name)
    content = self.searchInCache(client_file)
    if content == None or is_remote:
       request = self.commandToRequest('download %d %s' % (owner_id, file_name))
        flag, content = self.conn.root.requestHandle(request)
        if flag != 0:
          print(content)
          return
        self.printFileByRow(content)
        request = self.commandToRequest('freereadlock %d %s' % (owner_id, file_name))
        flag, message = self.conn.root.requestHandle(request)
        if flag != 0:
          print(message)
          return
          self.storeInCache(client_file, content)
    else:
          self.printFileByRow(content)
```

删除文件

从缓存中搜索文件内容，如果找到文件内容，删除缓存目录下的对应文件，发送删除请求并获取结果。

```
def delete(self, owner_id, file_name):
    client_file = '%d_%s' % (owner_id, file_name)
    content = self.searchInCache(client_file)
    if content != None:
        os.remove(self.cache_dir + '/' + client_file)
    request = self.commandToRequest('delete %d %s' % (owner_id, file_name))

    flag, content = self.conn.root.requestHandle(request)
    if flag != 0:
        print(content)
        return
    self.log_file.write('Delete %d/%s accepted\n' % (owner_id, file_name))
```

查找文件

在缓存目录中遍历文件，获取所有者ID和当前文件名，如果当前文件名与目标文件名相同，打印文件在缓存中的所有者和文件名，并将查找成功的信息写入日志文件。

```
def find(self, file_name):
    for x in os.listdir(self.cache_dir):
        temp = x.split('_')
        owner_id, now_file_name = int(temp[0]), temp[1]
        if now_file_name == file_name:
            self.log_file.write('Find %s accepted\n' % (file_name))
    request = self.commandToRequest('find %d %s' % (self.id, file_name))
    flag, content = self.conn.root.requestHandle(request)
    if flag != 0:
        self.log_file.write(content + '\n')
        print(content)
        return
    for x in content:
        print(x)
```

Server

检查是否存在锁的冲突

```
def isLockConflict(self, key, now_lock):
    file_lock.setdefault(key, [])
    locks = file_lock[key]
    if len(locks) == 0:
        return False
    for lock in locks:
        if lock[1] == 2 or (lock[1] == 1 and now_lock[1] == 2):
            return True
        if now_lock[0] == lock[0]:
            return True
    return False
```

添加锁

```
def addLock(self, key, now_lock):
    file_lock[key].append(now_lock)
```

释放锁

```
def freeLock(self, key, now_lock):
    file_lock.setdefault(key, [])
    if now_lock in file_lock[key]:
        file_lock[key].remove(now_lock)
        return 0
    return 5
```

上传文件

检查是否需要创建文件（是否文件的所有者/文件是否存在/是否存在锁冲突），如果需要进行锁定，则加锁，然后打开文件并写入新文件内容，需要设置文件权限，最后释放锁。

```
def exposed_uploadFile(self, request, new_file_content):
    now_file = self.__base_file_road + '/' + str(request['owner_id']) + '/' + request['file_name']
    key = (request['owner_id'], request['file_name'])
    client_id = request['client_id']
    free_lock = 'free_lock' in request and request['free_lock']
    is_create = 'is_create' in request and request['is_create']
    is_owner = client_id == key[0]
    now_lock = (client_id, 2)
    file_lock.setdefault(key, [])
    if free_lock == 0:
        self.addLock(key, now_lock)
    with open(now_file, 'w') as f:
        for x in new_file_content:
            f.writelines(x)
    if is_create:
        file_permission[key] = int(request['mode'])
    self.freeLock(key, now_lock)
```

Proxy Server

Master与用户建立联系

将客户端与主服务器建立连接，并记录连接信息到日志文件中。同时，通过调用其他客户端的 `connectionEstablish` 方法，将连接请求传递给其他客户端。

```
def exposed_connectionEstablish(self, request):
    client_id = request['client_id']
    if client_id in client_connection:
        return 7, PROMPTING_MESSAGE[7]
    client_connection[client_id] = 1
    log_file[client_id] = open(BASE_PROXYSERVER_LOGDIR + 'Log_Client%d.txt' % client_id, 'a')
    now_time = time.strftime("%a %b %d %H:%M:%S %Y\n", time.localtime())
    log_file[client_id].write(now_time)
    flag = 0
    for x in clients:
        x.root.connectionEstablish(request)
    return flag, None
```

Master与用户断开连接

```
def exposed_connectionCancel(self, request):
    client_id = request['client_id']
    if client_id not in client_connection:
        return 8, PROMPTING_MESSAGE[8]
    client_connection.pop(client_id)
    log_file[client_id].close()
    log_file.pop(client_id)
    flag = 0
    for x in clients:
        x.root.connectionCancel(request)
    return flag, None
```

处理文件上传请求

检查请求的合法性，包括是否是创建操作和是否是文件的所有者，根据文件是否存在选择要上传的服务器，并逐个将上传请求传递给相应的客户端。如果有任何错误发生，记录错误信息并返回相应的错误代码和消息。如果上传成功，返回标志表示成功。

```
def uploadFile(self, request, new_file_content):
    flag = 0
    key = (request['owner_id'], request['file_name'])
    client_id = request['client_id']
    # free_lock = request.get('free_lock', 0)
    is_create = 'is_create' in request and request['is_create']
    if is_create and client_id != key[0]:
        return 5, PROMPTING_MESSAGE[5]
    if key not in file_to_servers:
        if not is_create:
            return 5, PROMPTING_MESSAGE[5]
        servers_len = len(clients)
        half_len = ((servers_len) // 2) | 1
        L = [i + 1 for i in range(servers_len)]
        random.shuffle(L)
        servers = L[:half_len]
        file_to_servers[key] = servers
    error_id = 0
    for i in file_to_servers[key]:
        flag = clients[i - 1].root.uploadFile(request, new_file_content)
        if flag != 0:
            error_id = i
            break
      log_file[client_id].write("Upload %s to Server %d Accepted\n" % (key[1], i))
    if flag != 0:
        error_message = PROMPTING_MESSAGE[flag]
        log_file[client_id].write("Server %d: %s\n" % (error_id, error_message))
        return flag, error_message
    return flag, None
```

运行情况

启动Server端

```
python ServerLaucher.py --n 5
```

n为服务器数量，默认为5，上限为10

启动Master

```
python ProxyServer.py --n 5
```

Master中 –n选项的值必须与ServerLaucher的值一致

启动Client端

```
python Client.py
```

Step1

注册一个新用户，ID为1，并在服务器创建4个文本文件

1.txt 允许其他用户读和写，但不允许其他用户删除（writable），默认模式即为writable

```
create 1.txt
```

2.txt 只允许其他用户读取（readonly）

```
create 2.txt readonly
```

3.txt 是私有文件，不允许其他用户访问（private）

```
create 3.txt private
```

4.txt 是公有文件，允许别的用户读、写且删除（full）

```
create 4.txt full
```

创建文件后即可输入文件内容，并以 `:wq` 结束

```
>>Please Input your id: 1
[1] >>create 1.txt
id 1
file 1
:wq
[1] >>create 2.txt readonly
id 1
file 2
:wq
[1] >>create 3.txt private
id 1
file 3
:wq
[1] >>create 4.txt full
id 1
file 4
:wq
```

Step2

注册一个新用户，ID为2，并在服务器创建三个文本文件

```
create 1.txt
create 3.txt readonly
create 5.txt private
```

```
>>Please Input your id: 2
[2] >>create 1.txt
id 2
file 1
:wq
[2] >>create 3.txt readonly
id 2
file 3
:wq
[2] >>create 5.txt private
id 2
file 5
:wq
```

Step3

注册一个新用户，ID为3，并在服务器创建一个文本文件

```
create 1.txt
```

```
>>Please Input your id: 3
[3] >>create 1.txt
id 3
file 1
:wq
```

此时，如果再输入以下命令进行创建文件，则会报错

```
create 1.txt
```

```
[3] >>create 1.txt

id 3
new file 1
:wq
Error: Another file already exists by this name.
```

系统检测到有同名文件存在，操作终止

Step4

对用户3，执行查找 `find` 命令，其中，private文件只有所有者可见，其他用户不可见

```
find 1.txt
find 3.txt
find 5.txt
```

```
[3] >>find 1.txt
3/1.txt in cache
3/1.txt
1/1.txt
2/1.txt
[3] >>find 3.txt
2/3.txt
[3] >>find 5.txt
Error: Such file does not exist or it has been removed for.
```

执行 `list` 命令，列出一个路径下的所有文件

```
list
list cache
list 1
```

```
[3] >>list
Cache:
Remote:
1.txt
[3] >>list cache
Cache:
[3] >>list 1
2.txt
1.txt
4.txt
```

对于其他用户的个人文件夹中的private文件会对当前用户隐藏，因此不会显示在终端，上述例子可以发现，用户1的3.txt为private，对用户3不可见

如果不加路径名选项，默认会列出当前用户本地的cache和文件系统的个人文件夹中的所有文件名

`list cache`用于列出当前用户cache文件夹中的所有文件名（cache中的文件有一定几率被抛弃）

Step5

对用户3，执行`write`命令，写一个文件

```
write 1.txt
```

写文件的命令为`write [filename]`，若文件名没有指定用户（指定用户ID/filename；不指定用户，默认写自己的文件）

```
[3] >>write 1.txt
1   id 3
2   file 1
:wq
```

相关命令如下

> i [m] [n]：在第m行第n列后插入新内容（参数n可省略，之后接受一行输入，代表要插入的内容）
>
> r [m] [n]：替换第m行第n列的内容（参数n可省略，之后接受一行输入，代表要用来替换的内容）
>
> d [m] [n]:删除第m行第n列的内容（参数n可省略）
>
> :wq：退出写文件模式

```
[3] >>write 1.txt
1   id 3
2   file 1
i 1
hello world
1   id 3
2   hello world
3   file 1
r 3
file 1.txt
1   id 3
2   hello world
3   file 1.txt
d 2
1   id 3
2   file 1.txt
:wq
```

用户3在2/3.txt中写文件

```
write 2/3.txt
```

如果尝试写一个readonly的文件，会提示权限不够

```
[3] >>write 2/3.txt
Error: Permission Denied.
```

用户3在1/3.txt中写文件

```
write 1/3.txt
```

如果尝试写一个private的文件，会提示不存在该文件

```
[3] >>write 1/3.txt
Error: Such file does not exist or it has been removed for.
```

Step6

用户2执行 `read` 命令

```
read 1.txt
read 1/1.txt
```

读文件的命令为 `read [filename]`，若文件名没有指定用户（指定用户ID/filename; 不指定用户，默认读自己的文件）

输入命令后，会进入读文件模式，读文件模式输入Enter退出

```
[2] >>read 1.txt
1   id 2
2   file 1
Press Enter to Continue.
[2] >>read 1/1.txt
1   id 1
2   file 1
Press Enter to Continue.
```

Step7

用户2读文件时，用户3写同一个文件

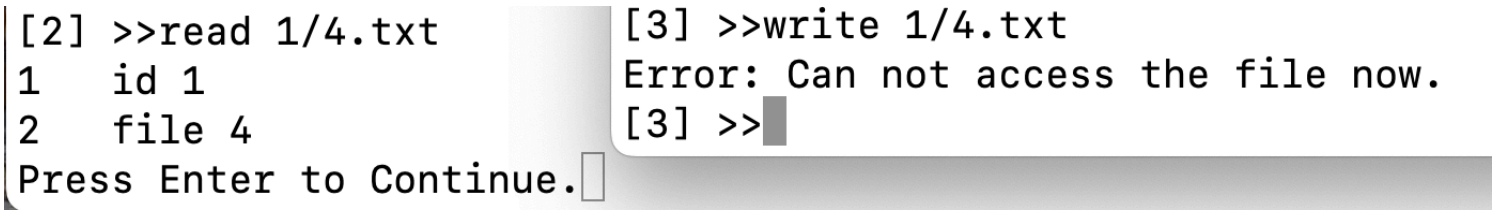
用户2执行

```
read 1/4.txt
```

用户3执行

```
write 1/4.txt
```

用户读文件时，会给文件加锁，直到退出读文件模式才解锁，其他用户无法同时写同一文件。



用户2读文件时，用户3读同一个文件

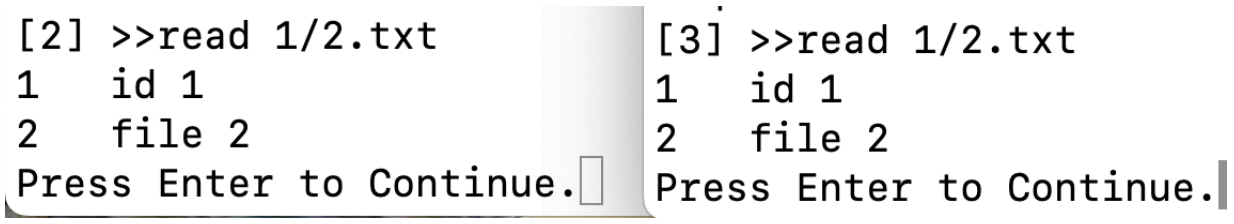
用户2执行

```
read 1/2.txt
```

用户3执行

```
read 1/2.txt
```

读操作可以同时执行



用户2写文件时，用户3读同一个文件

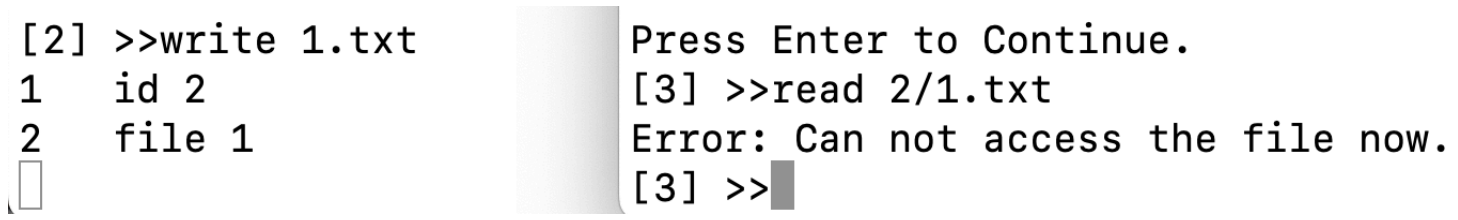
用户2执行

```
write 1.txt
```

用户3执行

```
read 2/1.txt
```

用户写文件时，会给文件加锁，直到退出写文件模式才解锁，其他用户无法同时读取同一文件。



Step8

用户2执行 `delete` 命令删除一个文件

```
delete 1.txt
delete 1/1.txt
delete 1/4.txt
delete 1.txt
```

删除文件的命令为 `delete [filename]`，若文件名没有指定用户（指定用户ID/filename; 不指定用户，默认删除自己的文件）

删除一个不存在的文件，会报错

如果删除其他用户的权限级别不是full的文件（如上图中的1/1.txt），会提示权限不够

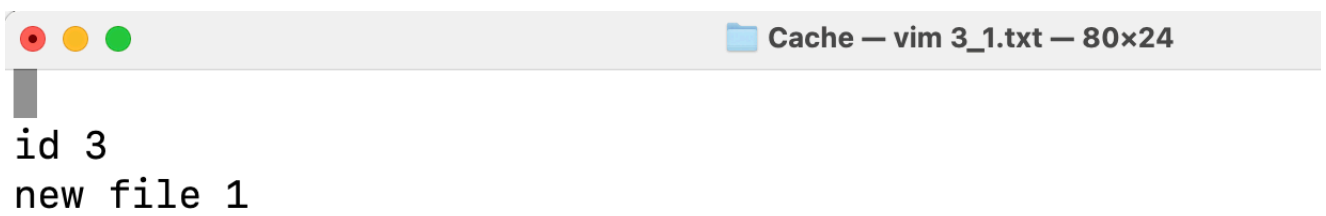
```
[2] >>delete 1.txt
[2] >>delete 1/1.txt
Error: Permission Denied.
[2] >>delete 1/4.txt
[2] >>delete 1.txt
Error: Such file does not exist or it has been removed for.
```

相关文件查看

Step1

退出系统后，可以通过路径`Client/ID/Cache`查看客户本地缓存信息

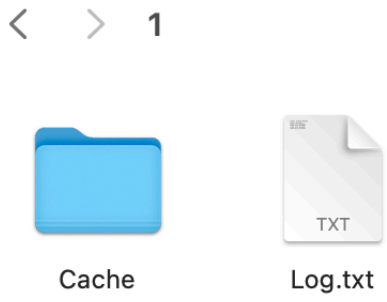
查看该缓存文件的内容



该内容为前面第二次执行`create 1.txt`后写入文件的内容，但由于同名文件存在，故创建文件失败，内容写入Cache

Step2

查看客户端日志文件，路径为`Client/ID/log.txt`

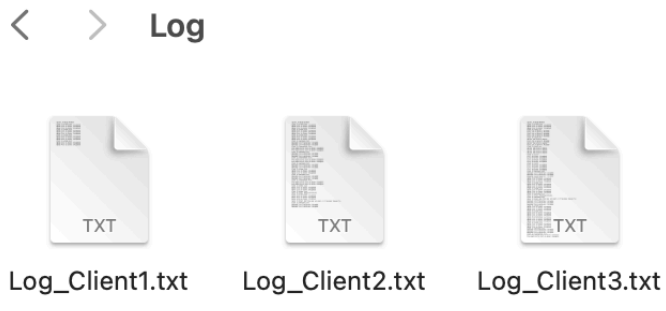


以用户1为例，查看其日志文件

```
Tue Dec 19 10:12:46 2023
Connect accepted
Create 1.txt accepted
Create 2.txt accepted
Create 3.txt accepted
Create 4.txt accepted
```

Step3

查看模拟的Master端日志文件，路径是`ProxyServer/Log`



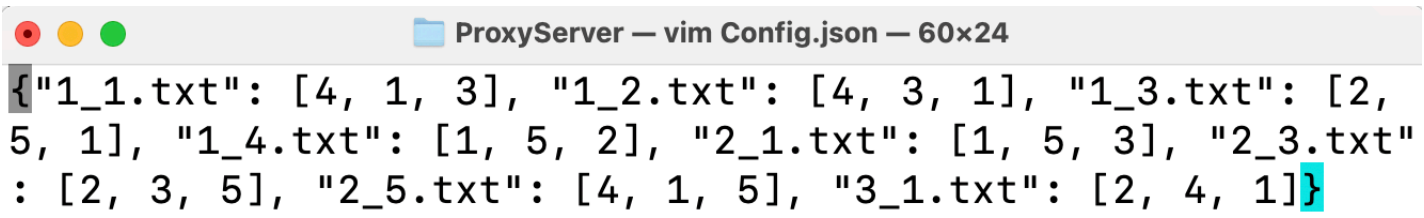
查看`Log_Client1.txt`文件

```
Tue Dec 19 10:12:46 2023
Client 1: upload 1.txt
Upload 1.txt to Server 4 Accepted
Upload 1.txt to Server 1 Accepted
Upload 1.txt to Server 3 Accepted
Client 1: upload 2.txt
Upload 2.txt to Server 4 Accepted
Upload 2.txt to Server 3 Accepted
Upload 2.txt to Server 1 Accepted
Client 1: upload 3.txt
Upload 3.txt to Server 2 Accepted
Upload 3.txt to Server 5 Accepted
Upload 3.txt to Server 1 Accepted
Client 1: upload 4.txt
Upload 4.txt to Server 1 Accepted
Upload 4.txt to Server 5 Accepted
Upload 4.txt to Server 2 Accepted
~
~
~
~
~
"Log_Client1.txt" 17L, 525B
```

Step4

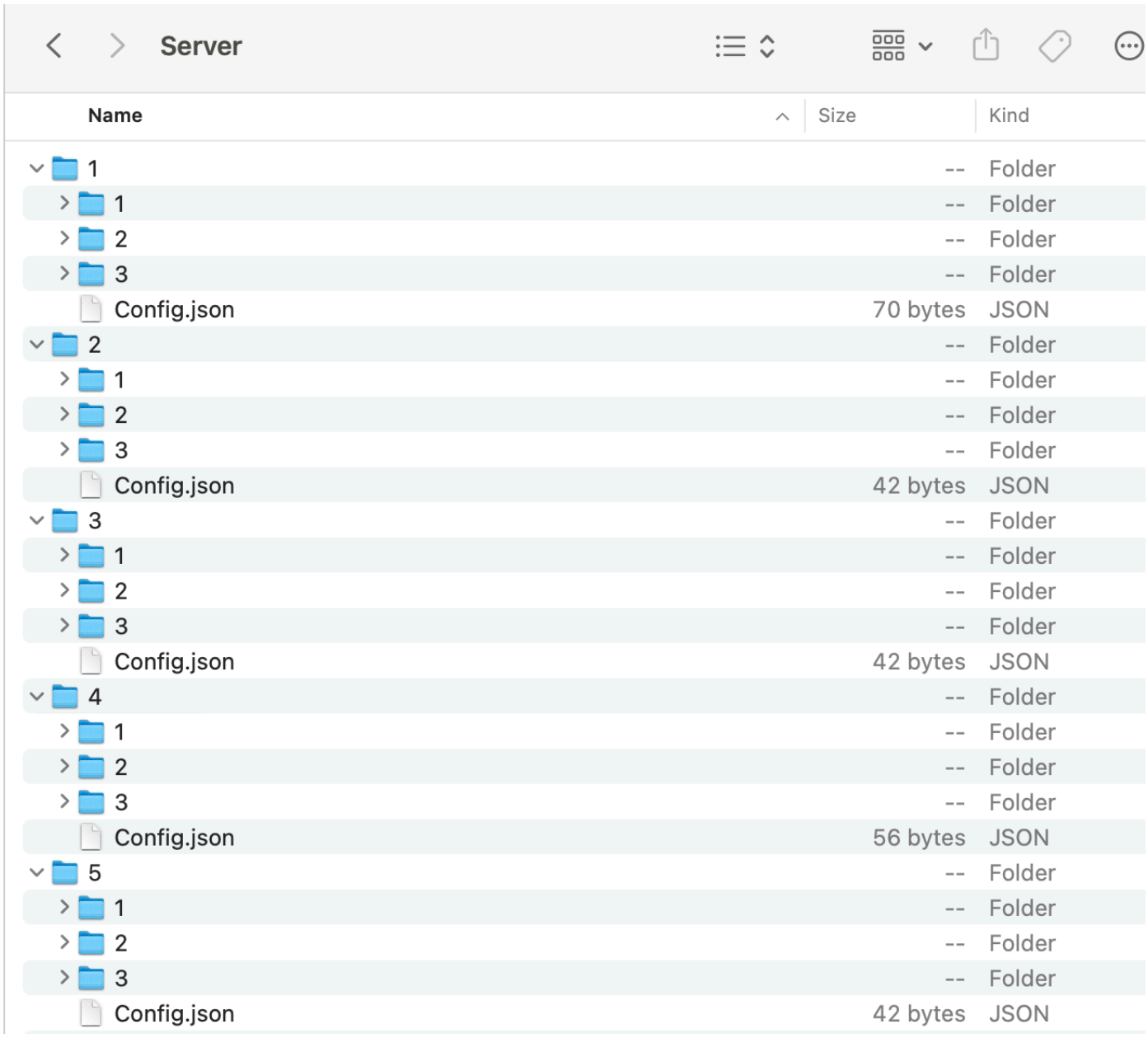
查看模拟的Master端的配置文件，路径是`ProxyServer/Config.json`

配置文件记录了不同的文件存放在哪个远程文件服务器上



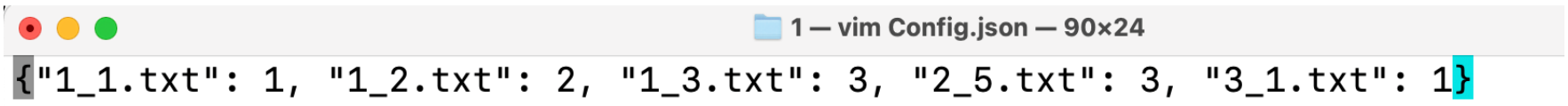
Step5

查看Server端文件夹，路径为`Server/ID`



Server端的配置文件`Config.json`记录不同文件的访问权限

以用户1为例



重新加载

重新打开文件系统，加载之前的配置

```
>>Please Input your id: 1
[1] >>find 1.txt
3/1.txt
1/1.txt
[1] >>find 5.txt
Error: Such file does not exist or it has been removed for.
[1] >>write 2/3.txt
Error: Permission Denied.
```

权限配置及文件配置加载成功

遇到的问题

1. 缓存的更新过于简单，当一个文件在远程文件系统被修改后，缓存不会立即更新，只有在用户读写该文件时，缓存才会更新。

2. 处理远程文件系统中某些文件损坏的问题，只用了简单的投票方法。
3. 无法处理远程服务器和Master结点宕机的问题。
4. 客户端用户的登录和退出登录的实现方法比较简单。
5. 服务器节点有限，且数量较少。
6. 多个客户端并行读写文件时，需要处理并发访问的一致性问题，如避免数据竞争和冲突，在实验中设计了有效的文件锁机制来防止并发写入导致的数据不一致。
7. 在分布式环境中，网络故障、节点故障或其他异常情况可能导致数据不一致或丢失，也需要设计机制来确保数据的一致性和可靠性，例如复制、冗余存储、故障恢复和数据备份等。
8. 分布式文件系统需要处理大量的数据和并发请求，性能和延迟是关键问题。需要考虑有效的数据传输和处理机制，以及合理的负载均衡和缓存策略，以提高系统的性能和响应速度。
9. 保护分布式文件系统的数据安全和隐私是重要的问题，需要设计适当的访问控制机制和身份验证，以确保只有授权用户可以访问和修改文件。
10. 分布式系统中的节点故障或网络故障可能导致系统的不可用性，需要设计容错机制和故障恢复策略，以确保系统在部分故障或异常情况下仍能正常运行。
11. 随着用户数量和数据量的增长，分布式文件系统需要具备良好的可伸缩性和扩展性，需要考虑水平扩展和负载均衡策略，以支持系统的可扩展性并满足不断增长的需求。

总结

1、通过这个项目，实现了一个简单的分布式文件系统，加深了对分布式系统中文件读写锁、缓存一致性、文件权限管理等一系列知识的理解。

2、在实现的过程中深刻地体会了分布式系统的作用，并在之后的学习过程中运用分布式系统的相关功能。

3、在设计分布式文件系统之前，深入了解相关的分布式系统理论、分布式文件系统的架构模式和现有的研究成果，借鉴经验。

4、通过实验设计和实现分布式文件系统，可以深入了解分布式系统的原理和技术。在实验过程中，不仅可以应用之前学到的知识，还可以不断学习和探索新的技术和概念。

5、在实验中，可能会遇到各种问题和挑战，如一致性问题、并发访问问题、性能优化等。解决这些问题需要思考和尝试不同的方法和策略，提高问题解决的能力。