《分布式系统》

实 验 报 告

实验序号：期末实验

实验名称：分布式文件系统设计

姓名：彭子辉

学号：16337196

实验名称： 分布式文件系统设计

1. 实验要求

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 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。

2. 思路与解决方案

2.1. 分布式文件系统架构设计

本实验中的分布式文件系统由 4 部分组成，这 4 个部分分别为客户端 Client，文件服务器 fileServer，文件路径管理服务器 directoryServer，锁服务器 lockingServer。它们的具体功能以及组织方式如下所述。

1. 客户端 Client

客户端负责处理并执行用户输入指令，并最终显示指令执行结果。客户端处理用户指令的步骤如下所示:

- 客户端 Client 读取用户的输入指令
- 客户端 Client 根据用户输入指令，生成特定的指令报文 1，将报文 1 发送给文件路径管理服务器 directoryServer，从响应报文中确定用户所要操作的文件所在的文件服务器的 IP 地址和端口。
- 客户端 Client 根据用户输入指令，生成特定的指令报文 2，将报文 2 发送给锁服务器 lockingServer, 从响应报文中确定用户所要操作的文件是否存在锁以及锁的相关信息。
- 客户端 Client 根据前面两个步骤中, 从文件路径服务器以及锁服务器的响应报文中得到的文件信息，确定是否执行用户的指令，并在需要时生成指令报文 3，将报文 3 发送给文件服务器 fileServer 以完成对文件的操作。

2. 文件服务器 fileServer

文件服务器 fileServer 负责存储具体的用户的文件，在本实验中可能存在多个文件服务器，这些服务器均受文件路径管理服务器 directoryServer 的管理。

文件服务器 fileServer 实现的基本功能包括：存储文件，打开文件，读取文件以及写文件。

3. 文件路径管理服务器 directoryServer

文件路径管理服务器 directoryServer 负责管理文件所存储的位置，在本实验中只能存在一个文件路径管理服务器，该文件路径管理服务器存储了所有文件服务器的 IP 地址和端口号，以及"文件名-所在文件服务器和端口号"的键值对，用这种实现思路来管理所有文件的路径。

文件服务器 directoryServer 实现的基本功能包括：查询文件是否存在，查询文件所在的文件服务器 IP 地址和端口号，添加文件路径和删除文件路径。

4. 锁服务器 lockingServer

锁服务器 lockingServer 中负责管理文件的锁，本实验中只能存在一个锁服务器，锁服务器记录了"文件名-加锁用户"键值对，可以有效管理文件锁的状态。

锁服务器 lockingServer 实现的基本功能包括：查询文件是否存在锁，查询给特定文件加锁的用户，文件加锁和文件解锁。

3. 实验具体实现

3.1. 客户端 Client 的具体实现

附注：客户端 Client 的具体实现请参见 Client.py。

1. 客户端 Client 初始化

客户端 Client 的初始化需要分布式文件系统的文件路径管理服务器的 IP 地址及端口，锁服务器的 IP 地址及端口即可。

具体的实现算法细节较多，请参考下述代码中的详细注释。

```
    def __init__(self, directoryAddress, directoryPort, lockAddress,
lockPort):
        '''
        : 初始化分布式文件系统客户端
```

```
        : directoryAddress: str,文件路径管理服务器 IP 地址
        : directoryPort: str, 文件路径管理服务器端口
        : lockAddress: str,锁服务器 IP 地址
        : lockPort: str,锁服务器端口
        '''
        self.id = str(uuid.uuid1())
        self.masterAddr = directoryAddress    #文件服务器 IP 地址
        self.directoryPort = directoryPort    #文件服务器端口
        self.lockAddr = lockAddress           #锁定服务 IP 地址
        self.lockPort = lockPort              #锁定服务端口
        self.fileCache = {}                   #客户端文件缓存
```

2. 客户端打开文件——open 方法

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
    def open(self, docname):
        '''
        : 客户端打开一个文件
        : docname: str,文件名
        '''

        #1. 客户端建立到文件服务器的链接
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((self.masterAddr, self.directoryPort))

        #2. 客户端发送一个 open 类型的请求报文给服务器，指示打开指定的文件
        message = json.dumps({"request": "open", "docname": docname,
"clientid": self.id})
        sock.sendall(message.encode())
        response = sock.recv(1024)

        return response
```

3. 客户端关闭文件——close 方法

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
    def close(self, docname):
        '''
        : 客户端关闭一个文件
        : docname: str,文件名
        '''

        #1. 客户端建立到文件服务器的链接
```

```
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((self.masterAddr, self.directoryPort))

        #2. 客户端发送一个 close 类型的请求报文给服务器，指示关闭指定的文件
        message = json.dumps({"request": "close", "docname": docname,
"clientid": self.id})
        sock.sendall(message.encode())
        response = sock.recv(2048)
        return response
```

4. 客户端检查文件的锁状态——checkLock 方法

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
    def checkLock(self, docname):
        '''
        : 客户端检查一个文件的锁状态
        : docname: str,文件名
        '''

        #1. 客户端建立到锁服务器的链接
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((self.lockAddr, self.lockPort))

        #2. 客户端发送一个 checklock 类型的请求报文给服务器，指示要获得指定文
件的锁的状态
        message = json.dumps({"request": "checklock", "docname":
docname, "clientid": self.id})
        sock.sendall(message.encode())
        response = sock.recv(1024)

        return response
```

5. 客户端文件加锁——obtainLock 方法

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
    def obtainLock(self, docname):
        '''
        : 为指定的打开的文件获得锁
        : docname: str,文件名
        '''

        #1. 客户端建立到锁服务器的链接
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
        sock.connect((self.lockAddr, self.lockPort))

        #2. 客户端发送一个 obtainlock 类型的请求报文给服务器，指示为指定的文件获得锁
        message = json.dumps({"request": "obtainlock", "docname": docname, "clientid": self.id})
        sock.sendall(message.encode())
        response = sock.recv(1024)

        return response
```

6. 客户端文件读取——read 方法

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
    def read(self, docname):
        '''
        : 读取指定文件名的文件
        : docname: str,文件名
        '''

        #1. 首先，调用打开文件方法 open，该方法将通知路径服务器打开文件，该方法的返回值中包含该文件的具体位置信息
        fileServerInfo = json.loads(self.open(docname))

        #2. 然后，检查 open 方法的服务器响应报文（返回值）fileServerInfo 的'isFile'字段，该字段为 True 表示目标文件存在，否则目标文件不存在
        if fileServerInfo['isFile']:

            #3. 若 fileServerInfo['isFile']==True，即该文件存在，则首先检查该文件是否在缓存中，并且通过 timestamp 字段检查缓存中的副本是否为最新版本
            if (docname in self.fileCache) and (self.fileCache[docname]['timestamp'] >= fileServerInfo['timestamp']):
                fileCacheFileInfo = self.fileCache[docname]   #3.1 若为最新版本，则直接返回缓存中的目标文件副本即可
                print("Read '" + docname + "' from fileCache!")
                return fileCacheFileInfo
            else: #3.2 若不为最新版本，则根据 fileServerInfo 的具体位置信息访问文件服务器获得最新版本，并更新缓存中的目标文件副本为最新版本
                addr = fileServerInfo['address']
                port = int(fileServerInfo['port'])
                #4. 客户端再与存有文件的服务器建立连接
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sock.connect((addr, port))
```

```
                #5. 客户端发送一个 read 类型的请求报文给服务器，指示要从指定文件所在的服务器获得指定的文件的最新版本
                message = json.dumps({"request": "read", "docname": docname, "clientid": self.id})
                sock.sendall(message.encode())

                #6. 使用建立的连接获得最新版本的目标文件，并更新缓存中的副本
                response = sock.recv(1024)

                self.fileCache['docname'] = json.loads(response)

                #7. 返回含有读取文件结果的服务器响应报文
                return response
        else:
            return docname + "不存在!"
```

7. 客户端文件写入——write 方法

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
    def write(self, docname, data):
        '''
        : 将更新信息写入文件
        : docname: str,要写入的文件名
        : data: str,要写入的文件的数据
        '''

        #1. 获得需要写入的目标的文件的锁信息 lockcheck
        lockcheck = json.loads(client.checkLock(docname))

        #2. 若文件锁信息表项 lockcheck['response']="locked"，说明目标文件此时被其他客户端锁定，这时不能写该文件，故直接返回错误信息
        if lockcheck['response'] == "locked":
            return "Cannot write as file is locked by another client!"

        #3. 若文件锁信息表象 lockcheck['response']!='locked'，这时目标文件未被其他客户端锁定，这时可以准备写该文件，首先创建到文件所在服务器的链接
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((self.masterAddr, self.directoryPort))

        #5. 客户端发送一个 write 类型的请求报文给服务器，指示将要修改指定的文件
        timestamp = time.time()   #生成最新时间戳，作为更新版本号使用

        message = json.dumps({"request": "write", "docname": docname, "clientid": self.id, "timestamp": timestamp})
```

```
        sock.sendall(message.encode())

        #6. 客户端受到服务器响应，该响应回送一个报文 response，报文中包含目标文件所在的服务器 IP 地址和端口号
        response = sock.recv(1024)

        fileServerInfo = json.loads(response)

        addr = fileServerInfo['address']
        port = int(fileServerInfo['port'])

        #7. 客户端根据响应报文 response 的信息，再与存有文件的服务器建立连接
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((addr, port))

        #8. 客户端向文件所在的服务器发送 write-data 请求报文，将需要写入的数据放在该报文中，指示服务器重新写入文件，并更新 fileCache 中缓存的文件的版本（若没有则在缓存中创建该文件）
        #附注：需要特别注意，write-data 请求报文和 write 请求报文不相同；write 请求报文是发给根结点的，是要请求所要写的文件所在的服务器的 IP 和端口号；而 write-data 请求报文是发送给文件所在的服务器的，是要请求该服务器将数据写入指定文件
        content = {"request": "write", "docname": docname, "data": data, "clientid": self.id, "timestamp": timestamp}

        self.fileCache[docname] = content

        message = json.dumps(content)
        sock.sendall(message.encode())      #客户端向文件所在服务器发送 write-data 请求报文

        response = sock.recv(1024)
        return response
```

8. 客户端输入输出界面

具体的实现算法已经详细分步骤地体现在注释中，请参考下述代码中的详细注释。

```
if __name__ == '__main__':
    client = Client(DIRECTORY_SERVER_ADDRESS, DIRECTORY_SERVER_PORT, LOCK_SERVER_ADDRESS, LOCK_SERVER_PORT)

    typeOfCommand = ""
    response = ""
```

```
    while typeOfCommand != "exit":
        typeOfCommand = input("请输入一个操作指令
[open/close/checklock/read/write]，输入 exit 以退出:")

        if typeOfCommand == "open":
            docname = str(input("请输入文件名称: "))
            response = client.open(docname)
        elif typeOfCommand == "close":
            docname = str(input("请输入文件名称: "))
            response = client.close(docname)
        elif typeOfCommand == "checklock":
            docname = str(input("请输入文件名称: "))
            response = client.checkLock(docname)
        elif typeOfCommand == "read":
            docname = str(input("请输入文件名称: "))
            response = client.read(docname)
        elif typeOfCommand == "write":
            docname = str(input("请输入文件名称: "))
            data = str(input("请输入要写入的文件内容: "))
            response = client.write(docname, data)
        elif typeOfCommand == "exit":
            response = "成功退出系统!"
        else:
            response = "输入的指令不合法，请重新输入"

        print (response)
```

3.2. 文件服务器 fileServer 的具体实现

附注：客户端 fileServer 的具体实现请参见 fileServer.py。

1. 文件服务器 fileServer 初始化

文件服务器的初始化需要预先设定一个 IP 地址及端口，并指定文件在文件服务器上的存储路径。

```
NODEID = ""
ADDRESS = "127.0.0.1"
PORT = 0

MASTER_ADDRESS = "127.0.0.1"
MASTER_PORT = 8080

CURRENT_DIRECTORY = os.getcwd()
BUCKET_NAME = "FileServerBucket"          #文件储存路径
```

```
BUCKET_PATH = os.path.join(CURRENT_DIRECTORY, BUCKET_NAME)
```

2. 文件服务器 fileServer 打开文件——dfsOpen 方法

```
    def dfsOpen(docname):
        path = os.path.join(BUCKET_PATH, docname)
        exists = os.path.isfile(path)
        return exists
```

3. 文件服务器 fileServer 读取文件——dfsRead 方法

```
    def dfsRead(docname):
        path = os.path.join(BUCKET_PATH, docname)
        file_handle = open(path, "r")
        data = file_handle.read()
        return data
```

4. 文件服务器 fileServer 写入文件——dfsWrite 方法

```
    def dfsWrite(docname, data):
        path = os.path.join(BUCKET_PATH, docname)
        file_handle = open(path, "w+")
        file_handle.write(data)
```

5. 文件服务器的 SocketServer 框架请求处理组件——ThreadedHandler

```
class ThreadedHandler(socketserver.BaseRequestHandler):
    '''
    : SocketServer 网络服务框架组件
    : 继承自类 socketserver.BaseRequestHandler，该组件协调处理文件路径服务器与锁服务器之间的通信，负责处理文件路径服务器发来的请求报文
    '''
    def handle(self):
        msg = self.request.recv(1024)
        print(msg)

        msg = json.loads(msg)
        requestType = msg['request']
        response = ""

        if requestType == "open":
            exists = dfsOpen(msg['docname'])
            response = json.dumps({"response": requestType, "docname": msg['docname'], "isFile": exists, "address": ADDRESS, "port": PORT})
        elif requestType == "close":
```

```
            response = json.dumps({"response": requestType, "address":
ADDRESS, "port": PORT})
        elif requestType == "read":
            data = dfsRead(msg['docname'])
            response = json.dumps({"response": requestType, "address":
ADDRESS, "port": PORT, "data": data})
        elif requestType == "write":
            dfsWrite(msg['docname'], msg['data'])
            response = json.dumps({"response": requestType, "address":
ADDRESS, "port": PORT, "uuid": NODEID})
        else:
            response = json.dumps({"response": "Error", "error":
requestType+" is not a valid request", "address": ADDRESS, "port":
PORT})

        self.request.sendall(response.encode())
```

3.3. 文件路径管理服务器 directoryServer 的具体实现

附注：**文件路径管理服务器 directoryServer** 的具体实现请参见 directoryServer.py。

1. 文件路径管理服务器 directoryServer 初始化

文件服务器的初始化需要预先设定一个 IP 地址及端口，并初始化文件服务器列表和文件路径映射列表。

```
ADDRESS = "127.0.0.1"
PORT = 8080

#文件路径服务器使用典型的 SocketServer 网络服务框架进行组织

FILE_SERVER = {}      #文件服务器列表
FILE_ADDRESS = {}     #文件路径映射列表：将单个文件映射到对应的文件服务器
```

2. 文件路径管理服务器 directoryServer 查找文件路径——getFileAddress 方法

```
    def getFileAddress(docname):
    '''
    : 获得给定的文件所在的文件服务器地址
    : docname: str,文件名
    : return -> 指定文件所在的文件服务器地址
    '''
    if fileExistsTest(docname):
```

```
        return FILE_ADDRESS[docname]   #从文件路径映射列表中读取文件所在的服务器
    else:
        return None
```

3. 文件路径管理服务器 directoryServer 添加文件路径——addFileAddress 方法

```
    def addFileAddress(docname, nodeID, address, port, timestamp):
    '''
    : 向指定的文件服务器中加入新文件
    : docname: str,文件名
    : nodeID: str,客户端 ID
    : address: str,文件服务器地址
    : port: str,文件服务器端口
    : timestamp: str,版本控制时间戳
    '''
    FILE_ADDRESS['docname'] = {"nodeID": nodeID, "address": address, "port": port, "timestamp": timestamp}
```

4. 文件路径管理服务器 directoryServer 删除文件路径——deleteFileMapping 方法

```
    def deleteFileMapping(docname):
    '''
    : 从文件路径中删除文件
    : docname: 待删除文件名
    '''
    del FILE_ADDRESS[docname]
```

5. 文件路径管理服务器 directoryServer 的 SocketServer 框架请求处理组件——ThreadedHandler

```
class ThreadedHandler(socketserver.BaseRequestHandler):
    '''
    : SocketServer 网络服务框架组件
    : 继承自类 socketserver.BaseRequestHandler，该组件协调处理本文件路径服务器与客户端之间的通信，负责处理客户端发来的请求报文
    '''
    def handle(self):
        message = self.request.recv(1024)
        message = json.loads(message)
        requestType = message['request']
```

```
        response = ""

        #1. 处理客户端发来的 open 指令报文
        #-- open 报文的处理步骤较为简单，如下所示：
        #-- a. 提取报文中的文件名
        #-- b. 根据报文中的目标文件名，使用文件路径服务器的 getFileAddress 方法，获得文件具体位置信息（包括文件所在文件服务器地址，文件服务器端口等）
        #-- c. 根据上一步骤中得到的位置信息，生成一个 open 请求报文，发送给文件服务器以读取其上的文件
        #-- d. 从文件服务器获得响应报文，确认文件已经打开
        if requestType == "open":
            if fileExistsTest(message['docname']):
                fileinfo = getFileAddress(message['docname'])
                response = json.dumps({
                    "response": "open-exists",
                    "docname": message['docname'],
                    "isFile": True,
                    "address": fileinfo['address'],
                    "port": fileinfo['port'],
                    "timestamp": fileinfo['timestamp']
                })
            else:
                fileinfo = getRandomServer()
                response = json.dumps({
                    "response": "open-null",
                    "docname": message['docname'],
                    "isFile": False,
                    "uuid": fileinfo[0],
                    "address": fileinfo[1]['address'],
                    "port": fileinfo[1]['port']
                })

        #2. 处理客户端发来的 close 指令报文
        #-- close 报文的处理步骤较为简单，如下所示：
        #-- a. 提取报文中的文件名
        #-- b. 根据报文中的目标文件名，使用文件路径服务器的 getFileAddress 方法，获得文件具体位置信息（包括文件所在文件服务器地址，文件服务器端口等）
        #-- c. 根据上一步骤中得到的位置信息，生成一个 close 请求报文，发送给文件服务器以关闭其上的文件
        #-- d. 从文件服务器获得响应报文，确认文件已经关闭
        elif requestType == "close":
            response = json.dumps({
                "response": "close",
                "docname": message['docname'],
```

```
                "isFile": True
            })
        elif requestType == "read":
            if fileExistsTest(message['docname']):
                fileinfo = getFileAddress(docname)
                response = json.dumps({
                    "response": "read-exists",
                    "docname": message['docname'],
                    "isFile": True,
                    "address": fileinfo['address'],
                    "port": fileinfo['port'],
                    "timestamp": fileinfo['timestamp']
                })
            else:
                response = json.dumps({
                    "response": "read-null",
                    "docname": message['docname'],
                    "isFile": False
                })

        #3. 处理客户端发来的 write 指令报文
        #-- 具体处理步骤和上面的 open 报文类似，此处不再赘述
        elif requestType == "write":
            print(message['docname'])
            print(FILE_ADDRESS)
            if fileExistsTest(message['docname']):
                print("write if")
                fileinfo = getFileAddress(message['docname'])
                response = json.dumps({
                    "response": "write-exists",
                    "docname": message['docname'],
                    "isFile": True,
                    "uuid": fileinfo['uuid'],
                    "address": fileinfo['address'],
                    "port": fileinfo['port'],
                    "timestamp": message['timestamp']
                })
            else:
                fileinfo = getRandomServer()
                FILE_ADDRESS[message['docname']] = {"uuid": fileinfo[0],
"address": fileinfo[1]['address'], "port": fileinfo[1]['port'],
"timestamp": message['timestamp']}
                print(FILE_ADDRESS)
                response = json.dumps({
```

```
                    "response": "write-null",
                    "docname": message['docname'],
                    "isFile": False,
                    "uuid": fileinfo[0],
                    "address": fileinfo[1]['address'],
                    "port": fileinfo[1]['port'],
                    "timestamp": message['timestamp']
                })
        elif requestType == "dfileinfojoin":
            nodeID = message['uuid']
            if(nodeID == ""):
                nodeID = str(uuid.uuid4())
            FILE_SERVER[nodeID] = {"address": message['address'], "port":
message['port']}
            response = json.dumps({"response": requestType, "uuid":
nodeID})
            #print(FILE_SERVER)
        else:
            response = json.dumps({"response": "error", "error":
requestType+"为非法指令"})

        self.request.sendall(response.encode())
```

3.4. 锁服务器 lockingServer 的具体实现

1. 锁服务器 lockingServer 初始化

锁服务器的初始化需要预先设定一个 IP 地址及端口，并初始化一个加锁文件的"文件名-客户 ID"键值对表。

```
ADDRESS = "127.0.0.1"
PORT = 8888

LOCK_TIMEOUT = 30

LOCK_LIST = {}      #加锁文件表：加锁文件的"文件名-客户 ID"键值对表
```

2. 锁服务器 lockingServer 查询文件锁——lockExistsTest 与 getLockClient 方法

```
    def lockExistsTest(docname):
    '''
    : 判断给定文件名的文件是否存在锁
    : docname: 文件名
    '''
```

```
    #1. 判断加锁文件表中是否含有该文件名，含有即代表该文件上有锁，否则该文件上无锁
    if docname in LOCK_LIST:
        return True
    else:
        return False

def getLockClient(docname):
    '''
    : 返回给定文件名的文件的锁信息（即加锁的客户 ID）
    : docname: 文件名
    '''
    #1. 检查该文件是否含有锁
    if lockExistsTest(docname):
        return LOCK_LIST[docname]
    #2. 若该文件不含有锁，则返回 None；否则，返回正在占用（即给文件加锁）的客户 ID
    else:
        return None
```

3. 锁服务器 lockingServer 添加文件锁——addLock 方法

```
    def addLock(docname, clientid, timestamp, timeout):
    '''
    : 为特定用户给指定文件加锁
    : docname: 文件名
    : clientid: 客户 ID
    : timestamp: 时间戳
    : timeout: 加锁最长时限（防止死锁）
    '''
    #1. 在加锁文件表中添加该文件的锁，以及锁的具体信息
    LOCK_LIST[docname] = {"clientid": clientid, "timestamp": timestamp, "timeout": timeout}
```

4. 锁服务器 lockingServer 删除文件锁——delLock 方法

```
def delLock(docname):
    '''
    : 为给指定文件名的文件解锁
    : docname: 文件名
    '''
    #1. 在加锁文件表中删除相关锁的记录以完成解锁
    del LOCK_LIST[docname]
```

5. 锁服务器 lockingServer 框架请求处理组件——ThreadedHandler

```
class ThreadedHandler(socketserver.BaseRequestHandler):
    '''
    : SocketServer 网络服务框架组件
    : 继承自类 socketserver.BaseRequestHandler，该组件协调处理客户端与锁服务器之间的通信，负责处理客户端发来的请求报文
    : 需要特别注意，锁控制服务是由客户端进行的
    '''
    def handle(self):
        msg = self.request.recv(1024)

        msg = json.loads(msg)
        requestType = msg['request']

        print("Request type = " + requestType)

        response = ""

        if requestType == "checklock":
            if lockExistsTest(msg['docname']):
                print("Check lock -> lock exists")
                timestamp = time.time()
                fs = getLockClient(msg['docname'])


                if fs['timestamp']+fs['timeout'] < timestamp:
                    print("Lock has timed out")
                    print(fs['timestamp']+fs['timeout'])
                    print(timestamp)

                    delLock(msg['docname'])
                    response = json.dumps({
                        "response": "unlocked"
                    })

                elif msg['clientid'] == fs['clientid']:
                    print("Check lock -> lockowned")
                    response = json.dumps({
                        "response": "lockowned",
                        "docname": msg['docname'],
                        "timestamp": fs['timestamp'],
                        "timeout": fs['timeout']
                    })

                else:
```

```
                print("Check lock -> locked")
                response = json.dumps({
                    "response": "locked",
                    "docname": msg['docname'],
                    "timestamp": fs['timestamp'],
                    "timeout": fs['timeout']
                })
            else:
                response = json.dumps({
                    "response": "unlocked"
                })

        elif requestType == "obtainlock":
            if lockExistsTest(msg['docname']):
                print("Obtain lock -> lock exists")

                fs = getLockClient(msg['docname'])
                timestamp = time.time()

                if fs['timestamp']+fs['timeout'] < timestamp:
                    print("Obtain lock -> lock timed out, obtain again")

                    print(fs['timestamp']+fs['timeout'])
                    print(timestamp)

                    delLock(msg['docname'])
                    addLock(msg['docname'], msg['clientid'], timestamp,
LOCK_TIMEOUT)

                    response = json.dumps({
                        "response": "lockgranted",
                        "docname": msg['docname'],
                        "timestamp": fs['timestamp'],
                        "timeout": fs['timeout']
                    })

                elif msg['clientid'] == fs['clientid']:
                    print("Check lock -> lockowned")
                    timestamp = time.time()
                    response = json.dumps({
                        "response": "lockregranted",
                        "docname": msg['docname'],
                        "timestamp": timestamp,
                        "timeout": LOCK_TIMEOUT
```

```
                })
            else:
                print("Obtain lock -> locked already")
                response = json.dumps({
                    "response": "locked",
                    "docname": msg['docname'],
                    "timestamp": fs['timestamp'],
                    "timeout": fs['timeout']
                })
        else:
            print("Obtain lock -> lock granted")
            timestamp = time.time()
            addLock(msg['docname'], msg['clientid'], timestamp,
LOCK_TIMEOUT)

            response = json.dumps({
                "response": "lockgranted",
                "docname": msg['docname'],
                "clientid": msg['clientid'],
                "timestamp": timestamp,
                "timeout": LOCK_TIMEOUT
            })
    else:
        response = json.dumps({"response": "Error", "error":
requestType+" is not a valid request"})

    self.request.sendall(response.encode())
```

4. 实验结果演示

在已经配置好 python 基础环境以及库环境的情况下，在 linux 平台上直接运行 run.sh 即可进行测试。

这里给出一个较为简单的测试示例，如下所示。

1. 客户 1 创建文件 hello_world 并写入"Hello_World!"

```
Please enter a request type [open/close/checklock/obtainlock/read/write] or type
 exit to quit: write
Please enter the filename: hello_world.txt
Please enter the file contents to write: Hello World!
```

2. 客户 2 打开并读取 hello_world 文件

```
Please enter a request type [open/close/checklock/obtainlock/read/write] or type
 exit to quit: open
Please enter the filename: hello_world.txt
{"timestamp": 1547050092.149348, "response": "open-exists", "address": "127.0.0.
1", "filename": "hello_world.txt", "isFile": true, "port": 54969}
```

```
Please enter a request type [open/close/checklock/obtainlock/read/write] or type
 exit to quit: read
Please enter the filename: hello_world.txt
Read 'hello_world.txt' from cache!
{'data': 'Hello World!', 'timestamp': 1547050092.149348, 'request': 'write', 'cl
ientid': 'ca89cb3b-ddfa-4048-b308-c7ffac7d747a', 'filename': 'hello_world.txt'}
```

3. 客户 1 查询 hello_world 文件的锁状态

这时没有用户对该文件加锁

```
Please enter a request type [open/close/checklock/obtainlock/read/write] or type
 exit to quit: checklock
Please enter the filename: hello_world.txt
{"response": "unlocked"}
```

4. 客户 2 对 hello_world 文件加锁

```
Please enter a request type [open/close/checklock/obtainlock/read/write] or type
 exit to quit: obtainlock
Please enter the filename: hello_world.txt
{"timestamp": 1547050389.073138, "timeout": 30, "response": "lockgranted", "clie
ntid": "ca89cb3b-ddfa-4048-b308-c7ffac7d747a", "filename": "hello_world.txt"}
```

5. 客户 1 对文件 hello_world 查询锁状态

这时文件已经加锁，客户 1 无法进行写操作。

```
Please enter a request type [open/close/checklock/obtainlock/read/write] or type
 exit to quit: checklock
Please enter the filename: hello_world.txt
{"timestamp": 1547050891.631073, "response": "lockowned", "timeout": 30, "filena
me": "hello_world.txt"}
```