分布式系统课程设计

分布式文件系统

姓名：汤焯俊

班级：计科大数据班

学号：21307272

日期：2023年12月12日

目录：

题目要求

1. 题目：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，实验报告模板稍后在课程网站下载，提交时间为考试之后一周内。
2. 题目要求：
   - 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

我的解决思路是通过两个服务器`database`和`server`和一个客户端`client`的分布式文件系统，其基本信息如下：

- 基于XMLRPC
- 可创建，删除，访问文件，本系统以对txt文件的操作作为示例
- 实现了本地缓存功能
- 文件副本存放在多个服务器，多个副本实现瞬时一致性
- 可多用户登录，文件可以并行读写，登录时会进行加密，具有安全性

database

`database`是一个服务器，它的作用是确保server和client可以正常通信，并且它保存了一个数据库(由sqlite3实现)，其中保存了三个表，分别为`users`,`servers`,`files`，其作用是保存用户，服务器，文件的信息。其实现了若干个和数据库交互的函数，并且可以让server端和client端通过RPC调用来对数据库进行操作，例如保存用户的信息，读取文件的信息等等，它是用Python的`xmlrpc`模块的`SimpleXMLRPCServer`类实现的 。

server

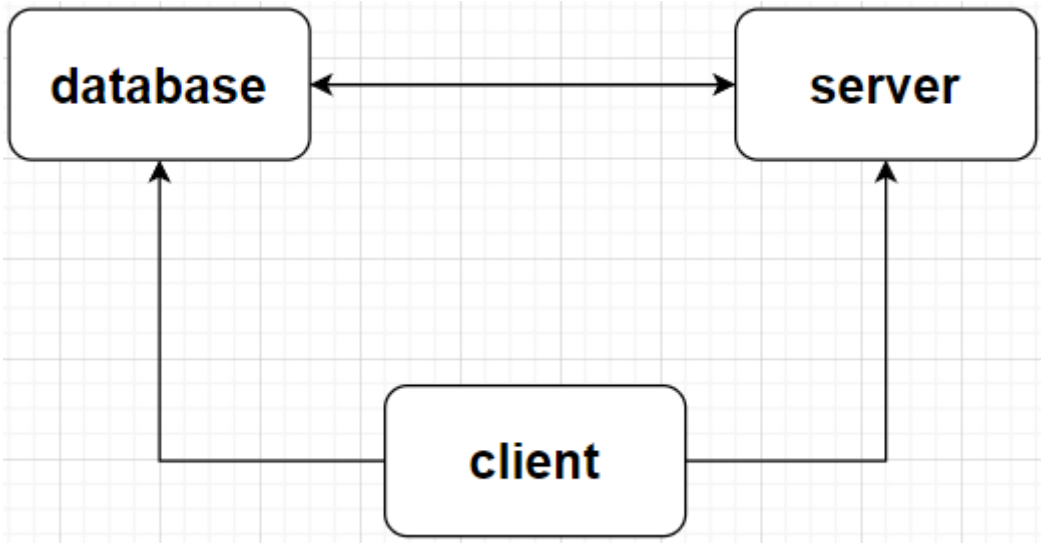
`server`也是一个服务器，服务器将用户上传的文件储存在云端，每个服务器在云端根目录中拥有一个和自己同名的文件夹，用于存放用户上传的文件，如果同时存在多个服务器的话，所有服务器都会存有文件的副本，对任一服务器上的任何文件进行操作的话，所有服务器上的文件都会被操作，这里保证了瞬时一致性。同样是用用Python的`xmlrpc`模块的`SimpleXMLRPCServer`类实现的 。

client

`client`是客户端，实现与用户直接交互的系统部分 ，它同时连接`database`和`server`，以实现对它们的操作。它实现了先注册再登录的用户系统，可以多用户登录，每个用户在本地根目录都有一个属于自己的文件夹，用于存放本地缓存，在读文件时，如果本地文件比云端文件要旧，则先从云端下载文件再读取，否则就可以直接在本地进行读取。而写文件时同样会先修改本地文件，再将其上传到云端。对于每个文件都有共享锁和排他锁，读写时要遵循锁的操作规范，避免了出现读写冲突。用户在登录时会根据明文密码计算出哈希密码和对应salt值，数据库内只会保存哈希密码和salt值，而不会保存明文密码，起到了安全性。

实现细节

三个部分的通信关系：



database

数据库服务器中保存了三个表，分别为`users`,`servers`,`files`，其格式如下(加粗为主键)：

- users 用户表

username	*password*	*salt*
text	*text*	*text*

- servers 服务器表

serverid	*address*
integer	*text*

- files 文件表

filename	***serverid***	*lastmodified*	*filehash*	*S_lock*	*X_lock*
text	*integer*	*float*	*text*	*integer*	*integer*

初始化

```
def init_user_table():
    cursor.execute('drop table if exists users;')
    connection.commit()
    cursor.execute('create table users (\
username text primary key,\
password text,\
salt text);')
    connection.commit()
def init_server_table():
    cursor.execute('drop table if exists servers;')
    connection.commit()
    cursor.execute('create table servers (\
serverid integer primary key,\
address text);')
    connection.commit()
def init_file_table():
    cursor.execute('drop table if exists files;')
    connection.commit()
    cursor.execute('create table files (\
filename text,\
serverid integer,\
lastmodified float,\
filehash text,\
S_lock integer check (S_lock ⩾ 0),\
X_lock integer check (X_lock = 0 or X_lock = 1),\
primary key (filename, serverid));')
    connection.commit()
def init_db():
    init_user_table()
    init_server_table()
    init_file_table()
    connection.commit()
```

对用户表的操作有：

- 添加一个用户

- 获取用户信息

```
def add_user(username, hash_password, salt):  # 添加一个用户
    try:
        cursor.execute('insert into users (username, password, salt) values (?, ?, ?);',
                       (username, str(base64.b64decode(hash_password), 'utf-8'), str(base64.b64decode(salt), 'utf-8')))
        connection.commit()
        return True
    except sqlite3.Error:
        return False
def get_user_info(username):  # 获取用户信息
    try:
        cursor.execute('select password, salt from users where username = ?;', (username, ))
        res = cursor.fetchone()
        return res
    except sqlite3.Error:
        return None
```

对服务器表的操作有：

- 添加新的服务器
- 删除服务器，一并删除其存的文件信息
- 获取所有服务器信息
- 输入服务器地址，返回服务器id
- 输入服务器id，返回服务器地址
- 获取所有服务器的地址

```
def add_server(server_id, address):  # 添加新的服务器
    try:
        cursor.execute('insert into servers (serverid, address) values (?, ?);', (server_id, address))
        connection.commit()
        return True
    except sqlite3.Error:
        return False
def delete_server(server_id):  # 删除服务器，一并删除其存的文件信息
    cursor.execute('delete from servers where serverid = ?;', (server_id, ))
    cursor.execute('delete from files where serverid = ?;', (server_id, ))
    connection.commit()
def get_server_info():  # 获取所有服务器信息
    try:
        cursor.execute('select serverid, address from servers;')
        res = cursor.fetchall()
        return res
    except sqlite3.Error:
        return []
def get_server_id(address):  # 输入服务器地址，返回服务器id
    try:
        cursor.execute('select serverid from servers where address = ?', (address, ))
        res = cursor.fetchone()
        return res
    except sqlite3.Error:
```

```
        return []
def get_server_address(serverid):  # 输入服务器id，返回服务器地址
    try:
        cursor.execute('select address from servers where serverid = ?;', (serverid, ))
        res = cursor.fetchone()
        return res
    except sqlite3.Error:
        return []
def get_all_server_addresses():  # 获取所有服务器的地址
    try:
        cursor.execute('select address from servers')
        res = cursor.fetchall()
        return [address for (address, ) in res]
    except sqlite3.Error:
        return []
```

对文件表的操作有：

- 添加新文件，如果已存在则覆盖
- 给定文件名和所在服务器，获取共享锁
- 给定文件名和所在服务器，归还共享锁
- 给定文件名和所在服务器，获取排他锁
- 给定文件名和所在服务器，归还排他锁
- 给定文件名和所在服务器，返回共享锁和排他锁的情况
- 给定服务器id，获取该服务器的所有文件信息
- 给定服务器id和文件名，返回文件哈希值
- 给定服务器id和文件名，删除该文件

```
def add_file(file_list):  # 添加新文件，如果已存在则覆盖
    try:
        cursor.executemany('insert or replace into files (filename, serverid, lastmodified, filehash, S_lock, X_lock) values (?, ?, ?, ?, ?, ?)', file_list)
        connection.commit()
        return True
    except sqlite3.Error:
        return False

def lock_S(serverid, filename):  # 给定文件名和所在服务器，获取共享锁
    try:
        cursor.execute('update files set S_lock = S_lock + 1 where serverid = ? and filename = ?;', (serverid, filename))
        connection.commit()
        return True
    except sqlite3.Error:
        return False

def unlock_S(serverid, filename):  # 给定文件名和所在服务器，归还共享锁
    try:
        cursor.execute('update files set S_lock = S_lock - 1 where serverid = ? and filename = ?;', (serverid, filename))
        connection.commit()
        return True
    except sqlite3.Error:
        return False
```

```python

def lock_X(serverid, filename):  # 给定文件名和所在服务器，获取排他锁
    try:
        cursor.execute('update files set X_lock = X_lock + 1 where serverid = ? and filename = ?;', (serverid, filename))
        connection.commit()
        return True
    except sqlite3.Error:
        return False

def unlock_X(serverid, filename):  # 给定文件名和所在服务器，归还排他锁
    try:
        cursor.execute('update files set X_lock = X_lock - 1 where serverid = ? and filename = ?;', (serverid, filename))
        connection.commit()
        return True
    except sqlite3.Error:
        return False

def get_lock(serverid, filename):  # 给定文件名和所在服务器，返回共享锁和排他锁的情况
    try:
        cursor.execute('select S_lock, X_lock from files where serverid = ? and filename = ?;', (serverid, filename))
        res = cursor.fetchone()
        return res
    except sqlite3.Error:
        return None

def get_file_infos(serverid):  # 输入服务器id，获取该服务器的所有文件信息
    try:
        cursor.execute('select filename, serverid, lastmodified, filehash from files\
                        where serverid = ?;', (serverid, ))
        res = cursor.fetchall()
        return res
    except sqlite3.Error:
        return []

def get_one_file_hash(serverid, filename):  # 给定服务器id和文件名，返回文件哈希值
    try:
        cursor.execute('select filehash from files where serverid = ? and filename = ?;', (serverid, filename))
        res = cursor.fetchone()
        return res
    except sqlite3.Error:
        return []

def delete_file(serverid, filename):  # 给定服务器id和文件名，删除该文件
    try:
        cursor.execute('delete from files where serverid = ? and filename = ?;', (serverid, filename))
        connection.commit()
        return True
    except sqlite3.Error:
        return False

```

主函数

```
if __name__ == '__main__':
    server_counter = 0
    connection = sqlite3.connect('info.db')
    cursor = connection.cursor()
    init_db()
    with SimpleXMLRPCServer(database_info, allow_none=True) as server:
        # 创建一个XML-RPC服务器，并注册多个函数来处理远程调用请求
        server.register_function(add_user)
        server.register_function(add_server)
        server.register_function(add_file)
        server.register_function(delete_server)
        server.register_function(delete_file)
        server.register_function(get_user_info)
        server.register_function(get_file_infos)
        server.register_function(get_all_server_addresses)
        server.register_function(get_server_address)
        server.register_function(get_server_id)
        server.register_function(get_one_file_hash)
        server.register_function(get_server_info)
        server.register_function(get_lock)
        server.register_function(lock_S)
        server.register_function(unlock_S)
        server.register_function(lock_X)
        server.register_function(unlock_X)
        try:
            print('Welcome to Tangzhj\'s database.')
            server.serve_forever()  # 启动服务器并开始监听端口上的请求
        except KeyboardInterrupt:
            print('Good bye.')
            connection.close()
```

server

其实现的函数及功能如下，用于对云端服务器上的文件进行直接操作，并将操作结果返回给用户。

```
def mktxt(name, content):  # 写文件
    path = root_dir / (name + '.txt')
    try:
        with path.open(mode='w') as f:
            # 写入数据
            f.write(content)
        return 'success'
    except:
        return ''
def deltxt(name):  # 删除文件
    path = root_dir / (name + '.txt')
    try:
        path.unlink()
    except FileNotFoundError:
        return 'Txt does not exist'
    except OSError as e:
        return 'An error occurred while deleting the txt'
    return 'success'
def print_cloud_filename():  # 获取云端服务器中所有文件名
```

```
    res = []
    for txt in root_dir.rglob('*'):
        res.append(txt.name)
    return res
def get_txt_content(name):  # 获取txt文件内容
    path = root_dir / name
    try:
        content = path.read_text()
        return content
    except IOError:
        print(f"Unable to open txt {path}")
        return False
def calculate_file_hash(file_path):  # 计算文件的哈希值
    with open(file_path, 'rb') as file:
        data = file.read()
        hash_value = hashlib.sha256(data).hexdigest()
        return hash_value
```

主函数如下

```
if __name__ == '__main__':
    root_dir =  Path(__file__).parent / 'cloud_server'  # 云端服务器

    # 添加初始化参数
    parser = argparse.ArgumentParser()
    parser.add_argument('server_id', help='ID of the file server.', type=int)
    parser.add_argument('port', help='Port of the file server.', type=int)
    args = parser.parse_args()
    with SimpleXMLRPCServer(('localhost', args.port)) as server:
        # 注册函数
        server.register_function(mktxt)
        server.register_function(deltxt)
        server.register_function(print_cloud_filename)
        server.register_function(get_txt_content)
        # 服务器地址
        server_address = 'http://{}:{}'.format(server.server_address[0],
server.server_address[1])
        # 更新数据库中服务器的信息
        server_registered = False
        with ServerProxy(database_url, allow_none=True) as proxy:
            server_registered = proxy.add_server(args.server_id,
server_address)
        if server_registered:
            # 更新数据库中的文件信息
            root_dir = root_dir / str(args.server_id)

            if not root_dir.exists():
                root_dir.mkdir(parents=True)
            print('Welcome to Tangzhj\'s server.')
            print('Initializing cloud server for files in "
{}"...'.format(str(root_dir)))
            # 将云端服务器中已有的文件的信息添加进数据库中
            files_registered = False
            file_list = []
            for filename in os.listdir(root_dir):
                path = os.path.join(root_dir, filename)
                lastmodified = os.path.getmtime(path)
```

```
                filehash = calculate_file_hash(path)
                file_list.append(tuple([filename, args.server_id,
lastmodified, filehash, 0, 0]))
            with ServerProxy(database_url, allow_none=True) as proxy:
                files_registered = proxy.add_file(file_list)
            if files_registered:
                print('Serving file cloud server on
{}.'.format(server.server_address))
                if len(file_list) != 0:
                    print('Successfully synchronized existing files:')
                    for file_info in file_list:
                        print(file_info[0])
                try:
                    server.serve_forever()
                except KeyboardInterrupt:
                    with ServerProxy(database_url, allow_none=True) as proxy:
                        # 关闭服务器，清理数据库中有关该服务器的信息和文件
                        proxy.delete_server(args.server_id)
                        print('Shutting down the server and cleaning up the
files in database.')
            else:
                print('Failed file registration.')
        else:
            print('Failed server registration.')
```

client

其实现的函数及功能如下，用于注册和登录，对本地文件的操作及将操作传递到云端，支持多用户并行操作。

```
def calculate_file_hash(file_path):  # 计算文件哈希值
    with open(file_path, 'rb') as file:
        data = file.read()
        hash_value = hashlib.sha256(data).hexdigest()
        return hash_value
def sign_up(username, password):  # 注册用户
    if len(password) > 72:
        print('Password must be less than 72 characters.')
        return
    # salt值属于随机值。用户注册时，系统用来和用户密码进行组合而生成的随机数值
    salt = bcrypt.gensalt()
    hash_password = bcrypt.hashpw(password.encode('utf-8'), salt)  # 将明文密
码哈希化，增加安全性，input的是字节串，哈希后的密码中会包含盐值的信息
    # 储存哈希化之后的密码和盐值，input的是base64格式的字符串
    if proxy.add_user(username, base64.b64encode(hash_password).decode('utf-
8'),
                      base64.b64encode(salt).decode('utf-8')):
        print('salt:{} hash_passpord:{}'.format(salt, hash_password))
        print('User {} created successfully.'.format(username))
    else:
        print('User already exists, Could not create user
{}.'.format(username))
def login(username, password):  # 登录
    results = proxy.get_user_info(username)
    if results is None:
        print('Username does not exist.')
    else:
```

```
        hash_password = results[0].encode('utf-8')  # 转为字节串
        if bcrypt.checkpw(password.encode('utf-8'), hash_password):  # 可以自动从哈希后密码中提取盐值，无需显式传递
            print('Logged in as {}.'.format(username))
            return App(username)
        else:
            print('Wrong password.')
    return None
def mktxt(path, name, content):  # 创建txt文件
    name = name + '.txt'
    txt_path = path / name
    try:
        with txt_path.open(mode='w') as f:
            # 写入数据
            f.write(content)
        print('The local txt file was written successfully.')
        update(path, name, 'upload')
    except:
        print('An error occurred while making the local txt or uploading the txt.')
def deltxt(path, name):  # 删除txt文件
    name = name + '.txt'
    txt_path = path / name
    try:
        txt_path.unlink()
        print('The local txt file was deleted successfully.')
        update(path, name, 'delete')
    except FileNotFoundError:
        print("Txt does not exist")
    except OSError as e:
        print("An error occurred while deleting the txt:", e)
def get_txt_content(path):  # 获取txt文件内容
    try:
        content = path.read_text()
        return content
    except IOError:
        print(f"Unable to open txt {path}")
        return False
def readtxt(path, name):  # 读文件
    name = name + '.txt'
    serverid = proxy.get_server_info()[0][0]  # 随便获取一个服务器id
    print(serverid, name)
    lock = proxy.get_lock(serverid, name)
    xlock = lock[1]
    if xlock == 1:  # 如果当前文件被加上了排他锁，则等待
        print('The txt is being written, please wait...')
    while proxy.get_lock(serverid, name)[1]:
        pass
    proxy.lock_S(serverid, name)  # 给文件加上一个共享锁
    if not (path / name).exists():  # 如果本地没有这个文件，则从服务器下载
        print('The txt is not existed loaclly, so we download it from server.')
        download(serverid, name, path)
    else:  # 如果本地有文件，但是和服务器的不一致，则从服务器下载更新
        local_filehash = calculate_file_hash(path / name)
        cloud_filehash = proxy.get_one_file_hash(serverid, name)[0]
        if (local_filehash != cloud_filehash):
```

```
            print('Local files are not the same as cloud files, so we update it from server.')
            download(serverid, name, path)
    content = get_txt_content(path / name)
    if content == False:
        print('Unable to read {}'.format(name))
    else:
        print('The content of {} is {}'.format(name, content))
    proxy.unlock_S(serverid, name)  # 解开一个共享锁
def upload_all(path):  # 将本地的所有txt都上传到服务器
    for txt in path.rglob('*'):
        update(path, txt.name, 'upload')
def update(path, name, op):  # 将本地的操作更新到服务器
    addresses = proxy.get_all_server_addresses()
    if op == 'upload':  # 上传操作
        if name.endswith('.txt'):
            content = get_txt_content(path / name)
            for address in addresses:  # 遍历所有服务器地址
                serverid = proxy.get_server_id(address)[0]
                lock = proxy.get_lock(serverid, name)
                if lock != None:
                    slock = lock[0]
                    xlock = lock[1]
                    if slock != 0 or xlock != 0:  # 如果该文件已被加上任意一把锁，则等待
                        print('The txt is being read or written, please wait...')
                        while True:
                            lock = proxy.get_lock(serverid, name)
                            slock = lock[0]
                            xlock = lock[1]
                            if slock == 0 and xlock == 0:
                                break
                    proxy.lock_X(serverid, name)  # 给文件加上一把排他锁
                with ServerProxy(address, allow_none=True) as server_proxy:
                    server_id = proxy.get_server_id(address)[0]
                    back = server_proxy.mktxt(name[:-4], content)
                    print('Server_id:{}'.format(str(server_id)), end=' ')
                    if back == 'success':
                        lastmodified = os.path.getmtime(os.path.join(path, name))
                        filehash = calculate_file_hash(os.path.join(path, name))
                        proxy.add_file([tuple([name, server_id, lastmodified, filehash, 0, 1])])
                        print('Successfully upload.')
                    else:
                        print('Fail to upload.')
                proxy.unlock_X(serverid, name)  # 解开一个排他锁
    elif op == 'delete':  # 删除操作
        if name.endswith('.txt'):
            for address in addresses:  # 遍历所有服务器地址
                serverid = proxy.get_server_id(address)[0]
                lock = proxy.get_lock(serverid, name)
                slock = lock[0]
                xlock = lock[1]
                if slock != 0 or xlock != 0:  # 如果该文件已被加上任意一把锁，则等待
```

```
                    print('The txt is being read or written, please wait...')
                    while True:
                        lock = proxy.get_lock(serverid, name)
                        slock = lock[0]
                        xlock = lock[1]
                        if slock == 0 and xlock == 0:
                            break
                proxy.lock_X(serverid, name)  # 给文件加上一把排他锁
                with ServerProxy(address, allow_none=True) as server_proxy:
                    server_id = proxy.get_server_id(address)[0]
                    back = server_proxy.deltxt(name[:-4])
                    print('Server_id:{}'.format(str(server_id)), end=':')
                    if back == 'success':
                        proxy.delete_file(server_id, name)
                        print('Successfully delete.')
                    else:
                        print(back)
                # 文件已经被删了，所以不需要解开排他锁了
def download(server_id, name, local_path):  # 从服务器下载文件
    address = proxy.get_server_address(server_id)[0]
    txt_list = proxy.get_file_infos(server_id)
    try:
        for info in txt_list:
            if info[0] == name:
                with ServerProxy(address, allow_none=True) as server_proxy:
                    content = server_proxy.get_txt_content(name)
                local_path = Path(local_path) / name
                with local_path.open(mode='w') as f:
                    # 写入数据
                    f.write(content)
                print('The txt file was downloaded successfully.')
    except:
        print('An error occurred while downloading the txt.')
def print_local_filename(path):  # 打印本地文件名
    for txt in path.rglob('*'):
        print(txt.name)
def print_cloud_filename():  # 打印服务器所有文件信息
    addresses = proxy.get_all_server_addresses()
    print('{0:25s} {1:7s} {2}'.format('File Name', 'Server', 'Last Modified Time'))
    for address in addresses:
        server_id = proxy.get_server_id(address)[0]
        txt_list = proxy.get_file_infos(server_id)
        for info in txt_list:
            modified_time_str = datetime.datetime.fromtimestamp(info[2]).strftime('%Y-%m-%d %H:%M:%S')
            print('{0:25s} {1:7s} {2}'.format(info[0], str(info[1]), modified_time_str))
```

还实现了一个App类，用于一体化管理用户的操作输入

```
class App(object):
    def __init__(self, username):
        self.username = username
        self.root_dir =  Path(__file__).parent / 'local_cache / username  # 用户的本地缓存
```

```
        if not self.root_dir.exists():
            self.root_dir.mkdir()
    def print_option(self):  # 打印操作教程
        print('OPTIONS')
        print('- ls')
        print('- server_ls')
        print('- mktxt <txt_name> <content>')
        print('- deltxt <txt_name>')
        print('- readtxt <txt_name>')
        print('- upload')
        print('- download <server_id> <txt_name>')
        print('- exit')
    def main_loop(self):
        print('Current user:', self.username)
        self.print_option()
        print('Type \'help\' to get tutorial')
        while True:
            print('Current user:', self.username)
            command = str(input('$ ')).split(' ')
            if command[0] == 'ls' and len(command) == 1:
                print_local_filename(self.root_dir)
            elif command[0] == 'server_ls' and len(command) == 1:
                print_cloud_filename()
            elif command[0] == 'mktxt' and len(command) == 3:
                mktxt(self.root_dir, command[1].strip(), command[2])
            elif command[0] == 'deltxt' and len(command) == 2:
                deltxt(self.root_dir, command[1].strip())
            elif command[0] == 'readtxt' and len(command) == 2:
                readtxt(self.root_dir, command[1].strip())
            elif command[0] == 'upload' and len(command) == 1:
                upload_all(self.root_dir)
            elif command[0] == 'download' and len(command) == 3:
                download(int(command[1].strip()), command[2].strip() +
'.txt', self.root_dir)
            elif command[0] == 'help' and len(command) == 1:
                self.print_option()
            elif command[0] == 'exit':
                break
            else:
                print('Invalid Command.')
```

最后，主函数如下

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('mode', help='Client mode, "signup" or "login".',
type=str)
    parser.add_argument('username', help='Username of the user.', type=str)
    parser.add_argument('password', help='Password of the user.', type=str)
    args = parser.parse_args()
    proxy = ServerProxy(database_url, allow_none=True)
    if args.mode == 'signup':
        sign_up(args.username, args.password)
    elif args.mode == 'login':
        app = login(args.username, args.password)
        if app is not None:
            app.main_loop()
```

```
    else:
        print('Invalid operation.')
```

运行情况

启动运行情况

这里我通过同时开启两个服务器和登录两个不同的用户来展示功能（注：展示终端内的(pytorch)是我自己命名的conda虚拟环境名，本次实验的环境要求和pytorch没关系）

首先需要安装一个依赖库bcrypt

```
pip install bcrypt
```

然后启动数据库服务器

```
python database.py
```

现象如下

```
(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python database.py
Welcome to Tangzhj's database.
```

然后启动RPC服务器，其参数项要加上服务器id和端口号

```
python server.py <serverid> <port>
```

现象如下，启动两个服务器

```
(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python server.py 1 8888
Welcome to Tangzhj's server.
Initializing cloud server for files in "C:\Users\11735\Desktop\分布式系统\大作业\code\cloud_server\1"...
Serving file cloud server on ('127.0.0.1', 8888).
```

```
(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python server.py 2 8080
Welcome to Tangzhj's server.
Initializing cloud server for files in "C:\Users\11735\Desktop\分布式系统\大作业\code\cloud_server\2"...
Serving file cloud server on ('127.0.0.1', 8080).
```

然后注册并登录两个用户

```
python client.py signup <username> <password>
python client.py login <usernme> <password>
```

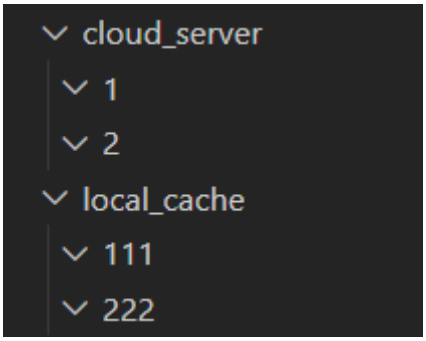
```
(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python client.py signup 111 123
salt:b'$2b$12$9HlucjOeJt.txf1jsPCFEu' hash_passpord:b'$2b$12$9HlucjOeJt.txf1jsPCFEuvgaTIqj1Q8xk8Q5uKWEiBHooow9N51a'
User 111 created successfully.

(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python client.py login 111 123
Logged in as 111.
Current user: 111
OPTIONS
- ls
- server_ls
- mktxt <txt_name> <content>
- deltxt <txt_name>
- readtxt <txt_name>
- upload
- download <server_id> <txt_name>
- help
- exit
Type 'help' to get tutorial
Current user: 111
$
```

```
(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python client.py signup 222 321
salt:b'$2b$12$oKBWL2I5GSjPCbRAedxSWu' hash_passpord:b'$2b$12$oKBWL2I5GSjPCbRAedxSWuYuuLeRbXFYHxf/D8HLsyyr0cWyTG7He'
User 222 created successfully.

(pytorch) C:\Users\11735\Desktop\分布式系统\大作业\code>python client.py login 222 321
Logged in as 222.
Current user: 222
OPTIONS
- ls
- server_ls
- mktxt <txt_name> <content>
- deltxt <txt_name>
- readtxt <txt_name>
- upload
- download <server_id> <txt_name>
- help
- exit
Type 'help' to get tutorial
Current user: 222
$
```

此时文件目录已被成功创建



登录之后，用户可以使用如下几条命令

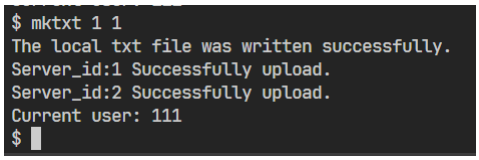
- **ls** 查看本地缓存的文件信息
- **server_ls** 查看云端服务器的所有文件信息
- **mktxt <txt_name> <content>** 创建或修改txt文件，先在本地操作后更新到云端
- **deltxt <txt_name>** 删除本地和所有云端服务器的某个txt文件
- **readtxt <txt_name>** 读txt文件，如果本地存在最新的版本，则直接在本地读取，否则先到服务器获取最新版本
- **upload** 上传本地所有文件到服务器中
- **download <server_id> <txt_name>** 从指定的服务器中下载指定名字的txt
- **help** 获取可用命令列表
- **exit** 登出

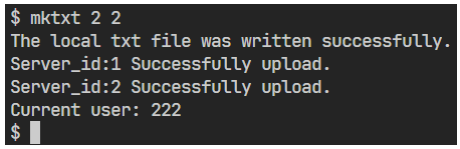
用户操作情况

下文图中的 `$` 符号后为测试命令

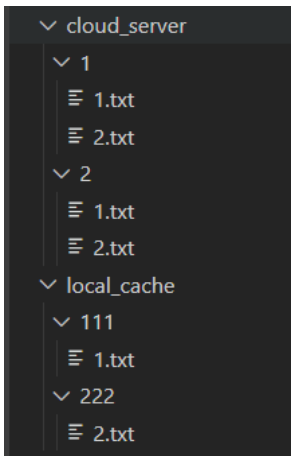
创建文件

由用户111创建1.txt，用户222创建2.txt

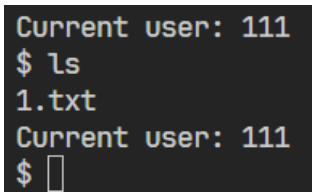


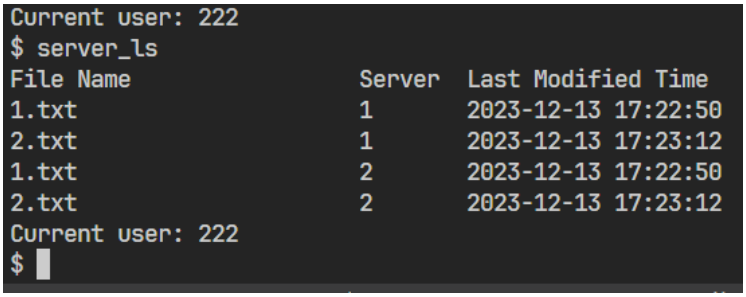


可以看到两个用户创建的文件在两个服务器里都有，同时也分别存在自己的本地缓存中



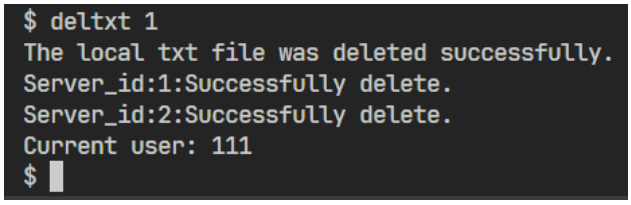
查看文件信息



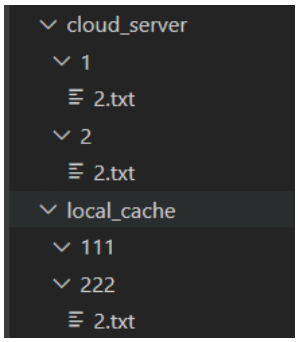


删除文件

由用户111删除1.txt



可以看到用户111的本地缓存和两个服务器内的1.txt都被删除了



读取文件

```
Current user: 111
$ readtxt 2
The txt is not existed loaclly, so we download it from server.
The txt file was downloaded successfully.
The content of 2.txt is 2
Current user: 111
$
```

可以看到此时111的本地缓存内没有2.txt，于是前往服务器下载再读取

如果222读取2.txt的话，由于本地缓存内存在最新的2.txt，所以直接在本地读，不需要前往服务器下载

```
Current user: 222
$ readtxt 2
The content of 2.txt is 2
Current user: 222
$
```

但是如果111修改了2.txt，那么222本地存的文件就是旧的，读之前需要前往服务器下载最新版本

```
Current user: 111
$ mktxt 2 333
The local txt file was written successfully.
Server_id:1 Successfully upload.
Server_id:2 Successfully upload.
Current user: 111
$
```

```
Current user: 222
$ readtxt 2
Local files are not the same as cloud files, so we update it from server.
The txt file was downloaded successfully.
The content of 2.txt is 333
Current user: 222
$
```

由于共享锁机制，两个用户同时读一个文件是可以被允许的，为了方便演示，我们在读取时获得共享锁后先加上一个5秒的延迟再进行读操作

```
Current user: 111
$ readtxt 2
The content of 2.txt is 333
Current user: 111
$
```

```
Current user: 222
$ readtxt 2
The content of 2.txt is 333
Current user: 222
$
```

上传和下载

111先创建一个1.txt，内容为1，222再创建一个1.txt，内容为456

```
Current user: 111
$ mktxt 1 1
The local txt file was written successfully.
Server_id:1 Successfully upload.
Server_id:2 Successfully upload.
Current user: 111
$
```

```
Current user: 222
$ mktxt 1 456
The local txt file was written successfully.
Server_id:1 Successfully upload.
Server_id:2 Successfully upload.
Current user: 222
$
```

111将文件上传到服务器

```
Current user: 111
$ upload
Server_id:1 Successfully upload.
Server_id:2 Successfully upload.
Server_id:1 Successfully upload.
Server_id:2 Successfully upload.
Current user: 111
$
```

此时111读取1.txt，读取到的内容是1，且不需要从服务器下载，证明了111前面的上传操作成功

```
Current user: 111
$ readtxt 1
The content of 1.txt is 1
Current user: 111
$
```

然后222从服务器中下载1.txt，再读取，可以看到本地文件内容已被改为1，证明下载操作成功

```
Current user: 222
$ download 1 1
The txt file was downloaded successfully.
Current user: 222
$ readtxt 1
The content of 1.txt is 1
Current user: 222
$
```

并行读写功能

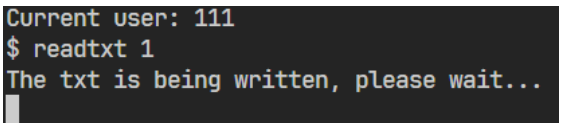
前文已经展示过可以共享读了，在这里先展示如果一个文件正在被读，那么它不能被写，为了方便演示，我们在读取时获得共享锁后先加上一个5秒的延迟再进行读操作

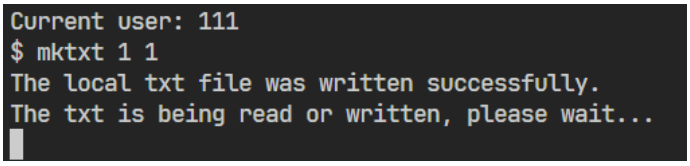
```
- help
- exit
Type 'help' to get tutorial
Current user: 111
$ readtxt 1
The content of 1.txt is 1
Current user: 111
$
```

```
Current user: 222
$ mktxt 1 21
The local txt file was written successfully.
The txt is being read or written, please wait...
Server_id:1 Successfully upload.
Server_id:2 Successfully upload.
Current user: 222
$
```

可以看到111先读取1.txt，在这5秒的延迟时，222修改1.txt的内容为21，由于共享锁的机制，在111读取完毕之前，不能修改这个文件，于是111能正确的读到文档的内容，读完之后才轮到222修改

接下来演示如果一个文件正在被写，那么这个文件不能被读写，为了方便演示，我们在写入时获得排他锁后先加上一个5秒的延迟再进行写操作





此时222正在修改这个文件，则111无论是读还是写这个文件都必须等待222完成。

遇到的问题

本次实验的实现思路并没有很复杂，但是具体上手编写代码时就发现有很多繁琐的细节需要考虑，没有想象中的这么容易实现。比如说对于文件的新旧程度，我一开始想着用最近修改时间来比较服务器文件和本地文件的新旧程度，但是我发现本地将文件上传后，服务器内的文件的最近修改时间就变成上传的这个时间了，虽然可以通过将本地这个文件的最近修改时间直接传递给数据库来规避上传时的这种错误，但是如果这样的话，服务器关闭再重启时这个最近修改时间依然会被记录为上传时间，于是我后面就改成了通过比较两个文件的哈希值来比较文件新旧程度，这样如果文件内容很大的话，会有一定的优化效果。

总结

在RPC的选择方面，在网上参考了一些资料后，我选择的是Python的 xmlrpc 模块的 SimpleXMLRPCServer类 。之前都是在课上听讲或者学习理论，还没有自己动手实现过，这次大作业完成了分布式文件系统，不仅让我对RPC有了更加深刻的认识，也让我对分布式系统这门课有了更深入的理解。