题目名称

分布式文件系统

姓名：陈欣宇

班级：人工智能与大数据

学号：21307347

日期：12.24

正文:

一、 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。

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；

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；

二、 解决思路

(1) 本次课程设计的实现思路主要按照 client-server 架构，编程语言为 python 代码，调用 python 的 xmlrpc 实现不同节点之间的 RPC 通信。

(2) 实现文件系统的功能有创建目录，删除目录，改变当前目录路径，写文件，读文件，删除文件 6 个功能。

(3) 为了比对文件一致性，为文件计算哈希值并储存起来，当再次需要读或写文件时会将该文件计算哈希值与存储哈希值比较，确定文件是否被篡改。

(4) 调用 python 的 sqlite3 用于存储和查询信息，因此在 client-server 架构基础上添加了 database 文件，用于调用 SQL 语句，并且起着维持服务器和客户端正常运行的功能。数据库主要设置了以下三个表：

服务器表(SERVERS)

server_id	address
服务器 ID	服务器 url，可被客户端查询调用

文件表(FILES)

Server_id	Path	Name	Is_backup	File_hash	lastmodified
服务器 ID	文件相对路径	文件名	是否为备份文件	哈希值	更新时间

缓存表(CACHES)

Server_id	Path	Cache_hash
服务器 ID	文件相对路径	哈希值

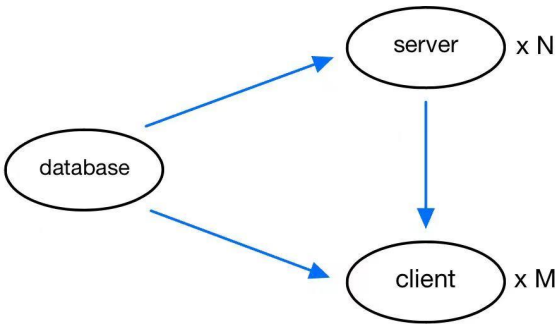
(5) 对于客户端缓存功能，我为每个用户创建了一个 cache 目录用于将存储最近文件，当用户读取文件时，都会将文件放到缓存中，并更新缓存表信息，在下次读取文件操作时，优先比对是否存在于缓存目录中（使用表查询即可比对），若缓存信息与文件原路径信息不符，则仍读取原路径文件，并更新缓存。对文件的写操作使用 write through 策略，即对写入本地缓存的同时也将数据写入分布式持久化存储中，这样也能够保持了不同用户之间多个缓存之间的一致性。

(6) 对于数据的可用性和文件系统性能，我为文件创建了副本，副本的更新主要在写文件后，以及在读文件时若检测到原文件受损，通过筛选可用副本，查找到第一个可用副本后将其作为原文件，并更新其他副本，保持副本之间的一致性。

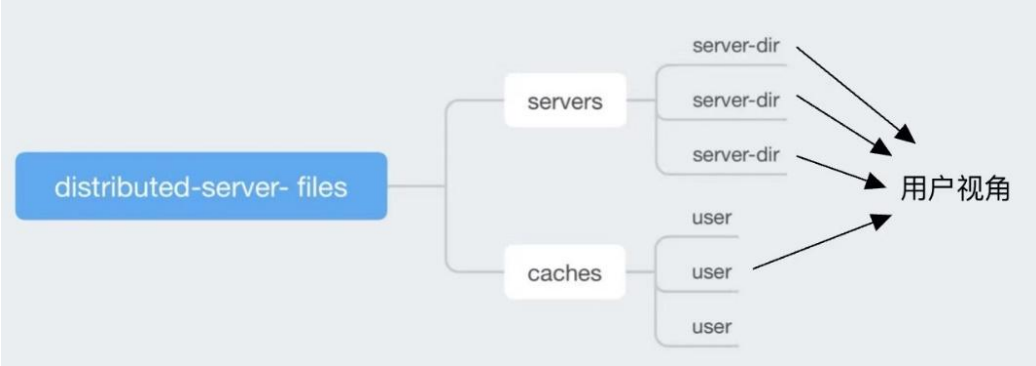
(7) 对于文件并行读写功能，我使用了两个列表表示共享锁和排他锁，存储着文件的相对路径，根据共享锁和排他锁的特性设置冲突条件，请求锁则判断是否冲突并添加文件路径，释放锁则删除相应的文件路径，读操作为路径申请共享锁，写操作申请排他锁。

三、 实现细节

架构如下：database 为 server 和 client 提供远程调用，server 为 client 提供远程调用



实现**目录结构**如下：文件散列在不同服务器节点中，用户能够查询到所有服务器上存在文件，以及每个用户都有一个本地存储目录 cache



1、Database.py 初始化：

(1) 连接 info.db 数据库

(2) SimpleXMLRPCServer 创建 RPC 服务器，供 client 和 server 远程调用以下函数，其中 data_info 选择在文件 config.py 中定义，通过 data_url 创建服务器代理对象即可调用 database.py 的函数

```
data_info = ('localhost', 9999)
data_url = 'http://{}:{}'.format(data_info[0], data_info[1])
```

```python
if __name__ == '__main__':
    server_counter = 0
    connection = sqlite3.connect('info.db')
    cursor = connection.cursor()
    init_db() #创建SERVERS/FILES/CACHES表
    with SimpleXMLRPCServer(data_info, allow_none=True) as server:
        server.register_function(insert_server)
        server.register_function(insert_files)
        server.register_function(delete_server)
        server.register_function(get_server_addresses)
        server.register_function(get_next_server)
        server.register_function(get_file_servers)
        server.register_function(get_file_backup_servers)
        server.register_function(get_file_backup_name)
        server.register_function(delete_file)
        server.register_function(get_file_infos)
        server.register_function(get_file_hash)
        server.register_function(get_cache_hash)
        server.register_function(delete_backup_file)
        server.register_function(remove_one_file)
        server.register_function(insert_cache)
        server.register_function(delete_cache)
        try:
            server.serve_forever()
        except KeyboardInterrupt:
            connection.close()
```

其中函数的主要功能介绍如下图展示

```
> def get_next_server():                      # 轮流取下一个address…
> def insert_server(server_id, address):      # 插入server…
> def insert_files(file_list):                # 批量插入file…
> def insert_cache(user_id,rel_path,hash):    # 插入cache…
> def delete_server(server_id):               # 删除server及有关file…
> def delete_file(rel_path):                  # 删除file及有关备份file…
> def remove_one_file(rel_path):              # 删除原file…
> def delete_backup_file(server_id,rel_path): # 删除备份file…
> def delete_cache(user_id,rel_path):         # 删除cache…
> def get_server_addresses():                 # 查找所有server的url…
> def get_file_servers(rel_path):             # 查找file原文件所在server的url…
> def get_file_backup_servers(rel_path):      # 查找file备份所在server的url…
> def get_file_backup_name(rel_path):         # 查找file备份的name…
> def get_file_infos(rel_path_list):          # 查找file列表对应的name、lastmodified列表…
> def get_file_hash(rel_path):                # 查找file原文件的hash…
> def get_cache_hash(user_id,rel_path):       # 查找cache的hash…
```

这些函数都是调用 SQL 语句，实现逻辑较简单，不一一展示，取有代表性进行介绍

其中 get_next_server()主要用于轮流取 server 为用户提供服务

```python
def get_next_server():                          # 轮流取下一个address
    global server_counter
    addresses = get_server_addresses()
    server_num = len(addresses)
    server_counter = (server_counter + 1) % server_num
    return addresses[server_counter]
```

下面的 get_server_address()获取所有 server 的 url 供用户，主要在查询所有文件时调用

```python
def get_server_addresses():                     # 查找所有server的url
    try:
        cursor.execute('SELECT address FROM SERVERS;')
        results = cursor.fetchall()
        return [address for (address, ) in results]
    except sqlite3.Error:
        return []
```

2、server.py 初始化

(1) 输入：调用 argparse 完善输入设置，要求在命令行键入运行

```
python server.py [server_id] [port]
```

(2) SimpleXMLRPCServer 创建 RPC 服务器，供 client 远程调用以下函数

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('server_id', help='ID of the server.', type=int)
    parser.add_argument('port', help='Port of the server.', type=int)
    args = parser.parse_args()

    with SimpleXMLRPCServer(('localhost', args.port)) as server:
        server.register_function(path_check)
        server.register_function(make_dir)
        server.register_function(del_dir)
        server.register_function(del_file)
        server.register_function(get_files)
        server.register_function(hash_check)
        server.register_function(read_path)
        server.register_function(write_new_file)
        server.register_function(update_backup)
        server.register_function(update_file)
        server.register_function(create_backup)
        server.register_function(initialize_file)
        server.register_function(hash_file)
```

(3) 更新 SERVERS 表信息（插入该 server 记录）

(4) 创建该服务器目录

(5) 遍历该服务器目录下文件，更新 FILES 表信息（插入该路径下所有文件记录）

```
server_url = 'http://{}:{}'.format(server.server_address[0], server.server_address[1])
server_ini = False
with ServerProxy(data_url, allow_none=True) as proxy:
    # 更新SERVERS表信息
    server_ini = proxy.insert_server(args.server_id,str(server_url))
if server_ini:
    # 根目录设置在当前工作目录下，创建服务器目录
    root = Path.cwd() / 'distributed_server_files'
    server_dir = root / ('server_'+str(args.server_id))
    if not server_dir.exists():
        server_dir.mkdir(parents=True)
    # 遍历原有文件，更新FILES表信息
    file_list = []
    for root, dirs, files in os.walk(str(server_dir)):
        for file in files:
            path = os.path.join(root, file)
            file_info = initialize_file(args.server_id, path, file)
            print('initialize file:', file_info[2])
            file_list.append(file_info)
    files_ini = False
    with ServerProxy(data_url, allow_none=True) as proxy:
        files_ini = proxy.insert_files(file_list)
    if files_ini:
        print("Server initialize successfully")
        try:
            server.serve_forever()  # 保持监听客户端请求
        except KeyboardInterrupt:
            with ServerProxy(data_url, allow_none=True) as proxy:
                proxy.delete_server(args.server_id)
```

3、client.py 初始化

(1) 同样设置输入格式为：

```
python client.py [user_id]
```

(2) 启动 datebase.py 的服务器代理

(3) 初始化一些待用参数

(4) 创建缓存目录

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('user_id', help='Userid of the user.', type=str)
    args = parser.parse_args()

    proxy = ServerProxy(data_url, allow_none=True)
    cd = ''
    shared_lock = []
    exclusive_lock = []

    root = Path.cwd() / 'distributed_server_files'
    cache_dir = root / ('cache_'+str(args.user_id))
    if cache_dir.exists(): shutil.rmtree(cache_dir)# 用户重新登录清空缓存
    cache_dir.mkdir(parents=True)
    main()
```

(5) 进入主函数，循环执行用户输入：

```
def main():
    global cd
    print('--<help> for commands')
    while True:
        print('Current directory:', str(args.user_id)+os.sep +cd)
        command = str(input('$ ')).split(' ')
        if command[0] == 'help':…
        elif command[0] == 'list':…
        elif command[0] == 'makedir' and len(command) == 2:      # 创建目录…
        elif command[0] == 'deletedir' and len(command) == 2:    # 删除目录…
        elif command[0] == 'delete' and len(command) == 2:       # 删除文件…
        elif command[0] == 'changedir':                          # 改变当前目录
        elif command[0] == 'read' and len(command) ==2:          # 读文件…
        elif command[0] == 'write' and len(command)==2:          # 写文件…
        elif command[0] == 'exit':…
        else:
            print('Invalid Command.')
```

4、文件系统功能的实现：

(1) 创建目录

用户输入 makedir <dir>执行操作：执行 make_dir 函数

```
elif command[0] == 'makedir' and len(command) == 2:      # 创建目录
    rel_dir = str(Path(cd) / command[1].strip())
    if make_dir(rel_dir): print('Successfully created "{}".'.format(rel_dir))
```

make_dir 函数实现：

① 首先查询所有服务器是否存在该目录

② 若存在则无需创建，都不存在该目录则调用一台服务器执行 mkdir 操作

```python
def make_dir(rel_dir):
    addresses = proxy.get_server_addresses()
    for address in addresses:
        with ServerProxy(address, allow_none=True) as server_pxy:
            path_exists,path_valid,_ = server_pxy.path_check(rel_dir)
            if path_valid and path_exists:
                print("Dir already existing !\n")
                return False
    address = proxy.get_next_server()
    with ServerProxy(address, allow_none=True) as server_pxy:
        return server_pxy.make_dir(rel_dir)
```

上面涉及到服务器 path_check 和 make_dir 函数

path_check 函数如下：返回路径"存在性"、"有效性"以及规范后的"相对路径"

```python
def path_check(rel_path):
    path = (server_dir / rel_path).resolve() #绝对路径(文件/目录)
    path_exists = path.exists()                          #存在
    path_valid = str(path).startswith(str(server_dir))  #有效
    new_rel_path = str(path).replace(str(server_dir),'')
    new_rel_path = new_rel_path[1:] if new_rel_path.startswith(os.sep) else new_rel_path
    return path_exists,path_valid,new_rel_path
```

server 端 make_dir 函数如下：检查路径是否有效和存在，执行 mkdir

```python
def make_dir(rel_path):
    path_exists,path_valid,new_rel_path =  path_check(rel_path)
    path = server_dir / new_rel_path
    if path_valid and not path_exists:
        path.mkdir(parents=True)
        return True
    return False
```

(2) 列出当前目录内容

用户键入 list，调用 list_file 函数:

遍历每个服务器，调用 get_file 函数，返回文件路径以及文件是否为目录项，根据文件路径在 FILES 表中搜索文件信息包括文件名和最近更新时间，打印出来

```python
def list_file(rel_dir):
    addresses = proxy.get_server_addresses()
    dir_paths = set()
    file_paths = set()
    print('=' * 80)
    print('{0:45s} {1:7s} {2}'.format('File Name', 'Type', 'Last Update'))
    print('-' * 80)
    for address in addresses:
        with ServerProxy(address, allow_none=True) as server_pxy:
            for is_dir, file_rel_path in server_pxy.get_files(rel_dir):#不返回备份文件路径
                if is_dir: dir_paths.add(file_rel_path)
                else: file_paths.add(file_rel_path)
    for dir_path in dir_paths:
        print('{0:45s} DIR'.format(Path(dir_path).name + '/'))
    for file_name, mod in proxy.get_file_infos(list(file_paths)):
        print('{0:45s} {1:7s} {2}'.format(file_name, Path(file_name).suffix[1:].upper(),datetime.datetime.fromtimestamp(mod)))
    print('=' * 80)
```

其中调用到服务端的 get_file 函数，用于返回该目录下所有文件及目录的路径，但不返回备份文件(隐藏显示):

首先检查路径有效和存在，之后通过 path.iterdir()遍历目录中内容，存储返回

```python
def get_files(rel_dir):
    path_exists,path_valid,new_rel_dir = path_check(rel_dir)
    dir_files = []
    if path_valid and path_exists:
        for it in (server_dir / new_rel_dir).iterdir():
            if '_backup' in it.name: continue # 不返回_backup文件
            dir_files.append((it.is_dir(),str(it.relative_to(server_dir))))
    return dir_files
```

(3) 删除目录

用户键入 deletedir <dir>后调用 del_dir 函数:

```
elif command[0] == 'deletedir' and len(command) == 2:    # 删除目录
    rel_dir = str(Path(cd) / command[1].strip())
    if del_dir(rel_dir): print('Successfully delete "{}".'.format(rel_dir))
```

调用 get_files 返回目录内容，若该目录不为空则返回不删除，空则调用服务器删除目录

```
def del_dir(rel_dir):
    addresses = proxy.get_server_addresses()
    for address in addresses:
        with ServerProxy(address, allow_none=True) as server_pxy:
            dir_files = server_pxy.get_files(rel_dir)
            if dir_files:# 目录不为空
                print("Dir Not Empty")
                return False
            server_pxy.del_dir(rel_dir)
    return True
```

server 端 del_dir 函数：例行检查并判断是否为目录，执行 rmdir()

```
def del_dir(rel_path):
    path_exists,path_valid,new_rel_path =  path_check(rel_path)
    path = server_dir / new_rel_path
    if path_valid and path_exists:
        if path.is_dir():
            path.rmdir()
            return True
        else: print("Not Dir Type\n")
    return False
```

(4) 改变当前目录

用户键入 changedir <dir-path>，如果没有<dir-path>则返回根路径，有则调用 change_dir 函数判断是否可用进入该目录

```
elif command[0] == 'changedir':
    if len(command) == 2:
        rel_dir = str(Path('') / command[1].strip())
        can, new_rel_dir = change_dir(rel_dir)
        if can:
            cd = new_rel_dir
            print('Change dir successfully.')
    elif len(command) == 1:
        cd = ''
        print('Change dir successfully.')
```

change_dir 函数：判断路径是否有效和存在即可

```
def change_dir(rel_dir):
    addresses = proxy.get_server_addresses()
    for address in addresses:
        with ServerProxy(address, allow_none=True) as server_pxy:
            path_exists,path_valid,new_rel_dir = server_pxy.path_check(rel_dir)
        if path_valid and path_exists:
            return True, new_rel_dir
    print("Dir Path not valid or exists\n")
    return False,''
```

(5) 写文件

同样根据用户键入 write <file>调用 write_file 函数

```
elif command[0] == 'write' and len(command)==2:          # 写文件
    rel_path = str(Path(cd) / command[1].strip())
    if write_file(rel_path): print('Write file successfully.')
```

读写操作涉及到缓存、副本一致性和并行读写问题，先看 write_file 大致思路：

调用读文件 read_file 函数(后面介绍), 若文件存在返回 done=true 和读文件的实际路径：

调用 lock **排他锁**，根据实际路径打开文件 file 写入，同时复制到缓存目录中（**write through**），在原文件所在服务器中 update_file 更新原文件以及 update_backup 更新备份文件（**瞬时一致性**），更新 CACHES 表

```
def write_file(rel_path):
    done, path = read_file(rel_path,False)
    cache_path = cache_dir / str(Path(rel_path).name)
    if done:
        if not lock(rel_path,False): return False
        with open(path,'w') as file:
            print('Enter the content:')
            content = input()
            file.write(content)
        shutil.copy2(path, cache_path)
        address = proxy.get_file_servers(rel_path)[0]
        with ServerProxy(address, allow_none=True) as server_pxy:
            server_pxy.update_file(rel_path)
            server_pxy.update_backup(rel_path)
            cache_hash = server_pxy.hash_file(str(cache_path))
        proxy.delete_cache(args.user_id,rel_path)
        proxy.insert_cache(args.user_id,rel_path,cache_hash)
        unlock(rel_path,False)
        return True
```

若文件不存在则直接创建新文件：

创建新文件这里不需要加锁，直接使用一台服务器执行 write_new_file，最后同样执行 update_file、update_backup 以及缓存更新

```
    reply = input('No existing file. Create new one? Enter Y for yes: ')
    reply = reply.upper()
    if reply == 'Y': # 写新文件不需要锁
        print('Enter the content:')
        content = input()
        address = proxy.get_next_server()
        with ServerProxy(address, allow_none=True) as server_pxy:
            path = server_pxy.write_new_file(content,rel_path)
            server_pxy.update_file(rel_path)     # 数据库初始化文件
            server_pxy.update_backup(rel_path)  # 更新备份
            shutil.copy2(path, cache_path)       # 缓存
            cache_hash = server_pxy.hash_file(str(cache_path))
        proxy.insert_cache(args.user_id,rel_path,cache_hash)
        return True
    return False
```

以上是服务端 write_file 的思路，下面介绍其中涉及函数：lock、unlock，update_file、update_backup 函数

Lock、unlock：在 database.py 中通过 share_lock、exclusive_lock 表示共享锁和排他锁，分别用于读写操作，申请共享锁只需不存在申请 rel_path 不存在于排他锁，申请排他锁需要该 rel_path 不在共享锁和排他锁列表中，若不成立则返回冲突提醒

```python
def lock(rel_path,shared = True):
    if shared:
        if proxy.aquire_shared(rel_path):
            return True
    else:
        if proxy.aquire_exclusive(rel_path):
            return True
    print('Lock conflict: The file is currently locked by another user.')
    return False
def unlock(rel_path,shared = True):
    if shared:
        proxy.release_shared(rel_path)
    else:
        proxy.release_exclusive(rel_path)
    return True
```

database.py 中申请锁和释放锁流程如下：

```python
def aquire_shared(rel_path):
    global shared_lock,exclusive_lock
    if rel_path not in exclusive_lock:
        shared_lock.append(rel_path)
        return True
def release_shared(rel_path):
    global shared_lock
    shared_lock.remove(rel_path)
```

```python
def aquire_exclusive(rel_path):
    global shared_lock,exclusive_lock
    if rel_path not in shared_lock:
        if rel_path not in exclusive_lock:
            exclusive_lock.append(rel_path)
        return True
def release_exclusive(rel_path):
    global exclusive_lock
    exclusive_lock.remove(rel_path)
```

server 端的 **update_file**：更新数据库中原文件的记录，存在记录则将其删除，再根据实际路径 path 调用 initialize_file 得到文件信息统一插入数据库

```python
def update_file(rel_path):# 更新数据库原文件
    path = (server_dir / rel_path).resolve()
    with ServerProxy(data_url, allow_none=True) as data_pxy:
        address = data_pxy.get_file_servers(rel_path)
        if address:
            data_pxy.remove_one_file(rel_path)
        data_pxy.insert_files([initialize_file(args.server_id, str(path), str(path.name))])
    return True
```

Initialize_file 函数如下：返回该文件需要插入数据库的信息

```python
def initialize_file(server_id, path, file):
    last_modified = os.path.getmtime(path)
    hash = hash_file(path)
    rel_path = Path(path).relative_to(server_dir) # 相对路径
    is_backup = 0
    if '_backup' in file:
        name_no_backup = file.replace('_backup', '')
        is_backup = 1
        rel_path = rel_path.with_name(name_no_backup )
    return  server_id, str(rel_path), file, is_backup, hash, last_modified
```

上面涉及到 hash_file 函数：用于计算文件的哈希值

```python
def hash_file(file_path):
    hash_obj = hashlib.sha256()
    with open(file_path, 'rb') as rbFile:
        for block in iter(lambda: rbFile.read(4096), b""):
            hash_obj.update(block)
    return hash_obj.hexdigest()
```

与之相关的还有 hash_check 函数：比对实际路径哈希值

```python
def hash_check(rel_path,hash_check):
    path_exists,path_valid,new_rel_path =  path_check(rel_path)
    if not path_valid or not path_exists: return False
    path = (server_dir / new_rel_path).resolve()
    hash = hash_file(str(path))
    if hash == hash_check:
        return True
    else: return False
```

server 端的 **update_backup 函数**:

根据传入参数，backup=true 判断执行服务器持有文件为备份文件，将相对路径改为备份文件的相对路径执行 hash_check 函数比较目标 hash 值，若不相同则更新该备份文件：删除该文件返回备份文件的实际路径和所在服务器

```python
def update_backup(rel_path,backup =False):# 更新备份
    with ServerProxy(data_url, allow_none=True) as data_pxy:
        if backup:  # 作为备份
            hash = data_pxy.get_file_hash(rel_path)[0]
            file_backup = data_pxy.get_file_backup_name(rel_path)[0]
            rel_path_backup= str(Path(rel_path).with_name(file_backup))
            if not hash_check(rel_path_backup,hash):# 检查哈希值
                backup_path = server_dir / rel_path_backup
                os.remove(str(backup_path))
                data_pxy.delete_backup_file(args.server_id,rel_path)
                return True,str(backup_path),args.server_id
            return False,'',0
```

根据 backup=false 判断执行服务器持有原文件，判断是否存在备份，存在则遍历备份文件所在服务器，递归执行该函数，也就是到上图的流程，如果返回备份文件需要更新，则复制原文件给备份文件，更新数据库 FILES 表。如果不存在备份，则在另一台服务器新建备份，调用 creat_backup 新建备份文件

```python
        #作为原文件
        addresses = data_pxy.get_file_backup_servers(rel_path)
        if addresses:   # 存在备份
            for address in addresses:# 作为原文件
                with ServerProxy(address, allow_none=True) as server_pxy:
                    up, backup_path, backup_id = server_pxy.update_backup(rel_path,True)
                    if up:
                        path = (server_dir / rel_path).resolve()
                        shutil.copy2(path, Path(backup_path))
                        data_pxy.insert_files([server_pxy.initialize_file(backup_id, str(backup_path), str(Path(backup_path).name))])
        else:# 不存在备份则在下一个服务器新建一个备份
            address = data_pxy.get_next_server()
            global server_url
            if address==server_url:return True,'',0
            file_name = str(Path(rel_path).name).replace('.','_backup.')
            rel_path_backup = str(Path(rel_path).with_name(file_name))
            with ServerProxy(address, allow_none=True) as server_pxy:
                server_pxy.create_backup(str(server_dir / rel_path),rel_path_backup)
    return True,'',0
```

creat_backup 函数如下：根据原文件和备份文件路径进行文件复制，之后更新数据库 FILES 表即可。

```python
def create_backup(path,rel_path_backup):
    backup_path = server_dir / rel_path_backup
    parent_directory = Path(backup_path).parent
    parent_directory.mkdir(parents=True, exist_ok=True)
    shutil.copy2(path, backup_path)
    with ServerProxy(data_url, allow_none=True) as data_pxy:
        return data_pxy.insert_files([initialize_file(args.server_id, str(backup_path), str(backup_path.name))])
```

写操作涉及的函数介绍就这些，主要是锁和更新文件副本这类的操作

(6) 读文件

同样用户键入 read <file>即可通过 read_file 进行读操作

```python
elif command[0] == 'read' and len(command) ==2:
    rel_path = str(Path(cd) / command[1].strip())
    read_file(rel_path)
```

Read_file 函数：先从本地缓存目录中搜索是否存在该路径文件，有则搜出缓存 hash，同原文件的 hash 进行比较，若相同则直接读取缓存，且无需使用锁，不相同则删除 CACHES 表中该缓存记录

```python
file_hash = proxy.get_file_hash(rel_path)
if not file_hash:
    print('File not existing.')
    return False,''
if use_cache:
    cache_hash = proxy.get_cache_hash(args.user_id,rel_path)
    if cache_hash:
        if file_hash[0] == cache_hash[0]:
            path = cache_dir / str(Path(rel_path).name)
            with open(path,'r') as file:
                content = file.read()
                print('Read file in cache:')
                print(content)
            return True,''
        proxy.delete_cache(args.user_id,rel_path)
```

不存在缓存则找到原文件所在服务器，使用 lock 共享锁，之后执行 read_path 函数返回文件绝对路径，读取文件，更新缓存，使用 unlock 释放锁即可

```python
address = proxy.get_file_servers(rel_path)
if not lock(rel_path):return False, ''
with ServerProxy(address[0], allow_none=True) as server_pxy:
    path = server_pxy.read_path(rel_path,file_hash[0])
    if path == '':
        unlock(rel_path)
        return False,''
    with open(path,'r') as file:
        content = file.read()
        print('Read file:')
        print(content)
    cache_path = cache_dir / str(Path(path).name)
    shutil.copy2(path, cache_path)
    cache_hash = server_pxy.hash_file(str(cache_path))
proxy.insert_cache(args.user_id,rel_path,cache_hash)
unlock(rel_path)
return True,path
```

server 端 read_path 函数：与 update_backup 类似，使用递归调用，检查该路径与目标 hash 是否一致，一致则如下情况，backup=false 判定为原文件所在服务器操作，直接返回文件实际路径，如果 backup=true 则说明原文件有误，需要将该备份文件作为新的原文件，首先改名为原文件(去掉后缀:_backup)，删除原文件及其他备份文件记录，更新

FILES 表将其作为原文件，并执行 update_backup 新建备份文件

```
def read_path(rel_path,hash,backup=False):
    rig = hash_check(rel_path,hash)
    if rig:
        path = (server_dir / rel_path).resolve()
        if backup: #如果找到路径为备份文件路径，将其改名为原文件，
            back_path = path
            name_no_backup = (back_path.name).replace('_backup', '')
            path = back_path.with_name(name_no_backup )
            os.rename(back_path,path) #备份文件改名为原文件,改名后调用del_file不会被删
            new_rel_path = str(Path(rel_path).with_name(name_no_backup))#原文件相对路径
            del_file(new_rel_path) #删除原文件以及其他备份文件
            with ServerProxy(data_url, allow_none=True) as data_pxy:#删除数据库中该备份文件的记录，更新原文件记录
                files_ini = proxy.insert_files(initialize_file(args.server_id, str(path), str(path.name)))
                if files_ini:
                    update_backup(new_rel_path) #更新备份
        return str(path)
```

对于 hash 不匹配，如果是备份文件直接返回''，若为原文件则遍历备份文件，如果得到 hash 匹配的备份文件，则删除原文件自己，返回备份文件实际路径

```
if not backup:# 对于原文件损坏，继续查找备份文件
    path = server_dir / rel_path
    with ServerProxy(data_url, allow_none=True) as data_pxy:
        addresses = data_pxy.get_file_backup_servers(rel_path)
        file_backup = data_pxy.get_file_backup_name(rel_path)
        if addresses:
            rel_path_backup= Path(rel_path).with_name(file_backup[0]) #备份文件真实路径
            for address in addresses:
                with ServerProxy(address, allow_none=True) as server_pxy:
                    path_backup = server_pxy.read_path(str(rel_path_backup),hash,True)
                    if path_backup!='':
                        os.remove(str(path)) # 删除原文件
                        return path_backup
return ''
```

(7) 删除文件

同理键入 delete <file>调用 del_file 函数进行文件删除

```
elif command[0] == 'delete' and len(command) == 2:      # 删除文件
    rel_path = str(Path(cd) / command[1].strip())
    if del_file(rel_path): print('File deleted successfully.')
```

Del_file 函数：查看文件路径是否存在，存在则调用 server 端 del_file 函数，删除成功去除 CACHES 表相应缓存记录

```
def del_file(rel_path):
    address = proxy.get_file_servers(rel_path)
    if address:
        with ServerProxy(address[0], allow_none=True) as server_pxy:
            if not server_pxy.del_file(rel_path):return False
            proxy.delete_cache(args.user_id,rel_path)
    else:
        print('File not exists')
        return False
    return True
```

server 端 del_file 函数:

检查路径有效且存在，以及是否为文件，使用 os.remove 删除，对于原文件，还需要删除备份文件以及数据库 FILES 的相关记录

```python
def del_file(rel_path, backup=False):#删除文件和所有备份文件
    path_exists,path_valid,new_rel_path = path_check(rel_path)
    if not path_valid or not path_exists:return False
    path = server_dir / rel_path
    if path.is_file():os.remove(str(path))
    else: return False
    if not backup: # 对于原文件，还需清除备份文件以及数据库记录
        with ServerProxy(data_url, allow_none=True) as data_pxy:
            addresses = data_pxy.get_file_backup_servers(new_rel_path)
            file_backup = data_pxy.get_file_backup_name(new_rel_path)
        if addresses:
            rel_path_backup= Path(new_rel_path).with_name(file_backup[0])
            for address in addresses:
                with ServerProxy(address, allow_none=True) as server_pxy:
                    if not server_pxy.del_file(str(rel_path_backup), True):
                        return False
            with ServerProxy(data_url, allow_none=True) as data_pxy:
                if not data_pxy.delete_file(new_rel_path):
                    return False
    return True
```

至此，所有操作执行函数介绍完毕

四、 运行情况

首先执行代码:

database.py:

```
PS C:\Users\asus\Desktop\分布式系统\src> python database.py
127.0.0.1 - - [23/Dec/2023 21:38:24] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [23/Dec/2023 21:38:26] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [23/Dec/2023 21:38:33] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [23/Dec/2023 21:38:36] "POST /RPC2 HTTP/1.1" 200 -
```

server.py:打开两个服务器测试

```
PS C:\Users\asus\Desktop\分布式系统\src> python server.py 1 1000
Server initialize successfully
```

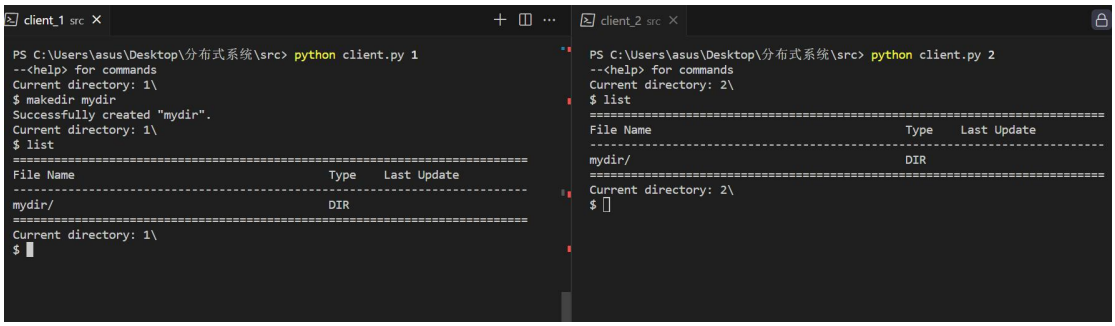
```
PS C:\Users\asus\Desktop\分布式系统\src> python server.py 2 2000
Server initialize successfully
```

client.py:打开两个用户测试

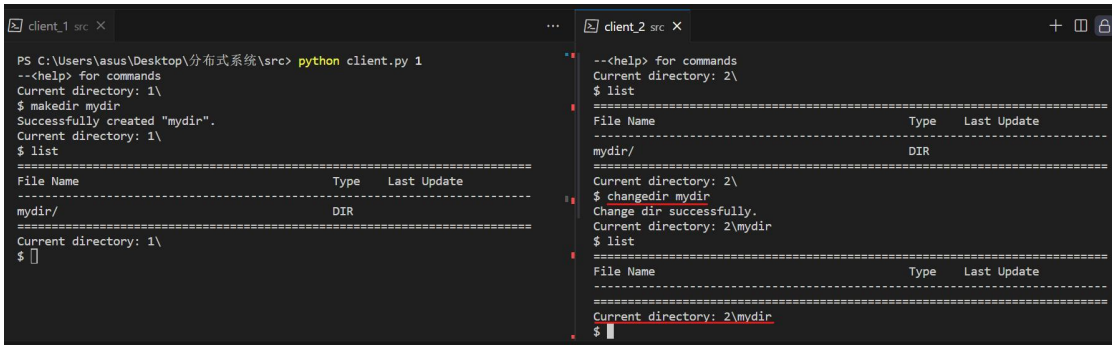
```
PS C:\Users\asus\Desktop\分布式系统\src> python client.py 1
--<help> for commands
Current directory: 1\
$
```

```
PS C:\Users\asus\Desktop\分布式系统\src> python client.py 2
--<help> for commands
Current directory: 2\
$
```

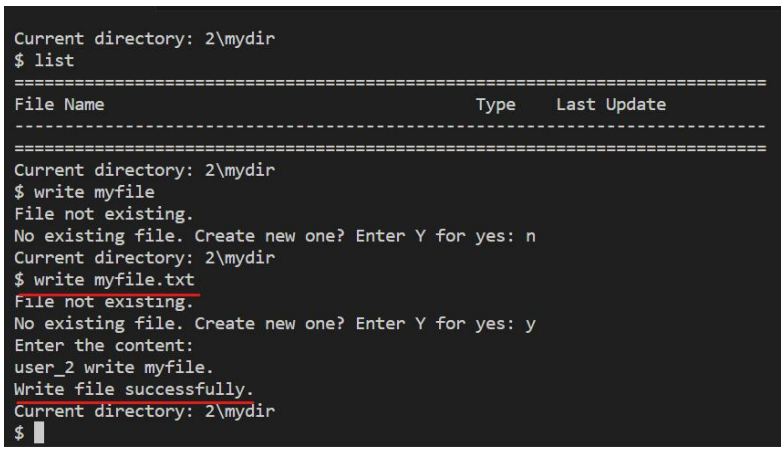
在用户 1 使用 makedir mydir 指令:



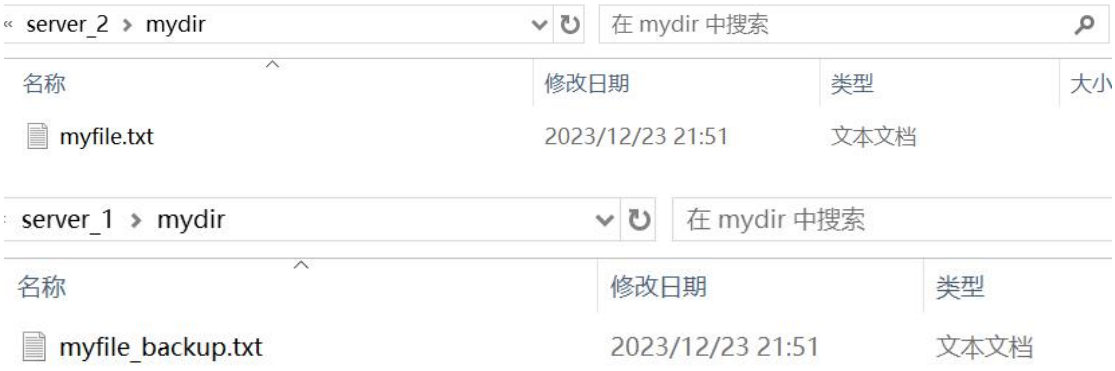
在用户 2 使用 changedir mydir 指令:



在用户 2 所在文件夹写文件



写入成功，查看文件夹：在 server_2 创建原文件，server_1 创建了副本



用户 1 查看文件:

```
$ changedir mydir
Change dir successfully.
Current directory: 1\mydir
$ list
================================================================================
File Name                                          Type    Last Update
--------------------------------------------------------------------------------
myfile.txt                                         TXT     2023-12-23 21:51:28.371975
================================================================================
Current directory: 1\mydir
$
```

用户 1 read 两次文件：可看到第二次 read 时使用的时 cache

```
$ read myfile.txt
Read file:
user_2 write myfile.
Current directory: 1\mydir
$ user_1 write myfile.
Invalid Command.
Current directory: 1\mydir
$ list
================================================================================
File Name                                          Type    Last Update
--------------------------------------------------------------------------------
myfile.txt                                         TXT     2023-12-23 21:51:28.37197
================================================================================
Current directory: 1\mydir
$ read myfile.txt
Read file in cache:
user_2 write myfile.
Current directory: 1\mydir
$
```

此时 CACHES 表：

user_id	path	cache_hash
2	mydir\myfile.txt	29392f705e9a2
1	mydir\myfile.txt	29392f705e9a2

FILES 表：

server_id	path	name	is_backup	file_hash	lastmodified
2	mydir\myfile.txt	myfile.txt	0	29392f705e9a2ed65893f3a0c276701a	03339488.37197
1	mydir\myfile.txt	myfile_backup.txt	1	29392f705e9a2ed65893f3a0c276701a	03339488.37197

用户 1 重新 write 该文件，用户 2 读该文件：

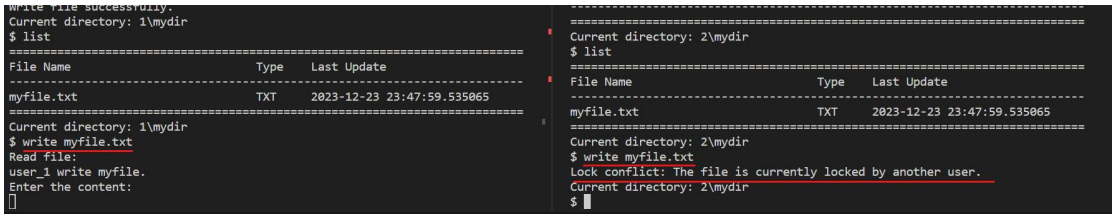
```
Current directory: 1\mydir
$ read myfile.txt
Read file in cache:
user_2 write myfile.
Current directory: 1\mydir
$ write myfile.txt
Read file:
user_2 write myfile.
Enter the content:
user_1 write myfile.
Write file successfully.
Current directory: 1\mydir
$
```

```
$ write myfile.txt
File not existing.
No existing file. Create new one? Enter Y for yes: y
Enter the content:
user_2 write myfile.
Write file successfully.
Current directory: 2\mydir
$ read myfile.txt
Read file:
user_1 write myfile.
Current directory: 2\mydir
$
```

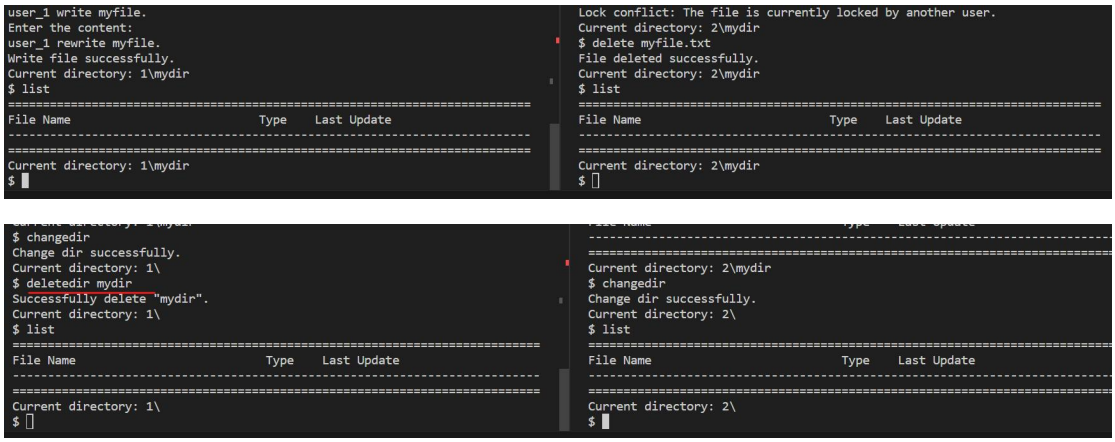
FILES 表：原文件和备份的 file_hash 都得到更新：

server_id	path	name	is_backup	file_hash	lastmodified
2	mydir\myfile.txt	myfile.txt	0	df754a3af26068550c84464eb4c734cbf	03340163.40919
1	mydir\myfile.txt	myfile_backup.txt	1	df754a3af26068550c84464eb4c734cbf	03340163.40919

验证并行写情况：可看到在用户 1 写入时用户 2 无法对相同文件进行写操作



最后删除文件并删除文件夹：



所有功能测试均正常。

五、 遇到的问题

主要在于学习使用 RPC 通信方式，以及利用 RPC 通信方式远程调用函数时，必须要有返回值。主要的实现困难在于考虑多台服务器之间的备份更新时，调用不同的 server.py 函数容易产生混淆和传递值的难题。以及实现文件锁时，起初考虑使用 Msvcrt 方法，但因为该方法在把文件上锁之后，客户自己也没法操作上锁的文件，这一点仍存在疑惑，后来通过自己维护共享锁和排他锁的方法解决问题。

六、 总结

本次作业完成了分布式文件系统，从参考到编写代码再到找 bug 的过程花了很长时间，最后只实现了基本的功能，且十分粗糙，在提升执行速度上还有很大提升空间。通过这次课程设计，也体会到分布式文件系统的实现需要考虑比想象中要多的细节，更加理解分布式系统之间分布、透明、同一性的特点，对这门课有了更深的感悟。