**分布式文件系统模拟实验报告**

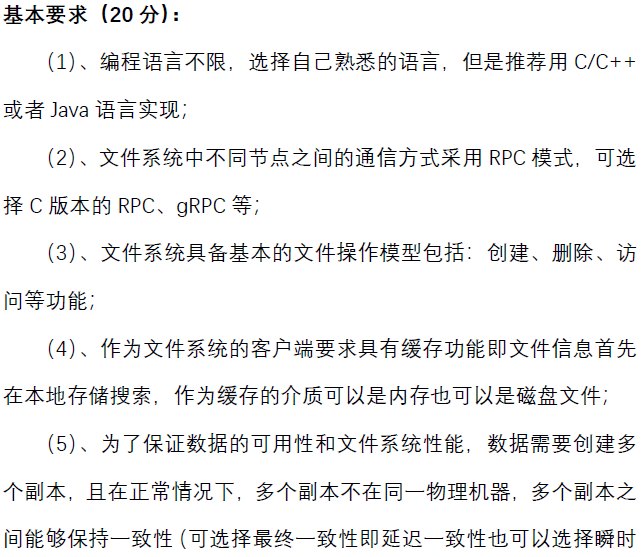
**姓名：欧阳润宇**

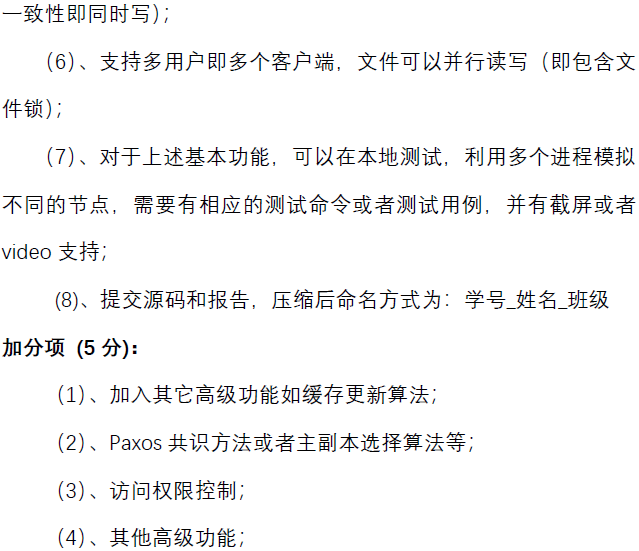
**班级：16计科五班**

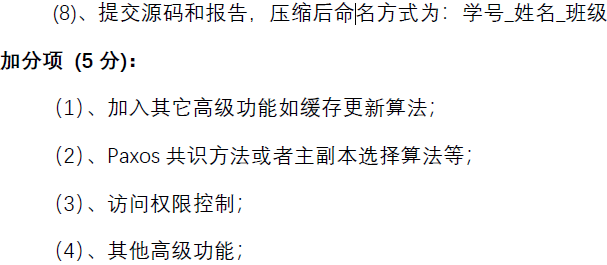
**学号：16337188**

**日期：2019年1月6日**

**题目要求：**







**系统功能简要说明：**

1. 在远程服务器创建文件、读写文件、删除文件、查找文件
2. 支持多用户登录，并发操作，支持简单的读写锁
3. 支持简单的用户缓存
4. 支持基本的权限管理
5. Master结点和Server端可保存配置文件
6. Master结点和用户端会保存Log文件

**文件说明：**

* **Client.py**: 客户端
* **ProxyServer.py**: 代理服务器（Master结点，默认**占用12340端口**）
* **Server.py**: 服务器端
* **ServerLaucher.py**: 服务器启动器（可以一次性启动多个服务器，最多10个，默认**占用12341~12341+n端口，n为服务器数**）
* **Clear.py:** 清空客户端和服务器所有配置以及文件
* **Client：**模拟客户端的本地存储（一开始为空）
* **ProxyServer：**模拟代理服务器的本地存储（一开始为空）
* **Server：**模拟服务器的本地存储（一开始为空）

**运行环境说明：**

* **运行环境：**Windows10 Python3.6
* **运行依赖包：**rpyc、shutil

**（具体操作方法和测试样例详见下面的内容）**

**操作方法及部分实现细节**

**启动Server端**



**（--n选项表示服务器个数，最多只能申请10个， 默认为5个，这里申请了7个）**

**启动Master**



**（Master中 –n选项的值必须与ServerLaucher的值一致）**

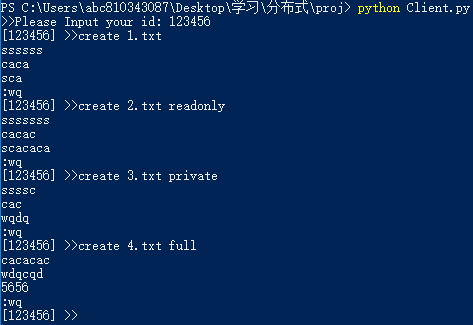
注册一个新用户123456，并在服务器创建4个文本文件

其中 1.txt **允许别的用户读和写**，但**不允许别的用户删除（writable）**。

2.txt **只允许别的用户读（readonly）**。

3.txt 是私有文件，**不允许别的用户访问（private）**。

4.txt 是完全公有的，**允许别的用户读、写且删除（full）**。



**注：**

**1. 输入的用户名只能有数字，且无需密码（没有注册，输入一个用户名即可使用系统）**

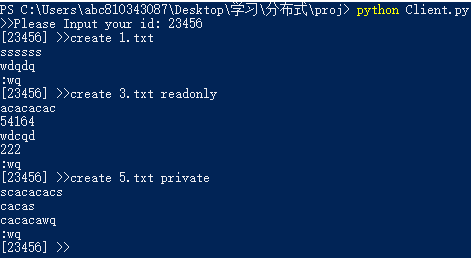
**2. 创建文件的命令格式为create [文件名] [权限]**

**（文件权限默认为writable）**

**3. 创建文件后，需手动输入文件内容，以:wq结尾**

开启新终端，注册一个新用户23456，并在服务器创建三个文件。

1.txt [**writable]** 5.txt **[private**] 3.txt[**readonly**]

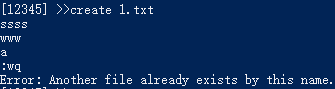


开启新终端，注册一个用户12345，并在服务器创建一个文件。

1.txt [**writable**]



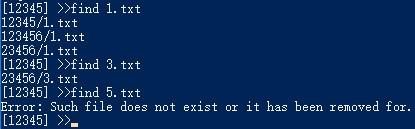
继续创建文件：



**注：**

**4.** create命令创建的文件会**存放在本地的缓存一段时间**，同时**推送到文件系统的用户个人文件夹**，**不允许同名文件**。

用户12345执行**find**命令，**寻找一个文件**。

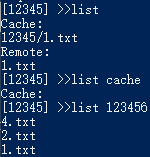


**注：**

1. **寻找文件的命令格式为find [文件名]，会得到文件的路径并显示在终端**
2. **别的用户的private文件会对当前用户隐藏起来。**

（由前面可知，用户23456创建了一个private的5.txt，但用户12345找不到该文件；同理，用户12345也找不到用户123456创建的3.txt）

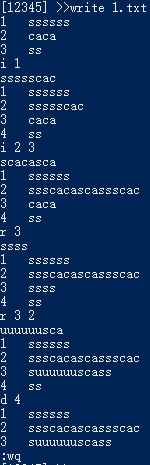
用户12345执行**list**命令，**列出一个路径下的所有文件**。



**注：**

1. **列出所有文件的命令格式为 list [路径名]，会将路径下的文件名显示在终端**
2. 如果不加路径名选项，默认会**列出自己本地的cache和文件系统的个人文件夹中的所有文件名。**
3. **list cache用于列出自己cache文件夹中的所有文件名（**cache中的文件有一定几率被抛弃**）**
4. **对于别的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（**如前面截图所示，用户123456的3.txt没有显示在终端**）**

用户12345执行**write**命令，写一个文件。



**注：**

1. **写文件的命令为write [文件名]，若文件名没有指定用户（**指定用户，例:如: 123456/1.txt; 不指定用户，例如：1.txt**），默认写自己的文件。**
2. 输入命令后，之后会进入**写文件模式**，写文件有如下几条命令**：**

**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**

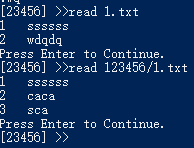
用户12345继续尝试写文件：



**注：**

1. **如果尝试写一个readonly的文件，会提示权限不够；如果尝试写一个private的文件，会提示不存在该文件。**
2. **用户会将修改后的文件先放在缓存一段时间。**

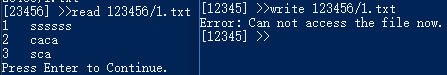
用户23456执行**read**命令，读一个文件。



**注：**

1. **读文件的命令为read [文件名]，若文件名没有指定用户（**指定用户，例:如: 123456/1.txt; 不指定用户，例如：1.txt**），默认读自己的文件。**
2. 输入命令后，之后会进入**读文件模式**，读文件模式输入Enter后退出。
3. **用户会将读过的文件先放在缓存一段时间。**

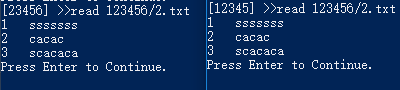
用户23456读文件时，用户12345写同一个文件。



**注：**

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

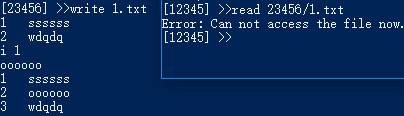
用户23456读文件时，用户12345读同一个文件。



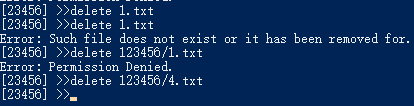
**注：**

**读操作可以同时执行**

用户23456写文件时，用户12345读同一个文件



用户23456执行**delete**命令**,** 删除一个文件。

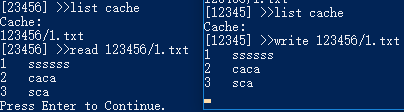


**注：**

1. **删除文件的命令为delete [文件名]，若文件名没有指定用户（**指定用户，例:如: 123456/1.txt; 不指定用户，例如：1.txt**），默认删除自己的文件。**
2. **如果删除一个不存在的文件，会报错**
3. **如果删除别的用户的权限级别不是full的文件**（如上图中的123456/1.txt）**，会提示权限不够**

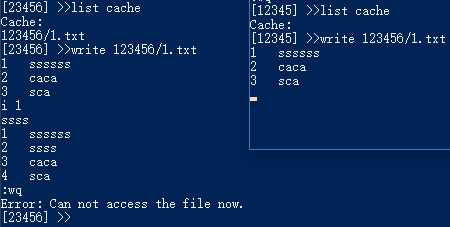
**以下是几种特殊情况及应对方法：**

1. 若一个文件a在用户A本地的缓存中，用户B正在远程修改文件系统中的文件a，**用户A读文件a的操作可以与B写文件a的操作同时进行**（因为此时用户B读的是自己缓存中的文件）



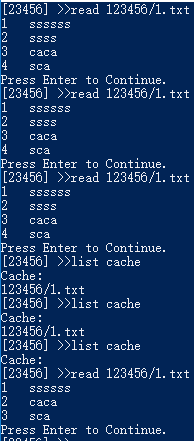
（上图表示在用户12345正在写文件123456/1.txt时，用户23456读本地缓存中的123456/1.txt文件）

1. 若一个文件a在用户A本地的缓存中，用户B正在修改远程文件系统中的文件a，**用户A可以修改自己本地缓存中的文件a，但修改后的文件a不会更新到远程文件系统中**。

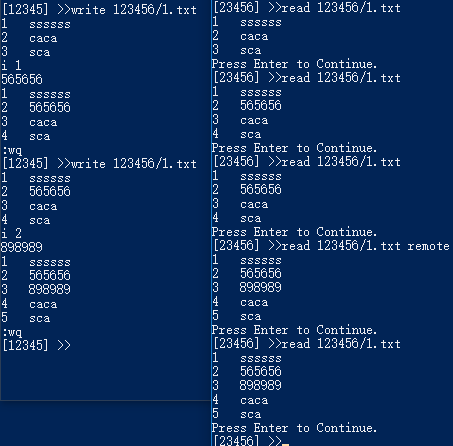


（上图表示在用户12345正在写文件123456/1.txt时，用户23456修改本地缓存中的123456/1.txt文件，修改完后提交到文件系统时，由于写锁的存在，无法更新）

1. 如下图（下图的状态是上面的操作完成之后），在之后的一段时间内，用户23456本地缓存中的123456/1.txt与远程文件系统中的123456/1.txt会出现不一致的情况，**直到缓存中的123456/1.txt失效，read才能读到远程文件系统中真实的123456/1.txt**。

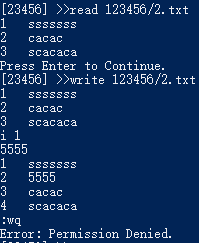


为解决上述情况，可**在read命令后加上remote选项，强制读取远程文件系统中的文件，并更新到缓存中**。



（如上图，用户12345对123456/1.txt的某次修改没有立刻更新到用户23456的本地缓存中，用户23456在read后面加上remote，强制读取远程文件系统的123456/1.txt，并更新到缓存中）

1. 若一个只读文件a在用户A的本地缓存中，**用户A可对本地缓存中的文件a进行修改，但修改后的文件a不能更新到文件系统中**（因为是只读文件）



（如上图，用户23456读取只读文件123456/2.txt并存在缓存中，之后用户23456修改缓存中的123456/2.txt，修改完成后由于权限不够无法更新）

之后退出整个系统（**客户端、 Master端和Server端均需在终端输入ctrl+c退出，客户端和Server端退出后可能会报错，但并无影响**）。

退出系统后，查看模拟的客户本地缓存，以客户12345为例：



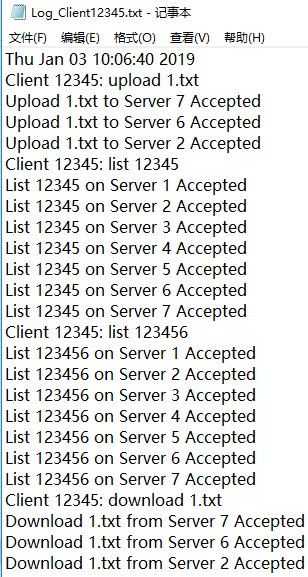
路径是 **[主目录]/Client/用户名/Cache**

可以看到缓存中存放了123456/1.txt

同样可以看到客户端的日志文件，路径是**[主目录]/Client/[用户名]/Log.txt**

查看模拟的Master端日志文件，路径是**[主目录]/ProxyServer/Log/**

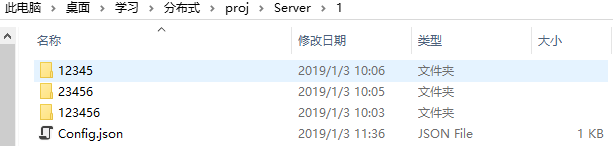




查看模拟的Master端的配置文件，路径是**[主目录]/ProxyServer/Config.json**

配置文件主要**记录了不同的文件存放在哪个远程文件服务器上**

查看Server端文件夹，以id为1的Server为例



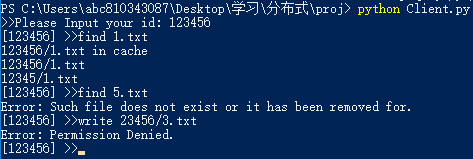
路径是**[主目录]/Server/[id]**



Server端的配置文件[Config.json]主要是**记录不同文件的访问权限**。



重开整个系统，可以加载之前的配置。



（**可见权限配置和文件配置均已加载成功**）

**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，通常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. 用户的登录和退出登录的实现方法过于简单

**总结：**

这次proj我尝试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，且在实现过程中体会到了分布式系统的精妙，之后的学习过程中我会记住分布式系统的一系列知识并加以运用。