实验十设计文档

1. 实验任务

服务器采用多线程的方式，用 thread 创建任务，能够实现多客户端的连接服务。客户端连接成功，接收客户端的请求信息。

- 客户端能够从服务器下载指定文件
- 客户端也能上传本地文件到服务器

2. 实验过程

2. 1 客户端程序

main.py

在主程序中，根据用户输入的功能选择以及文件名来调用对应的函数。

```
if __name__ == '__main__':
    print('We provide the following two functions: ')
    print('1. Upload file ')
    print('2. Download file \n')
    print('Please input the function number you want: (eg. 1) ')
    option = input()
    if option == '1':
        print('Please input the file name you want to upload: (eg. Goodbye.txt)
')
    else:
        print('Please input the file name you want to download: (eg.
Goodbye.txt) ')
    filename = input()
    if option == '1':
        print('Upload... ')
        client.upload_file(filename)
    else:
        print('Download... ')
        client.download_file(filename)
```

client.py

- 下载文件
    1. 新建socket，然后调用connect()和服务器建立TCP连接（ip为127.0.0.1，端口为5000）
    2. 发送的数据中，第一个字符串为download表示这是下载功能，第二个字符串为文件名

3. 调用recv()接收服务器传来的数据，对其用空格切割。若第一个字符为1，表示存在该文件，那么循环读入数据并创建文件，写入内容；若第一个字符为0，表示不存在该文件，输出找不到该文件。
4. 调用close()关闭socket

```
def download_file(filename):
    clientSocket = socket(AF_INET, SOCK_STREAM)
    clientSocket.connect((serverHost, serverPort))
    # 发送要下载文件的命令以及文件名给服务器
    outputdata = 'download ' + filename
    clientSocket.send(outputdata.encode())
    data = 1
    while data:
        data = clientSocket.recv(1024)
        list = data.decode().split()
        if len(list):
            flag = list[0]
            if flag == '1':
                del list[0]
                str = ' '
                data = str.join(list)
                # 创建文件，写入内容
                f = open(filename, 'a+')
                f.write(data)
                f.close()
            elif flag == '0':
                print('Not found this file!')

    clientSocket.close()
    print('Download finished!\n')
```

- 上传文件

1. 新建socket，然后调用connect()和服务器建立TCP连接（ip为127.0.0.1，端口为5000）
2. 发送的数据中，第一个字符串为upload表示这是上传功能，第二个字符串为文件名
3. 打开客户端上对应的文件，接到前两个参数之后，发送给服务器。
4. 调用close()关闭socket

```
def upload_file(filename):
    clientSocket = socket(AF_INET, SOCK_STREAM)
    clientSocket.connect((serverHost, serverPort))
    outputdata = 'upload ' + filename + ' '

    f = open(filename, "rb")
    outputdata += f.read().decode()
    f.close()
    clientSocket.send(outputdata.encode())
    clientSocket.close()
    print('Upload finished!\n')
```

2. 2 服务器程序

main函数

1. 新建socket
2. 调用bind绑定指定的ip和端口号
3. 调用listen函数保证挂起的连接池中至少要有10个连接
4. 调用accept()函数，用于从已完成连接队列队头返回下一个已完成连接。
5. 在python里面，线程的创建有两种方式，在本次实验中我们采用Thread类创建。创建该类的对象时需要两个参数，target和args：target为需要线程去执行的方法名；args为线程执行方法接收的参数，该属性是一个元组，如果只有一个参数也需要在末尾加逗号。然后对象调用start()方法就可以运行线程。

   使用线程，首先创建一个主线程，在固定端口监听客户端请求。当从客户端收到TCP连接请求时，它将通过另一个端口建立TCP连接，并在另外的单独线程中为客户端请求提供服务。这样在每个请求/响应对的独立线程中将有一个独立的TCP连接。

```
if __name__ == '__main__':
    serverSocket = socket(AF_INET, SOCK_STREAM)
    # Prepare a sever socket
    serverHost = '127.0.0.1'
    serverPort = 5000
    serverSocket.bind((serverHost, serverPort))  # 指定ip和端口号
    serverSocket.listen(10)  # 保证挂起的连接池中至少要有10个连接

    while True:
        # Establish the connection
        print('Ready to serve...')
        connectionSocket, addr = serverSocket.accept()
        print(connectionSocket)
        thread = threading.Thread(target=server_process, args=
(connectionSocket,))
        thread.start()

    serverSocket.close()
```

server_process函数

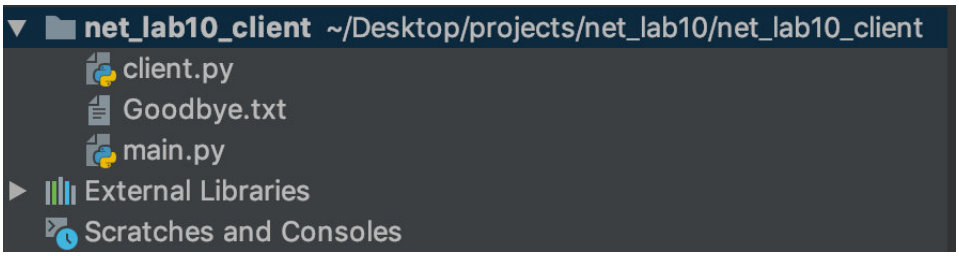
1. 调用recv()接收客户端传来的数据，对其用空格切割。若第一个参数为download，说明是下载文件功能，转2；若第一个参数为upload，说明是上传文件功能，转3
2. 若能打开对应的文件，则发送的数据第一个字符串为1，后面跟着文件中的数据，发送完关闭socket；若不能，发送的数据第一个字符串为0，且后面跟着Not found this file!
3. 删掉接收到的数据中的前两个元素option和filename，剩下的即为文件内容，创建文件并且写入内容。

```
def server_process(connectionSocket):
    try:
        message = connectionSocket.recv(1024)
        message = message.decode()
        list = message.split()  # 将数据用空格分开
        option = list[0]
        filename = list[1]
        if option == 'download':
            outputdata = '1 '  # 表示有该文件
            f = open(filename, "rb")
            outputdata += f.read().decode()
            f.close()
            connectionSocket.send(outputdata.encode())
            connectionSocket.close()
            print('Download finished!')
        elif option == 'upload':
            data = 1
            while data:
                f = open(filename, 'a+')
                del list[0:2]  # 删掉前两个元素option和filename
                str = ' '
                data = str.join(list)
                print(data)
                f.write(data)
                f.close()
            connectionSocket.close()
            print('Upload finished!')
    except IOError:
        # Send response message for file not found
        outputdata = '0 Not found this file!\r\n\r\n'
        # Close client socket
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i].encode())
        connectionSocket.close()
```

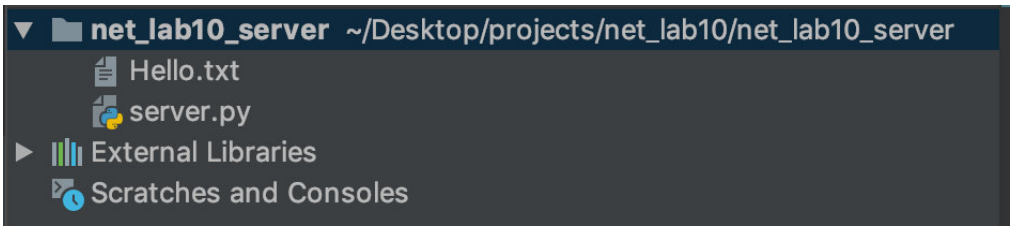
3. 实验结果

3. 1 下载文件

1. 一开始，客户端只有Goodbye.txt一个文件

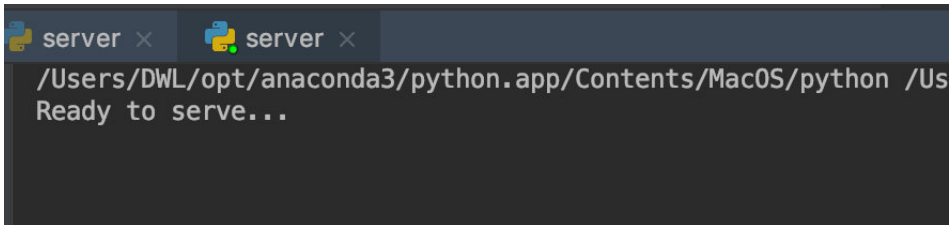


2. 服务器只有Hello.txt一个文件

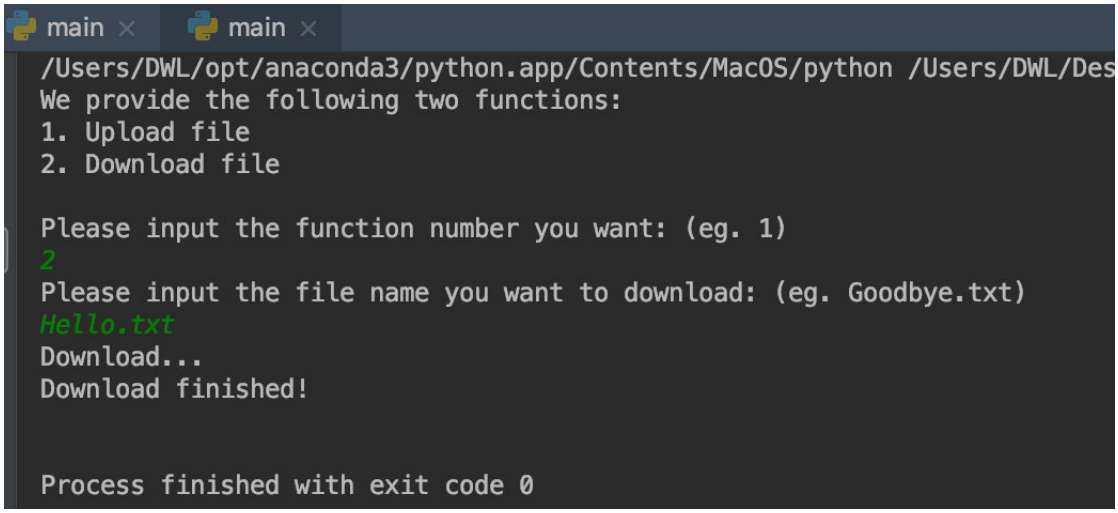


文件中的内容是：Hello, I am server.

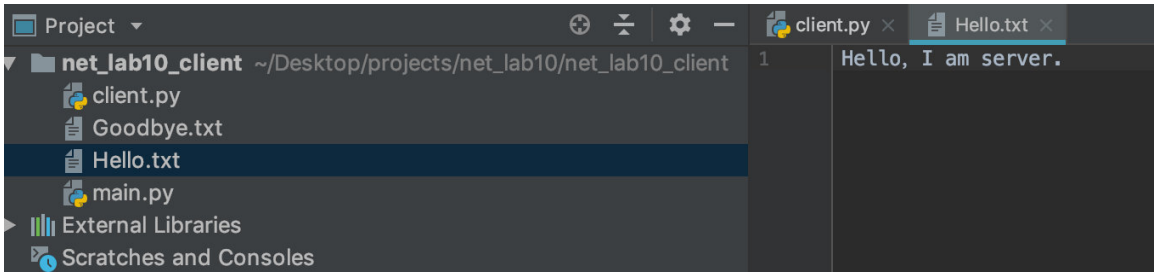
3. 运行服务器程序，保证服务器启动。



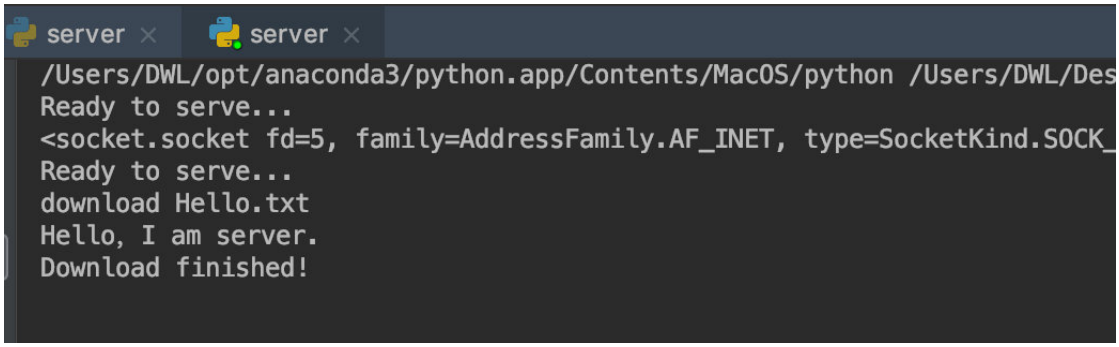
4. 开始运行客户端程序，根据程序提示，选择功能2：下载文件。并且输入要下载的文件名：Hello.txt。



5. 下载完成，可以看到客户端程序显示下载完成，并且多了Hello.txt文件，且文件内容正确。



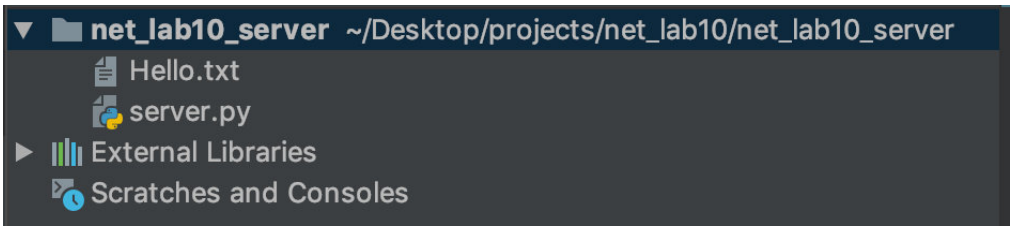
6. 可以看到服务器程序显示下载完成。



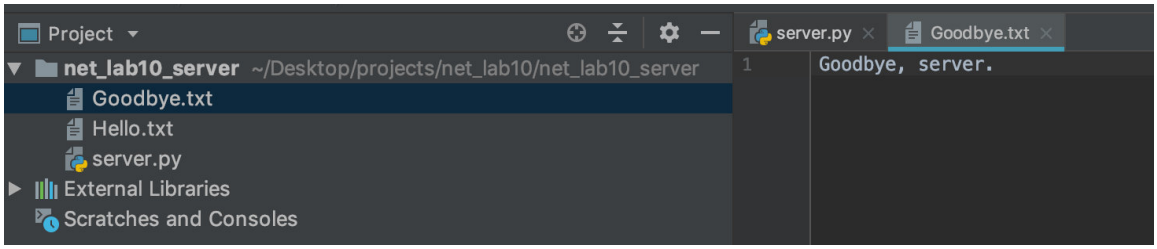
7. 若输入不存在的文件名，会显示Not found this file!

3. 2 上传文件

1. 一开始，服务器只有Hello.txt一个文件



2. 客户端有Goodbye.txt和Hello.txt两个文件，且Goodbye.txt的文件内容为Goodbye, server.



3. 运行服务器程序，保证服务器启动。

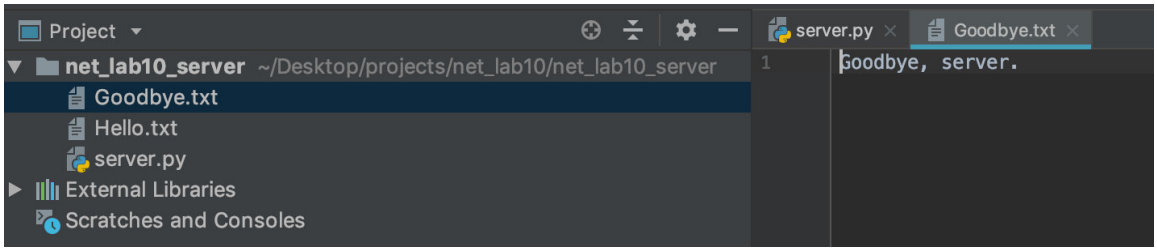
4. 开始运行客户端程序，根据程序提示，选择功能1：上传文件。并且输入要上传的文件名：Goodbye.txt。

```
/Users/DWL/opt/anaconda3/python.app/Contents/MacOS/python /Users/DWL/Desktop/
We provide the following two functions:
1. Upload file
2. Download file

Please input the function number you want: (eg. 1)
1
Please input the file name you want to upload: (eg. Goodbye.txt)
Goodbye.txt
Upload...
Upload finished!


Process finished with exit code 0
```

5. 上传完成，可以看到客户端程序显示上传完成。
6. 可以看到服务器多了Goodbye.txt文件，且内容正确。




```
server
/Users/DWL/opt/anaconda3/python.app/Contents/MacOS/python /Users/DWL/Desktop/projects/net_lab
Ready to serve...
<socket.socket fd=5, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, ladd
Ready to serve...
Goodbye, server.

Upload finished!
```

3. 3 多线程

1. 为了演示多线程的功能，我打开两个相同的客户端程序，并且注释掉下载文件完成时关闭socket的代码，并且在服务端代码中加入线程id的打印。

```
    # clientSocket.close()
    print('Download finished!\n')
```

```
# connectionSocket.close()
print('thread id: ', threading.currentThread())
print('Download finished!')
```

注意：起初我采用threading.currentThread().ident打印线程号，但打印出来的是一样的。查找解答发现：

我怀疑这是预期的行为；以下是来自threading文档的信息：

> The 'thread identifier' of this thread or None if the thread has not been started. This is a nonzero integer. See the thread.get_ident() function. **Thread identifiers may be recycled when a thread exits and another thread is created.**

此外,我在REPL中修改了thid,如下所示：

```
>>> def thid():
...     print threading.currentThread().ident
...     sleep(10)
```

当我在调用t1.start()的10秒内调用t2.start()时,它们具有不同的ID,但是如果我等待10秒钟以上,则它们具有相同的ID.

如果要区分不同的线程,只需调用id(threading.currentThread()). Python ID始终是不同的.

2. 然后两个客户端使用下载文件功能

```
/Users/DWL/opt/anaconda3/python.app/Contents/MacOS/python /Users/DWL/De
We provide the following two functions:
1. Upload file
2. Download file

Please input the function number you want: (eg. 1)
2
Please input the file name you want to download: (eg. Goodbye.txt)
Hello.txt
Download...
```

3. 可以看到服务器答应出的Thread号是不一样的

```
/Users/DWL/opt/anaconda3/python.app/Contents/MacOS/python /Users/D
Ready to serve...
Ready to serve...
thread id:  <Thread(Thread-1, started 123145542496256)>
Download finished!
Ready to serve...
thread id:  <Thread(Thread-2, started 123145542496256)>
Download finished!
```