CSC 401: Data and Computer Communications Networks
2016 Fall Programming Assignment

VM DOES NOT HAVE JAVA INSTALLED

1 Overview

In this assignment you will develop a client program written in Python (or JAVA) to interact with a "**CSC 401 Server**" running on a remote machine. The 401 server, is running on a VCL Virtual Machine with IP address **152.1.13.219** and waits for TCP connections on ports **20001**, **20002**, and **20003**. You can contact the server at any one of these ports.

This project has two parts. For the first part, you will create a client to interact with server. For the second part, you will modify your client and write additional code to extend the interaction between the server and the client.The primary objectives of this assignment are:

- to illustrate the client-server paradigm
- to show you how programs actually access network services
- to introduce you to the sockets programming interface

2 Systems Used

To complete this assignment, you **HAVE** to use the VCL image called **CSC401 RHEL 6 Base (64 bit VM)** image.

Go to **vcl.ncsu.edu** and make a reservation. You need to store your code on your AFS drive rather than on the hard drive of the VM that you are working on. If you don't do that, when your VCL reservation for that VM expires then all your files will be gone.

Your code will NOT work if you try to execute it on other machines. This is because the correct ports are opened on that VCL image and thus the client and server can talk.

3 Four-Way Handshake - Exercise 0

In this exercise you will write a client program and use it to interact with the CS 401 Server according to a four-way handshake protocol:

Step 1: The **client** begins by sending a request
Step 2: the **server** sends back a confirmation
Step 3: then the **client** sends an **ack**
Step 4: and the **server** replies with an **ack**.

The protocol is specified next.

3.1 The Protocol

The client and server communicate by exchanging lines of ASCII characters via the reliable byte-stream service provided by TCP. (In Python, the socket type is SOCK_STREAM, and the address family is AF_INET.) The interaction between the client and server for this exercise proceeds as follows:

1. The server runs on IP address **152.1.13.219**. It "listens" for connections on ports 20001, 20002, and 20003. Your can contact the server at any one of these three ports.
2. The client opens a connection to the server's socket.
3. The server accepts the connection and waits to receive a request string from the client.
4. Once the client is connected to the server, it immediately sends a request string. The format of the request string is:

<request type> <WS> <connection specifier> <WS> <usernum> <WS> <username> <newline>

where:

- The <request type> is a string of the form "exi", where i is the exercise number (i.e. for this exercise the string is **ex0**). Use lower case in this string for the letters.
- <WS> is "whitespace", one or more blank characters.
- The <connection specifier> is of the form

  <server endpoint specifier> <WS> <client endpoint specifier>

  i.e., two endpoint specifiers separated by white space, where each endpoint specifier is of the form

  <dotted quad>-<port number>

  The first of these specifiers refers to the server's socket, and the second to the client's socket.
- The <usernum> is any integer (randomly selected by you) from 20100 to 60000. Don't hardcode this number, pick a random number each time the code is executed.
- The <username> is the student's (i.e. your) name, in the form of first initial, middle initial, last name, all one word with no whitespace (for example, "J.S.Doe").
- <newline> is the end-of-line marker, represented by the single character '\n'.

Thus, an example of a client request is:

```
ex0 152.1.13.219-20001 152.46.18.227-36088 31437  I.M.Student\n
```

5. Upon receiving and parsing the request string, the server sends back a confirmation in the form of one or more lines terminated by '\n'. The first line always contains an identification ("CSC 401 Server") and the date and time. If the request was properly formatted, and the connection specifier does in fact refer to the current connection, the second line will contain the string "OK", followed by whitespace, followed by identification information from the client request (in the form of **usernum+1** and **username**), followed by a random integer (to be called

servernum):

```
CSC 401 Server 2015-02-13 05:16:50\n
OK 31438 I.M.Student 57261764\n
```

If the request is NOT properly formed, or does not refer to the present connection, the second line will contain an indication that an error occurred, and no random integer.

6. If the server confirmation is OK, the client then sends an ack string as follows:

   `<request type> <WS> <usernum+1> <WS> <servernum+1> <newline>`

   Thus an example of a client ack is:

   ```
   ex0 31438 57261765\n
   ```

7. Upon receiving the client ack, the server sends back an ack string terminated by '\n', containing its identification ("CSC 401 Server") and the date and time. If the client ack was properly formatted, the server ack will contain the string "OK", followed by whitespace, followed by **servernum+1**, such as:

   ```
   CSC 401 Server 2015-02-13 05:16:50\n
   OK 57261765\n
   ```

   If the client ack was not properly formed, the server ack will contain an indication that an error occurred.

8. After sending the ack, the server waits for the client to close the connection. Upon receiving the '\n' character of the server ack, the client closes the connection.

3.2 Client Operation

To implement the above protocol, the client does the following. The functions given are for Python.

1. Create a socket (of type SOCK_STREAM and family AF_INET) using socket().
2. Call connect() with the server's IP address and port number to initiate a connection to the server. If unsuccessful, print out the reason for the error and exit.
3. Generate an integer from 20100 to 60000 randomly (**usernum**). Construct a request string using the server and client address information. To determine the client address information, the client can use the getsockname() function in Python.

   When the client socket is first created, it is not bound to any address. At connect time, however, the system implicitly assigns an address to the socket. The IP address will be the host's IP address, while the port number is chosen by the OS. In Python, the desired information can be obtained from getsockname() after the socket is connected to the server.

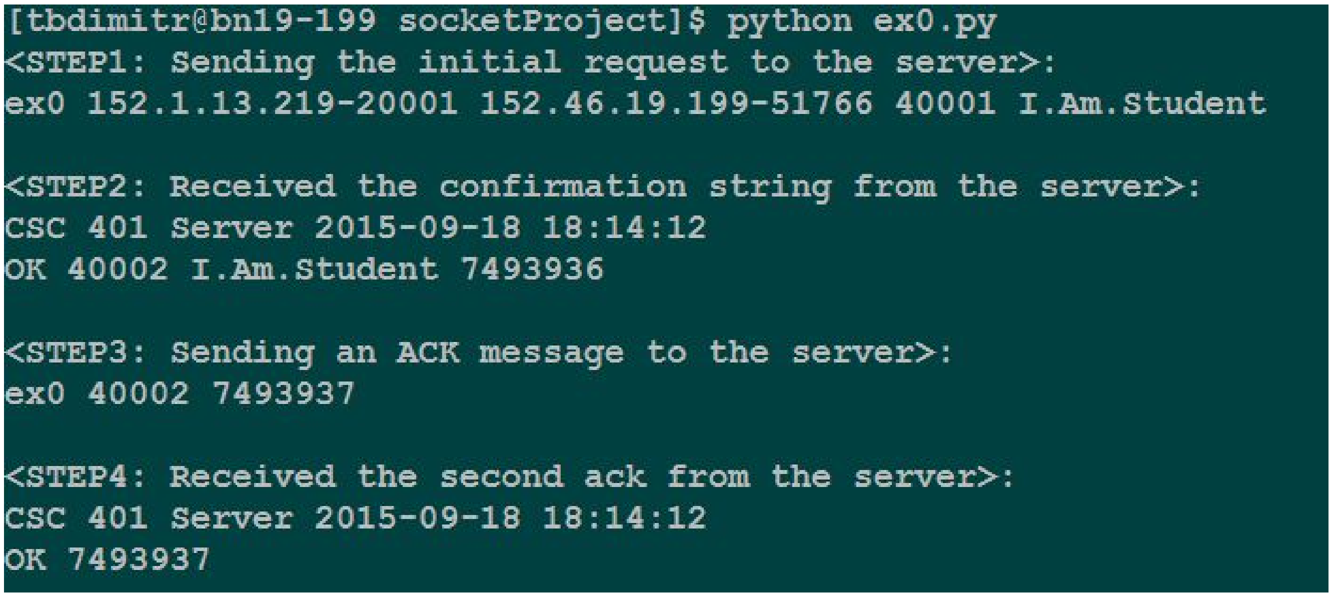
4. Write the client request string to the socket (using send()).
5. Read data from the socket (using recv()) until the second newline character is encountered. Verify that the first word on the second line is "OK", the value of **usernum+1**, and output the

received random number (**servernum**). If the first word is not “OK”, print an error indication and the received string.

6. Construct an ack string and write it to the socket (using send()).
7. Read data from the socket (using recv()) until the newline character is encountered. Verify that the string “OK” is received and output the received value of **servernum+1**. If not verified, print an error indication and the received string.
8. Close the connection (using close()).

3.3 Example Output.

Here is example output of running the client. It shows the four steps of the four-way handshake protocol. **Follow this format of output for your code**.



```
[tbdimitr@bn19-199 socketProject]$ python ex0.py
<STEP1: Sending the initial request to the server>:
ex0 152.1.13.219-20001 152.46.19.199-51766 40001 I.Am.Student

<STEP2: Received the confirmation string from the server>:
CSC 401 Server 2015-09-18 18:14:12
OK 40002 I.Am.Student 7493936

<STEP3: Sending an ACK message to the server>:
ex0 40002 7493937

<STEP4: Received the second ack from the server>:
CSC 401 Server 2015-09-18 18:14:12
OK 7493937
```

4 Listening for a new connection - Exercise 1

In this exercise, the client sends a request with the same format as above but with a different request type (“ex1”). For this type of request, the server interprets the `usernum` as the target port to which it (the server) should initiate a second connection to the client.

The server first sends the confirmation (including the server’s random number) on the original connection, and then tries to open a new connection back to the client on the indicated port (i.e. `usernum`). Thus, the roles of the client and server are *swapped* during the latter phase of this exercise, with the client passively waiting for a new connection initiated by the server.

4.1 The Protocol

The first five steps of the protocol are identical to Exercise 0 with two exceptions. The request format is identical to that of Exercise 0, except that the request type is **ex1** instead of **ex0**. Also, in the request string, the `usernum` specifies the port where the client is waiting the server to contact it back via a new connection. The client will have *two* sockets for this exercise.

The protocol diverges from the Exercise 0 protocol at Step 6:

6. After sending the confirmation and random number, the server immediately initiates a connection back to the client on the port specified by the client as `usernum`. Thus, the client must be "listening" for that new connection.
7. After accepting the second connection, the client waits for the server to send on the second connection a new random integer (**newservernum**) as follows:

   `CSC 401 Server calling <WS> <newservernum><newline>`

   `(Subsequently, the CSC 401 server stops sending.)`
8. After receiving the string, the client prints CSC 401 server sent <**newservernum**> and then sends back on the second socket connection the following string

   `<servernum+1> <WS> <newservernum+1> <newline>`

   where <**servernum**> is the random number previously received from the server in the initial confirmation string. The client then closes the second connection.
9. Upon receiving this string, the server closes its end of the second connection.
10. After verifying that the string received on the second connection contains the two random numbers it sent, the server sends a second confirmation string on the original connection, and then waits for the client to close that connection. (Note: the second confirmation string does not include the string with the date, only "OK" followed by a new random number, followed by newline.) If the server has encountered some problem, it instead sends an error indication on the original connection.
11. When the client has received the second confirmation string, it closes the original connection to the server.

4.2 "Client" Operation

The steps followed by the client for the protocol of Exercise 1 are. The specific functions mentioned are for Python.

1. Create a socket (call it **`psock`**) of type SOCK_STREAM and family AF_INET using **`socket()`**.
2. Bind the socket **`psock`** to a random port number in the range from 20100 to 60000, using **`bind()`**.
3. Find out what port **`psock`** was bound to using the **`getsockname()`** function. To determine the IP address of the socket, you can use the **`gethostname()`** and **`getaddrinfo()`** functions. The former returns the name of the host, while the latter returns the list of IP addresses associated with a name of this type. Print out the address and port, so the user can see what's going on.
4. Call **`listen()`** on **`psock`** so it will accept new connections.
5. Construct the request string to be sent to the server. The process is similar to Exercise 0, but the client endpoint specifier has a different meaning.
6. Open the first connection to the server and send the request.

7. Receive the confirmation string from the server on the first connection opened in step 6; if the first word of the status line is "OK", save the random number for constructing the string that will be sent on the second connection. Otherwise close the connection, print an error message, and exit.
8. Call **`accept()`** on **`psock`**; if successful, this will return another socket (call it **`newsock`**) for the second connection, which has been initiated by the server.
9. Call **`recv()`** on **`newsock`** to get the new random number from server or until it returns None (indicating that the other end of the second connection has stopped sending). If the random number is received, print the line

    `CSC 401 server sent <WS> <newservernum>`

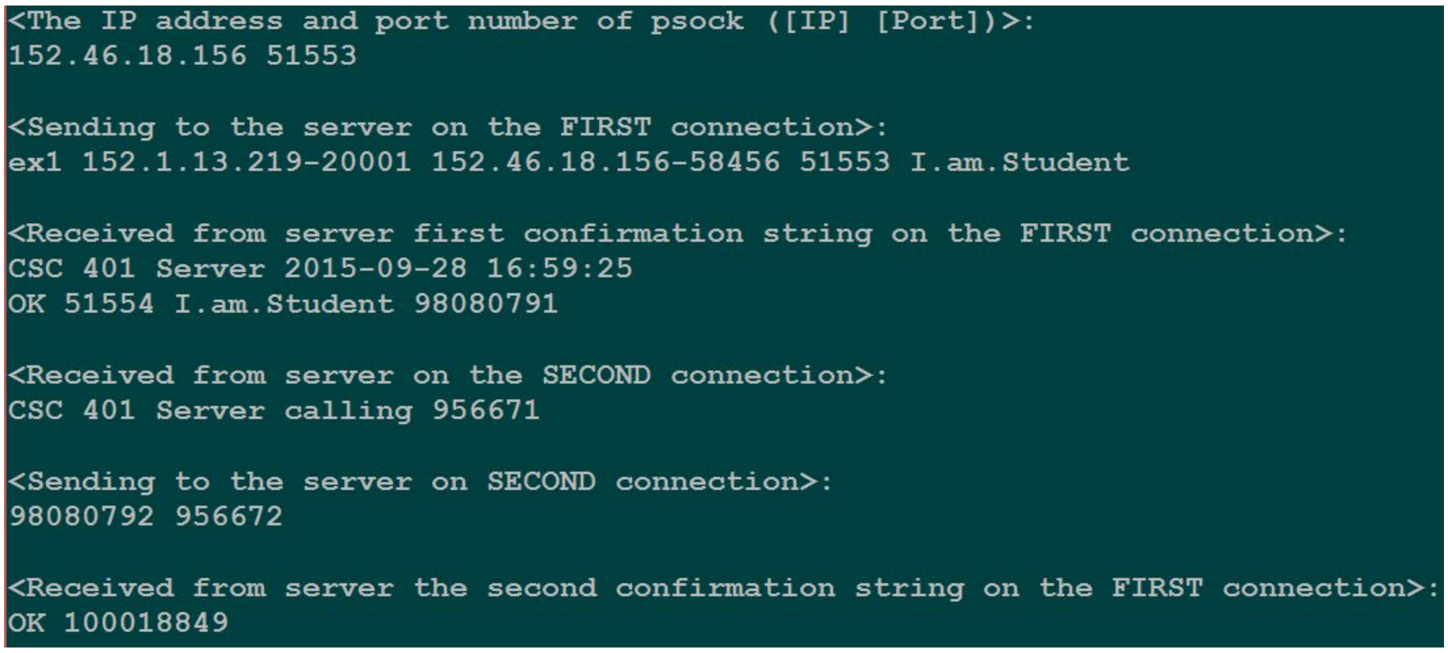
10. **`send()`** the string

    `<servernum+1> <WS> <newservernum+1> <newline>`

    on **`newsock`**, and then close the second connection.
11. Receive data on the original connection, printing out the result. Close the original connection and exit.

Note: Be sure to call **`listen()`** on **`psock`** before sending the initial request to the server. Otherwise, the server request may arrive before **`psock`** is ready to accept connections, and it will be refused.

4.3 Example Output



```
<The IP address and port number of psock ([IP] [Port])>:
152.46.18.156 51553

<Sending to the server on the FIRST connection>:
ex1 152.1.13.219-20001 152.46.18.156-58456 51553 I.am.Student

<Received from server first confirmation string on the FIRST connection>:
CSC 401 Server 2015-09-28 16:59:25
OK 51554 I.am.Student 98080791

<Received from server on the SECOND connection>:
CSC 401 Server calling 956671

<Sending to the server on SECOND connection>:
98080792 956672

<Received from server the second confirmation string on the FIRST connection>:
OK 100018849
```

5 Submission Details

The project is due Sunday, Oct 2nd, 2016 at 11:55 p.m. We will follow the late submissions from the syllabus. If you submit your work late, we will give you credit for it on this scale:

- 90% for work submitted up to 12 hours late

- 80% for work submitted up to 24 hours late
- 70% for work submitted up to 2 days late
- 0% for work submitted more than 2 days late

5.1 Team Work

You will be able (but not required) to work in teams of two. If you choose to work with a partner, there won't be any excuses in regard to partners not doing their share of the work.

5.2 Programming Language Choice

You can write your code either in Python (recommended) or Java. The textbook presents sockets programming in Python in Sec 2.7. On Moodle, you can find that same section from a previous version of the textbook with sockets programming shown in Java.

A tutorial on Python is available at http://docs.python.org/tutorial/
A guide for beginners to learn Python can be found at https://developers.google.com/edu/python/

However the above guide does not cover socket programming. The following is a good supplement: http://www.ibm.com/developerworks/linux/tutorials/l-pysocks/

Here is a reference for Java Sockets:
http://docs.oracle.com/javase/tutorial/networking/sockets/index.html

5.3 What to submit

Well-commented **source files** (python or java): ex0 and ex1. Put your name at the top of each source file.

Also submit a **README** text file. In the README file, please provide the following information:

- Your name(s)
- Your unityid(s)
- A **printscreen** of the information your client received from the CS 401 server when testing your code.
- **A detailed description on how to compile & run your code.**
- (Optional) Other things you want to tell us.

6 Questions

- If you have any problems, please post on Piazza. Questions helping to clarify the project specification, or the questions about general socket programming are welcome. However, please do not e-mail your code to us for debugging, instead go to office hours.
- If the server is down, please let us know immediately via a direct email.