Overview

In this Lab assignment you will develop a simple client program written in **Python(use a socket library)** to interact with a “CSE 3300 Server” running on a remote machine. The server, running on the machine tao.ite.uconn.edu, waits for TCP connections on port 3300. When a client establishes a connection to that port, the server waits for the client to initiate communication.

This assignment has two parts. For the first part, you will create the client. For the second part, you will modify the first part and write additional code to extend the interaction.

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 UNIX sockets interface.

Setup and Preparation

To complete this assignment, you may use any UNIX system that is connected to the department’s network (or use our VPN solution vpn.uconn.edu).

This assignment requires that you have some understanding of the use of the sockets API for connection-oriented interprocess communication. You will definitely need to read carefully the manual (“man”) pages for the **socket**, **connect**, **send**, **recv**,** close**, **bind**, **listen**, and **accept** routines, if you are not already familiar with them.

Since computers can be either big-endian or little-endian, care must be taken to ensure that multi-byte integers sent onto the network are laid out in a standard network byte order, which may or may not be the same as host byte order. Several functions are available for this purpose: **ntohs()**, **ntohl()**, **htons()**, **htonl()**, etc.

**Note**: To read the man page for the “foo” routine, type **man foo**. This will print the first man page encountered that is titled “foo”. However, if you want the man page for the “foo” socket routine and there’s also a command **foo**, you have to specify the section of the manual you want. Thus if you want the socket routine “send”, you type **man send** on a Linux machine, because section 2 contains manual pages for system calls. To learn how to use the command man, you should type **man man**. There’s a better way to browse manual pages if you use the X window system: **xman** provides a nice point-and-click interface, with search capability.

Exercise 0: Four-way handshake

In this exercise you will create a client program and use it to interact with the CSE 3300 Server according to a four-way handshake protocol: The client begins by sending a **request**, the server sends back a **confirmation**; then the client sends an **ack**, and the server replies with an **ack**. The protocol is specified next.

**The Protocol (Exercise 0)**

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

1. The server “listens” for connections on port 3300 of tao.ite.uconn.edu (IP address 137.99.3.212).

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:


   where:
   
   • The (request type) is a string of the form “ex\text{\textquoteleft\textquoteleft}i\text{\textquoteleft\textquoteleft}”, where \textit{i} is the exercise number (i.e. for this exercise the string would be \textit{ex0}). Case is ignored in this string.
   • (WS) is “whitespace”, one or more blank or tab 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 1000 to 8000.
   • 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, “E.W.Dijkstra”).
   • (newline) is the end-of-line marker, represented in C language by the single character ‘\n’.

   Thus, an example of a client request is:

   ex0 137.99.11.9-3300 137.99.10.7-10323 4321 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 (“CSE 3300 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):

   CSE 3300 Server Tue Feb 10 16:29:00 CDT 2005\n
---

1 To increase availability and spread the load, an identical server runs on port 3301. Your client may access any one of the servers.
OK 4322 I.M.Student 1601812701

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+2) (WS) (servernum+1) (newline)

Thus an example of a client ack is:

ex0 4323 1601812702

7. Upon receiving the client ack, the server sends back an ack string terminated by ‘\n’, containing its identification (“CSE 3300 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:

CSE 3300 Server Tue Feb 10 16:29:02 CDT 2005 OK 1601812702

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. When the server sees that the client has closed the connection, it closes also, and goes on to service the next client request.

**Client Operation**

To implement the above protocol, the client does the following:

(i) Create a socket (of type `SOCK_STREAM` and family `AF_INET`) using `socket()`.

(ii) 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.

(iii) Generate an integer from 1000 to 8000 randomly (usernum). Construct a request string using the server and client address information. To determine the client address information, the client must use the `gethostname()` function.

(iv) Write the client request string to the socket (using `send()`).

(v) 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.

(vi) Construct an ack string and write it to the socket (using `send()`).

(vii) 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.

(viii) Close the connection (using `close()`).
Writeup

Turn in your well-commented code along with a record of the information your client received from the CSE 3300 server, including the server numbers obtained in steps 5 and 7 of the protocol. (Please turn in the zip named “Firstname Lastname.zip”.)

Exercise 1: Listening for a new connection

In this part of the 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 second endpoint of the connection specifier as the target to which it (the server) should initiate a second connection. The server first sends the confirmation (including the server’s random number) on the original connection, and then tries to open a new connection to the indicated endpoint. 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. Remember, you are requiring to accept an incoming connection to your computer, so you will need public IP address with firewall disabled, or use UConn VPN to get one.

The Protocol (Exercise 1)

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 second endpoint-specifier refers to a socket different from the client side socket of the original connection. In other words, the client has multiple 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 to the second endpoint address specified in the client request. This address must refer to a host and port on which the client is “listening” for a new connection. That is, after sending the request string, the client must be prepared to accept a new connection on the second endpoint address in the request. Note that this port number (endpoint) must be different from the port number (endpoint) on which the client is originally connected to the server.

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:

   CSE 3300 server calling (WS) (newservernum) (newline)

   (Subsequently, the CSE 3300 server stops sending by calling shutdown with argument 1.)

8. After receiving the string, the client prints CSE 3300 server sent (newservernum) and then sends back on the second connection the following string

   (servernum+1) (WS) (newservernum+1) (newline)

   where (servernum) is the random number previously received from the server in the confirmation string (second line). 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.

“Client” Operation

The steps followed by the client for the protocol of Exercise 1 are:

a. Create a socket (call it psock) of type SOCK_STREAM and family AF_INET using socket().

b. Bind the socket psock to an available port, using bind(). (Note that by calling bind() with the special IP address INADDR_ANY and port number 0, the system will bind the socket to a “random” available port on the host.)

c. Find out what port psock was bound to using the getsockname() system call.² Print out the address and port, so the user can see what’s going on.

d. Call listen() on psock so it will accept new connections.

e. 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.

f. Open the first connection to the server and send the request.

g. Receive the confirmation string from the server on the first connection opened in step f; 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.

h. 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.

i. Call recv() on newsock to get the new random number from server. If the random number is received, print the line

   CSE 3300 server sent (WS) (newservernum)

j. send() the string

   (servernum+1) (WS) (newservernum+1) (newline)

   on newsock, and then close the second connection.

k. 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.

Writeup

Turn in your well-commented code along with a record of the information your client received from the

² Note that the IP part of the socket address will be specified as 0 (wildcard) if that is what was specified when it was bound. To determine the IP address of the socket, you can use the getsockname() routine. Remember to use them on a connected socket (the first socket created in ex0, and after connect()).
CSE 3300 server, including all three server numbers obtained. (Please turn in the zip named “Firstname Lastname.zip”.)