

Project 1: A Client-Server Application

Network Architecture and Protocols (CS/ECE 5565)

Parth Siddharth Shroff (9061-06876)

Date Due: 10th October, 2017

Development Tools

Operating System: Windows 10 home (64-bit)

Code Editor: Notepad++ v7.5.1 (64-bit)

Programming Language: Python v3.6.3

All codes run on windows command prompt.

Abstract

This report is mainly based on the interaction between a HTTP server and a HTTP client and the consequent exchange of response messages between the two. Following this the client requests a file from the server and the server sends the file to the client.

Keywords: HTTP, server, client

Implementation

The client-server application for this project was implemented using python v3.6.3. Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

Client and Server Model

The application uses a number of functions to establish a socket for communication between client and server and the consequent file transfer between the two. In this application we are making use of TCP sockets.

The following are the steps for the server:

1. Create a socket object.
2. Bind the socket to a particular socket.
3. Listen for incoming connections from the clients.
4. Connect with a client.
5. Accept file request from client and send the file.
6. Close the socket.

The following are the steps for the client:

1. Create a socket object.
2. Ask to connect to a server socket.
3. Send a file request to the server, receive file, display contents and download the file.
4. Close socket.

Method

The client and server sockets are created using the following command:

$$S = \text{socket.socket}(\text{family type}, \text{socket type})$$

Here the family type can be either *AF_INET* or *AF_UNIX*, where *AF_INET* refers to the IPv4 address family and is the default type. The socket type can either be *SOCK_STREAM* or *SOCK_DGRAM*, where *SOCK_STREAM* is connection oriented(TCP) and the default type and *SOCK_DGRAM* is connectionless(UDP).

Server socket methods

The following methods were used in the server side socket:

- *s.bind()*: This method accepts hostname and the port number as the arguments and thereafter bind the server socket to that host and the corresponding port.
- *s.listen()*: This method activates the TCP listener and takes an integer as the argument. This integer indicates the number of clients that the server can accept at a time.
- *s.accept()*: This accepts incoming TCP connections from the clients waiting until any request for connection arrives.

Client socket methods:

The following methods were used in the client side socket:

- *s.connect()*: This method accepts hostname and port number as the arguments and connects the client to the said host at the mentioned port.

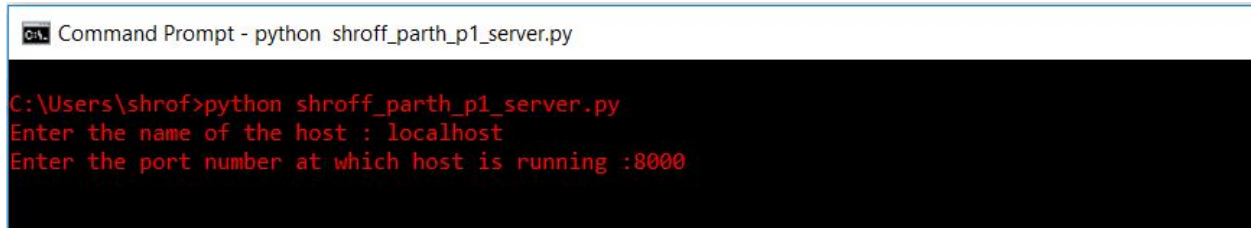
General socket methods used:

- *s.recv()*: This method is used to receive TCP messages.
- *s.send()*: This method is used to send TCP messages.
- *s.close()*: This method is used to close the socket.

Testing Procedure

Notepad++ was used to develop the python code and to do further editing. The version of python used to interpret the code for this application was python 3.6.3. All the codes were then run and tested on the windows command prompt.

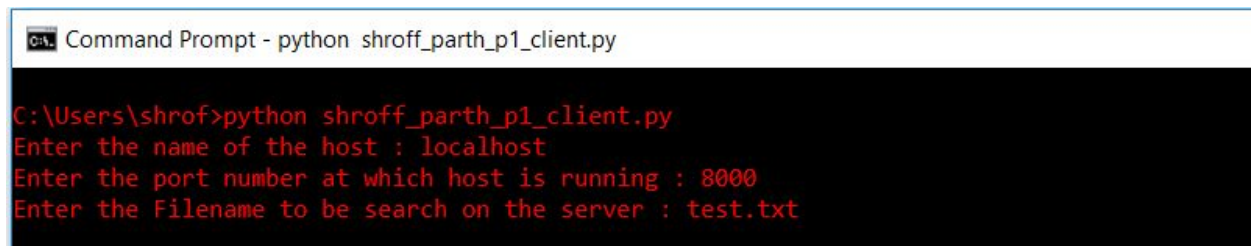
When the server side code is run on the command prompt, the user is prompted to enter the host and the port number to which the server has to bind.



```
C:\> Command Prompt - python shroff_parth_p1_server.py

C:\Users\shrof>python shroff_parth_p1_server.py
Enter the name of the host : localhost
Enter the port number at which host is running :8000
```

When the client side code is run on the command prompt, the user is prompted to enter the name of the host and the port number to which it wishes to connect and also the name of the file it wants from the server.



```
C:\> Command Prompt - python shroff_parth_p1_client.py

C:\Users\shrof>python shroff_parth_p1_client.py
Enter the name of the host : localhost
Enter the port number at which host is running : 8000
Enter the Filename to be search on the server : test.txt
```

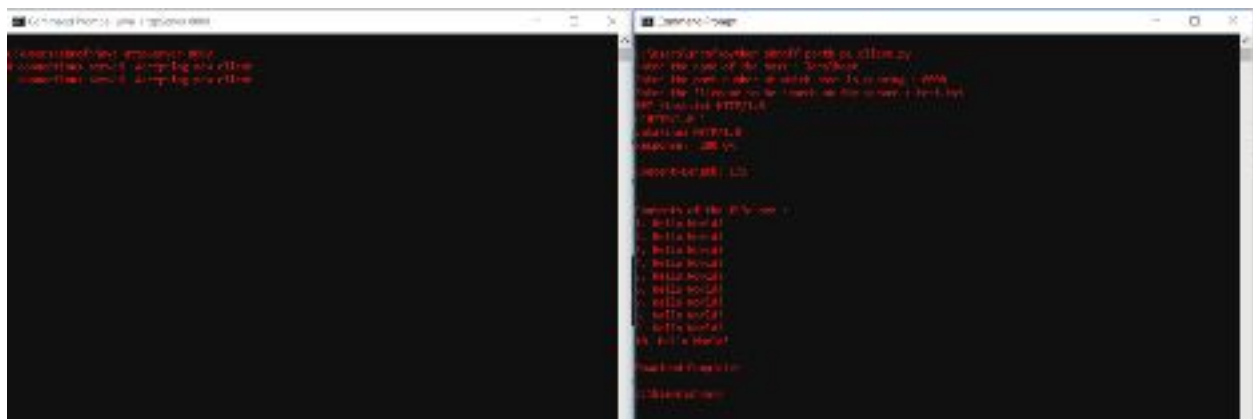
The student client is then tested with the testing server and the student server is tested with the testing client. Also the student client and server are run with each other. Depending on the request sent by the client the server responds with a response message. The results for the procedure are shown under test results.

Test Results

The following results were obtained when the servers and clients were tested

Outcome 1

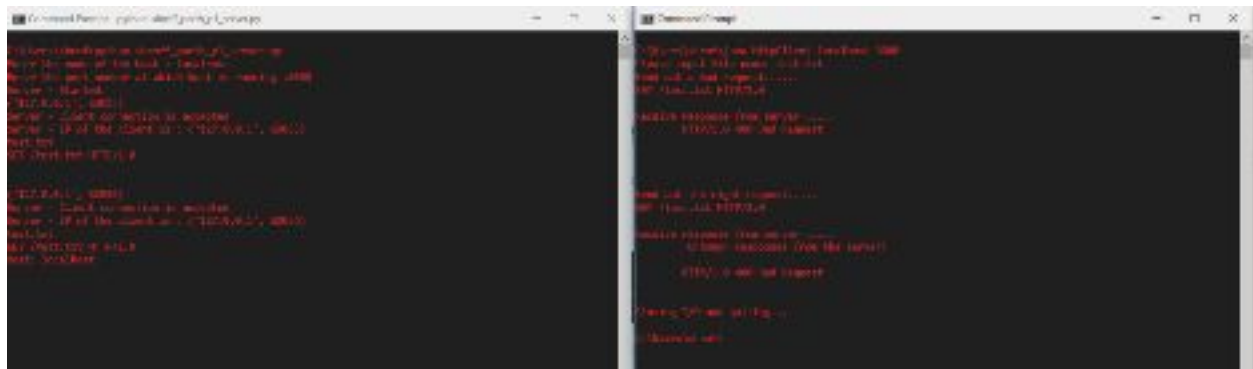
- ❖ Server: Testing
- ❖ Client: Student



```
Client:
$ ./client.py
Enter the name of the user: Student
Enter the message to be sent: hello world
Enter the IP address of the server: 127.0.0.1
Press Enter to send the message
Message sent successfully
Server:
$ ./server.py
Enter the port number: 8080
Server is running...
Message received from client: hello world
Server response: hello world
```

Outcome 2

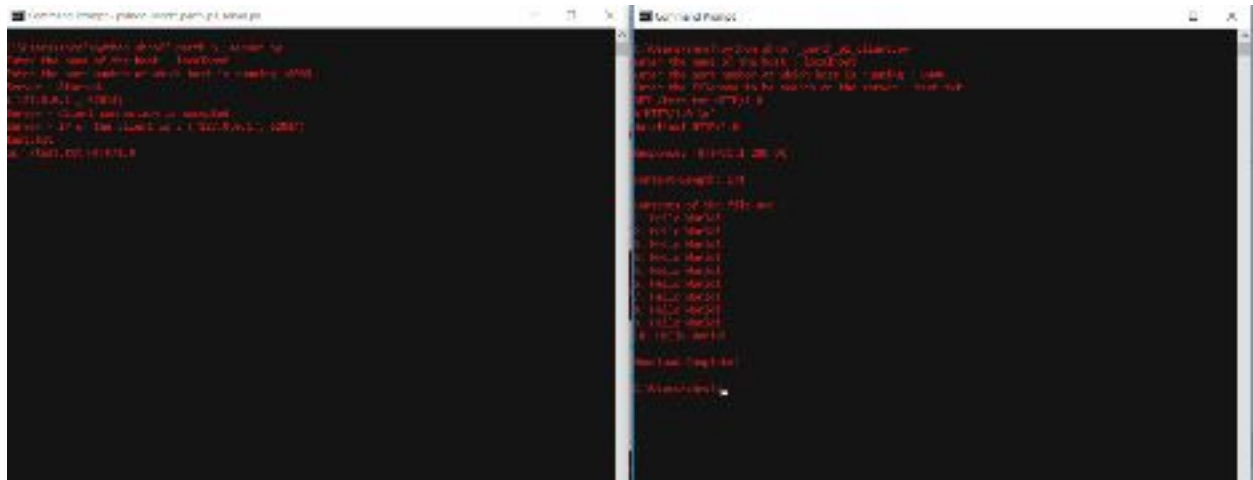
- ❖ Server: Student
- ❖ Client: Testing



```
Server:
$ ./server.py
Enter the port number: 8080
Server is running...
Message received from client: hello world
Server response: hello world
Client:
$ ./client.py
Enter the name of the user: Student
Enter the message to be sent: hello world
Enter the IP address of the server: 127.0.0.1
Press Enter to send the message
Message sent successfully
```

Outcome 3

- ❖ Server: Student
- ❖ Client: Student



```
Left Window (Client):
127.0.0.1:54321 -> 127.0.0.1:8080: GET / HTTP/1.1
127.0.0.1:54321 -> 127.0.0.1:8080: 200 OK (text/html)
127.0.0.1:54321 -> 127.0.0.1:8080: GET /index.html HTTP/1.1
127.0.0.1:54321 -> 127.0.0.1:8080: 200 OK (text/html)

Right Window (Server):
127.0.0.1:8080 -> 127.0.0.1:54321: GET / HTTP/1.1
127.0.0.1:8080 -> 127.0.0.1:54321: 200 OK (text/html)
127.0.0.1:8080 -> 127.0.0.1:54321: GET /index.html HTTP/1.1
127.0.0.1:8080 -> 127.0.0.1:54321: 200 OK (text/html)
```

Test Summary

When the testing server was run with the student client, both response and request messages were sent and interpreted appropriately and the file transfer followed by the download of the file was successful.

When the student server was run with the testing client, when a bad request was sent by the client, the server correctly interpreted it and sent the appropriate response. However when the client sent a good request, the server sent the corresponding response but the client somehow could not read this response and displayed unknown responses from the server. Hence the file transfer was not successful.

When the student server was run with the student client, both response and request messages were sent and interpreted appropriately and the file transfer followed by the download of the file was successful.

References

1. <https://docs.python.org/3/tutorial/index.html>
2. <https://docs.python.org/3/library/index.html>
3. <https://docs.python.org/3/reference/index.html>
4. <https://docs.python.org/3/howto/sockets.html>
5. http://www.bogotobogo.com/python/python_network_programming_server_client.php