Lab: Socket Programming[1]

Objective

Socket programming is a way of connecting two processes on a network to communicate with each other. One socket listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. In simpler terms, there is a server and a client. The server forms the listener socket while the client reaches out to the server. They are the real backbones behind web browsing.

The purposes of this lab are to

- introduce the concepts of client/server network programming using Python socket;
- develop a TCP socket client software that communicates with a TCP socket server software.

Tasks

Getting Python

To run a socket program in python, we need to download and install Python in your PC:

- Go to https://www.python.org/downloads/ and download the latest Python for windows.
- Install the Python into a directory, e.g., "c:\Python".

Getting Socket Programming Started

Socket programming is started by importing the socket library and making a simple socket.

```
import socket
s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address-family IPv4. The SOCK_STREAM means connection-oriented TCP protocol.

Connecting to a Server

We can connect to a server using this socket. Here is an example of a python program for connecting to Google. First of all, we made a socket.Then we resolved google's IP and lastly, we connected to google. Note that if any error occurs during the creation of a socket then a socket.error is thrown.

[1] Reference: Socket Programming in Python, https://www.geeksforgeeks.org/socket-programming-python/.

```
# an example script to connect to Google using socket programming in Python
import socket
import sys

try:
	s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	print ("socket successfully created")
except socket.error as err:
	print ("socket creation failed with error %s" %(err))

# default port for socket
port = 80

try:
	# find the IP of the Google's web server
	host_ip = socket.gethostbyname('www.google.com')
except socket.gaierror:
	# this means could not resolve the host
	print ("there was an error resolving the host")
	sys.exit()

# connecting to the server
s.connect((host_ip, port))

print ("the socket has successfully connected to google")
```

Now save this file as “GoogleClient.py” in your working directory and run it from the window terminal:

```
d:\lab> c:\Python\python GoogleClient.py
```

Question 1: What is the output when running this python program? Screen capture the executing result.

A Simple Client-Server Program

- Server program:

A server has a bind() method which binds it to a specific IP and port so that it can listen to incoming requests on that IP and port. A server has a listen() method which puts the server into listening mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.

Here is a python program "TCPServer.py" for the server:

```
# import the socket library
import socket

# create a socket object
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print ("socket successfully created")

# reserve a port=12345 on your computer
serverPort = 12345

# bind to the port
# we have not typed any ip in the ip field, instead we have inputted an empty string.
# this makes the server listen to requests coming from other computers on the network
serverSocket.bind(('', serverPort))
print ("socket binded to %s" %(serverPort))

# put the socket into listening mode
serverSocket.listen(5)
print ("socket is listening")

# a forever loop until we interrupt it or an error occurs
while True:
    # establish connection with client.
    connectionSocket, addr = serverSocket.accept()
    print ('got connection from', addr )

    # send a message to the client, using encode() to send byte type
    sentence='thank you for connecting'
    connectionSocket.send(sentence.encode())

    # close the connection with the client
    connectionSocket.close()
    break
```

* First of all, we import socket which is necessary.

* Then we made a socket object and reserved a port on our PC.
* After that, we bind our server to the specified port. Passing an empty string as the IP address means that the server can listen to incoming connections from other computers as well. If we have passed ‘127.0.0.1’ as the IP address, then it will listen to only those calls made within the local computer.
* After that we put the server into listening mode. 5 here means that 5 connections can be kept. If the server is busy for 5 connections and a 6th socket tries to connect, then the connection is refused.
* At last, we make a while loop and start to accept all incoming connections and close those connections after a thank you message is sent to the connected socket.

- Client program:

Here is a python program “TCPClient.py” for the client:

```
# import the socket library
import socket

# create a socket object
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# define the server's name and port on which you want to connect
serverName = '127.0.0.1'
serverPort = 12345

# connect to the server
clientSocket.connect((serverName, serverPort))

# receive data from the server and decode to get the string.
sentence = clientSocket.recv(1024).decode()
print ("from server:", sentence)

# close the connection
clientSocket.close()
```

* First of all, we make a socket object.
* Then we connect to localhost on port 12345 (the port on which our server runs)
* Lastly, we receive data from the server and close the connection.

Run the Python programs in two separated window terminals:

```
d:\lab> c:\Python\python TCPServer.py
```

&

```
d:\lab> c:\Python\python TCPClient.py
```

Question 2: What are the outputs when running these Python programs? Screen capture the executing result.

Congratulation! You have successfully run both client socket and server socket programs on your computer. You will be surprised to know that all applications on the Internet, such as Telnet, FTP, web, etc., are written in a similar manner. Once you are comfortable with this exercise, we will be ready to tackle the socket programming on both the client and server sides.

More Tasks

Question 3: Modify the Python program "TCPClient.py". Use a new socket port number for the client's connected socket to the server socket. You need to use 5-digit number "4xxxx" as your new socket port number, where "xxxx" is the last four digits of your student ID. Your program needs to print out the TCP connection (TCP socket's four tuples) established between the client and server processes. Run the Python programs in two separate window terminals and screen capture the executing result. Attach your modified program code.

Question 4: Modify the Python programs "TCPClient.py" and "TCPServer.py". Add a function to allow the client user to input the string of 5-digit number "xxxxx" as the password to the server process, where "xxxxx" is the last five digits of your student ID. When the user inputs the correct 5-digit number, the server process will send a message of "Your password is correct!" to the client process; otherwise, the server process will send a message of "Your password is incorrect!" to the client process. Run the Python programs in two separate window terminals and screen capture the executing result. Attach your modified program codes.