



Faculty of Engineering and Technology
Department of Electrical and Computer Engineering

ENCS436
COMPUTER NETWORKS
Project #1 Report
Socket Programming

Prepared by:
Haitham Daana 1121331
Mohammad Ibrahim Mousa 1141618
Section #3

Instructor: Dr. Iyad Tumar
5/12/2017

Abstract

This report represents simple socket program, it's a simple client-server program that works on both of TCP/UDP using Python Language. The server when it receives a message/text, it replies by reversing all characters, and reverses all the capitalization of the strings.

Contents

Abstract.....	2
Introduction	4
Socket in Python	5
Methodology of Design.....	6
Results and Discussion	7
User Datagram Protocol (UDP)	7
Transmission Control Protocol (TCP)	9
Conclusion.....	12
References	13

Introduction

network socket is an endpoint of an inter-process communication flow across a computer network. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Today, most communication between computers is based on the internet protocol; therefore, most network sockets are internet sockets. To create a connection between machines.

Socket in Python

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, SMTP, and so on.

Sockets may be implemented over a number of different channel types: UNIX domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Methodology of Design

As mentioned earlier, Python has many libraries that can be used to provide interfacing between network connections. For example, **`socket.socket()`** is used to open a socket: It has the following syntax:

```
s = socket.socket (socket_family, socket_type, protocol = 0)
```

<i>s.bind()</i>	This function binds address (hostname, port number pair) to socket.
<i>s.listen()</i>	This method sets up and start TCP listener.
<i>s.accept()</i>	This passively accept TCP client connection, waiting until connection arrives (blocking).
<i>s.connect()</i>	This function actively initiates TCP server connection.
<i>s.recv()</i>	This method receives TCP message
<i>s.send()</i>	This method transmits TCP message
<i>s.recvfrom()</i>	This method receives UDP message
<i>s.sendto()</i>	This method transmits UDP message
<i>s.close()</i>	This method closes socket
<i>socket.gethostname()</i>	Returns the hostname

The program, was created using two types of connections. One using TCP and the other was UDP. The server listens to the clients, waiting for a connection. In case of TCP there is a handshaking process that takes place, before exchanging data, but in case of UDP there is an open connection between host and client, it's much faster. Then It takes the message from the client, and reverses all characters, and the capitalization of the text string sent, and sends it back to the client.

Results and Discussion

User Datagram Protocol (UDP)

UDP is a simple transport protocol that extends the host-to-host delivery of packets of the underlying network into a process-to-process communication. Since there are many processes running on a given host (e.g. multiple Internet browsers), UDP needs to add a level of demultiplexing, allowing multiple application processes on each host to share the network. Therefore, the only interesting issue in UDP is the form of address used to identify a process. Although it is possible to directly identify a process with the operating system (OS) assigned id (pid), such an approach is only practical in a close distributed system with one OS that assigns unique ids to all processes (does not work for the entire world!). Instead, a process is indirectly identified using an abstract locator, often called a port. A source process sends a message to a port, and a destination process receives a message from a port. The UDP header contains a 16 bit port number for both the sender (source) and the receiver (destination).

We see TCP Flow in Figer.2

server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

↓
read datagram from
serverSocket

↓
write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

↓
Create datagram with server IP and
port=x; send datagram via
clientSocket

↓
read datagram from
clientSocket
↓
close
clientSocket

Figure.2 (Client/server socket interaction: UDP)

Code for UDP Client and Server

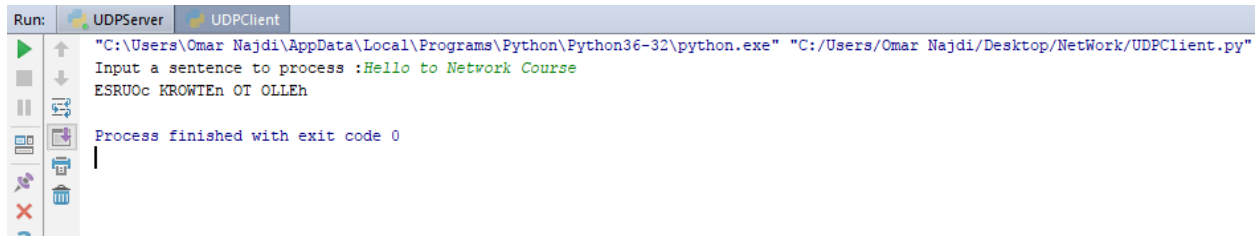
```

1  #UDP Client
2  from socket import *
3  serverName = "127.0.0.1"
4  serverPort = 12000
5  #create UDP socket for server
6  clientSocket = socket(AF_INET, SOCK_DGRAM)
7  #get user keyboard input
8  message = input("Input a sentence to process :")
9  #Attach server name, port to message; send into socket
10 clientSocket.sendto(message.encode(), (serverName, serverPort))
11 #read reply characters from socket into string
12 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
13 #print out received string and close socket
14 print(modifiedMessage.decode())
15 clientSocket.close()
16
17

TCPClient.py x  TCPServer.py x  UDPCient.py x  UDPSever.py x

1  #UDP Server
2  from socket import *
3  serverPort = 12000
4  #create UDP socket
5  serverSocket = socket(AF_INET, SOCK_DGRAM)
6  #bind socket to local port number 12000
7  serverSocket.bind(('', serverPort))
8  #print("The server ready to receive")
9
10 while True:
11     #Read from UDP socket into message, getting client's address (client IP and port)
12     message, clientAddress = serverSocket.recvfrom(2048)
13     #reverse message
14     rev_message=message[::-1]
15     # swap case from Upper to Lower And vice versa
16     capi_message = rev_message.swapcase()
17     modifiedMessage = capi_message.decode()
18     #send Final string back to this client
19     serverSocket.sendto(modifiedMessage.encode(), clientAddress)
20
```


Output



```
Run: UDPServer UDPCliet
"C:\Users\Omar Najdi\AppData\Local\Programs\Python\Python36-32\python.exe" "C:/Users/Omar Najdi/Desktop/NetWork/UDPCliet.py"
Input a sentence to process :Hello to Network Course
ESRUOc KROWTEh OT OLLEh
Process finished with exit code 0
```

Transmission Control Protocol (TCP)

In contrast to a simple demultiplexing protocol like UDP, a more sophisticated transport protocol in one that offers reliable communication. The Internet's Transmission Control Protocol (TCP) is probably the most widely used protocol of this type. As a transport protocol, TCP provides reliable, in order delivery of messages. It is a full duplex protocol, meaning that each TCP connection supports a pair of streams, one flowing in each direction. It also includes a flow control mechanism for each of these streams that allows the receiver (on both ends) to limit how much data the sender can transmit at a given time (we will look at this feature later). Of course, TCP supports the demultiplexing mechanism of UDP to allow multiple application programs on a given host to simultaneously communicate over the Internet. However, the demultiplexing key used by TCP is the 4-tuple "source port, source host, destination port, destination host " to identify the particular TCP connection.

At the heart of TCP is the sliding window algorithm. Even though this is the same basic algorithm we have seen before for DLC, because TCP runs over the network rather than a single link, there are many important differences that complicate TCP

We see TCP Flow in Figer.1

server (running on hostid)

client

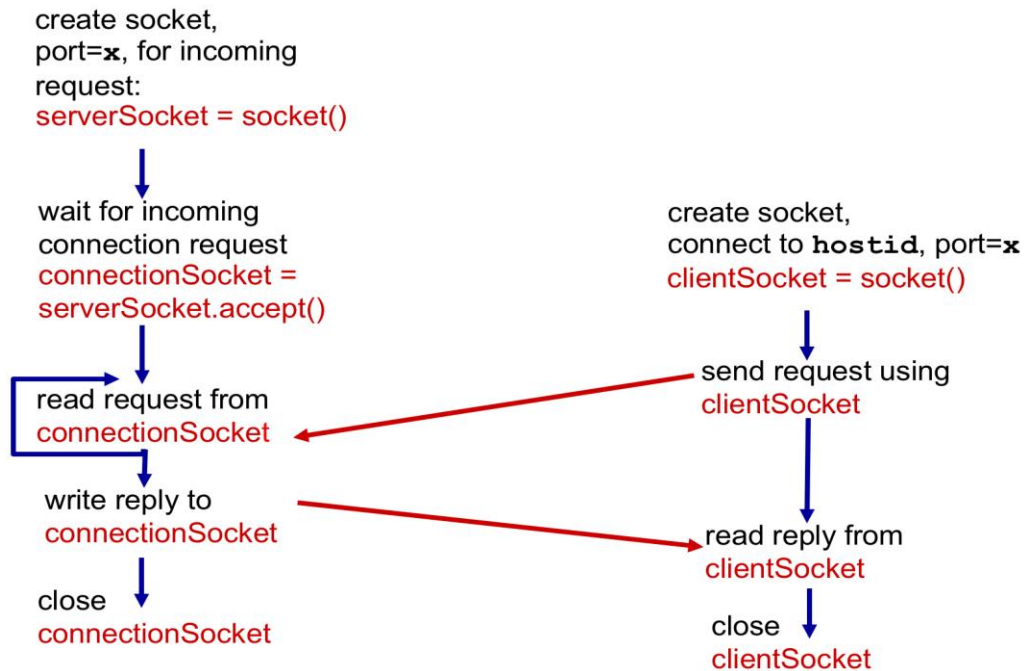


Figure.1 (Client/server socket interaction: TCP)

Code for TCP Client and Server

```
1  #TCP Client
2  from socket import *
3  serverName = "127.0.0.1"
4  serverPort = 13000
5  #create TCP socket for server, remote port 13000
6  clientSocket = socket(AF_INET, SOCK_STREAM)
7  clientSocket.connect((serverName, serverPort))
8  sentence = input("Input a sentence to process :")
9  #No need to attach server name, port
10 clientSocket.send(sentence.encode('utf-8'))
11 modifiedSentence = clientSocket.recv(1024)
12 print("From Server:", modifiedSentence.decode())
13 clientSocket.close()
```

```

TCPClient.py x TCPServer.py x UDPClient.py x UDPServer.py x
1  #TCP Server
2  from socket import *
3  serverPort = 13000
4  #create TCP welcoming socket
5  serverSocket = socket(AF_INET, SOCK_STREAM)
6  serverSocket.bind(("", serverPort))
7  #server begins listening for incoming TCP requests
8  serverSocket.listen(1)
9  #loop forever
10 while True:
11     #server waits on accept() for incoming requests, new socket created on return
12     connectionSocket, addr = serverSocket.accept()
13     #read bytes from socket
14     sentence = connectionSocket.recv(1024)
15     #reverse message
16     rev_sentence = sentence[::-1]
17     # swap case from Upper to Lower And vice versa
18     capi_sentence = rev_sentence.swapcase()
19     final_output = capi_sentence
20     connectionSocket.send(final_output)
21     #close connection to this client
22     connectionSocket.close()

```

Output

```

Run: TCPServer TCPClnt
"C:\Users\Omar Najdi\AppData\Local\Programs\Python\Python36-32\python.exe" "C:/Users/Omar Najdi/Desktop/NetWork/TCPClnt.py"
Input a sentence to process :Hello World!
From Server: !DLROw OLLEh
Process finished with exit code 0

```

Conclusion

By the end of this mini-project, we were able to code in python for first time, understood the basic programming skills needed for socket programming and comprehended the main concepts of networking in that matter. Which will help us in upper Networking courses, or in programming applications that work online.

References

[1]: Jim Kurose and Keith Ross. ***Computer Networking: A Top Down Approach***, 7th Edition, Addison-Wesley, 2013.

[2]:<http://www.biogem.org/downloads/notes/Socket%20Programming%20in%20Python.pdf>

[3]:<https://pdfs.semanticscholar.org/18f2/8c21f0b94d072c9727f670357b26a43ff735.pdf>