

SIMPLE CLIENT SERVER APPLICATION DEVELOPMENT USING PYTHON

DECIMAL TO HEXADECIMAL CONVERSION

- **CB.EN.U4ECE18107 (ARUNESWARI S)**

THEORY:

SOCKET:

Socket programming is a method of allowing two network nodes to communicate with one another. One socket (node) listens on a specific port at an IP address, while the other socket connects with it. While the client connects to the server, the server creates the listener socket. A socket has a port number that allows the TCP/UDP layer to identify the application to which data is being transmitted. A combination of an IP address and a port number makes up an endpoint.

CLIENT- SERVER ARCHITECTURE:

The design of a computer network in which numerous clients (remote processors) request and receive services from a centralised server is known as client-server architecture (host computer). Client computers provide an interface that allows a computer user to request server services and view the results returned by the server. Servers wait for requests from clients to arrive before responding. Clients need not be aware of the specifics of the system (i.e., the hardware and software) that is providing the service if a server provides a standardised transparent interface. Clients are typically found at workstations or on personal computers, while servers are typically found elsewhere on the network, on more powerful devices. This computing architecture is especially useful when clients and servers each have separate duties to complete on a regular basis.

TRANSPORT LAYER PROTOCOLS – TCP, UDP:

TCP - Transmission Control Protocol	UDP - User Datagram Protocol
TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	The Datagram Oriented Protocol (UDP) is a datagram-oriented protocol. This is due to the lack of expense associated with creating, maintaining, and terminating connections. For broadcast and multicast network transmission, UDP is a good choice.
The Transmission Control Protocol has a function that allows data to be sequenced (TCP). This means that packets arrive to the receiver in the sequence they were sent.	In UDP, there is no data sequencing. If ordering is required, the application layer must handle it.
TCP is reliable as it guarantees delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.

TCP is comparatively slower than UDP.	UDP is faster, simpler and more efficient than TCP.
TCP allows for the retransmission of dropped packets, however UDP does not.	The User Datagram Protocol does not allow for the retransmission of lost packets (UDP).
TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.

SERVER PROCESS:

A server's bind() method connects it to a specific IP address and port, allowing it to listen for incoming requests on that address and port. Listen() is a method on a server that puts it in listen mode. This enables the server to receive incoming connections and listen for them. Finally, a server has two methods: accept() and close(). The accept method initiates a connection with the client and the close method closes the connection with the client.

- Firstly, we imported the module socket.
- Then we made a socket object and reserved a port on our pc for this specific application.
- After that, we binded our server to the specified port. Passing an empty string means that the server can listen to incoming connections from other computers as well. If we would have passed 192.168.29.158, then it would have listened to only those calls made within the local computer. In this way the server can listen to requests from multiple systems or clients.
- After that we put the server into listening mode.
- Finally, we make a while loop and start to accept all incoming connections.
- Server sends back the hexadecimal equivalent of the integer sent by the client.

CLIENT PROCESS:

- Firstly, we made a socket object.
- Then we connected to localhost on specified port (the port on which our server runs/listens) and lastly, we received data from the server and closed the connection.
- Now we saved this file and ran it from the terminal after starting the server script

SOURCE CODE:

TCP SERVER:

```
import socket

HOST_ADDRESS = '192.168.29.158'
PORT_NUMBER = 99
while True:
```

```

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    print("Socket has been successfully created")
    s.bind(('',PORT_NUMBER))
    print("Socket is binded to %s" % (PORT_NUMBER))
    s.listen()
    print("Socket is listening")
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            xx = data.decode("utf-8")
            n=int(xx)
            b=[]
            hexaDeciNum = ['0'] * 100
            i = 0
            while (n != 0):
                temp = 0
                temp = n % 16

                if (temp < 10):
                    hexaDeciNum[i] = chr(temp + 48)
                    i = i + 1
                else:
                    hexaDeciNum[i] = chr(temp + 55)
                    i = i + 1
                n = int(n / 16)

            j = i - 1

            while (j>=0):
                b.append(hexaDeciNum[j])
                j = j - 1

            str1 = ""
            for elem in b:
                str1 += elem

            res = str(str1)
            conn.send(res.encode("utf-8"))

```

TCP CLIENT:

```
import socket

HOST_ADDRESS = '192.168.29.158'
PORT_NUMBER = 99

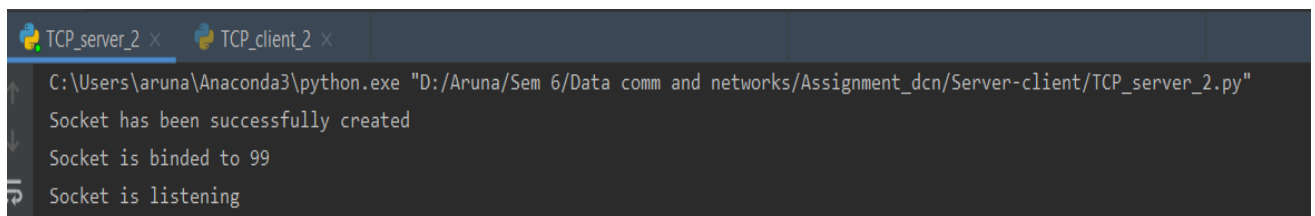
ss = input('Enter an integer: ')
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST_ADDRESS, PORT_NUMBER))
    s.send(bytes(ss, 'utf-8'))
    data = s.recv(1024)
    res = data.decode("utf-8")
    str_res = str(res)

print('Result (in Hexadecimal) :', str_res)
```

OUTPUT:

TCP server:

Socket has been successfully created
Socket is binded to 99
Socket is listening
Connected by ('192.168.29.158', 64553)

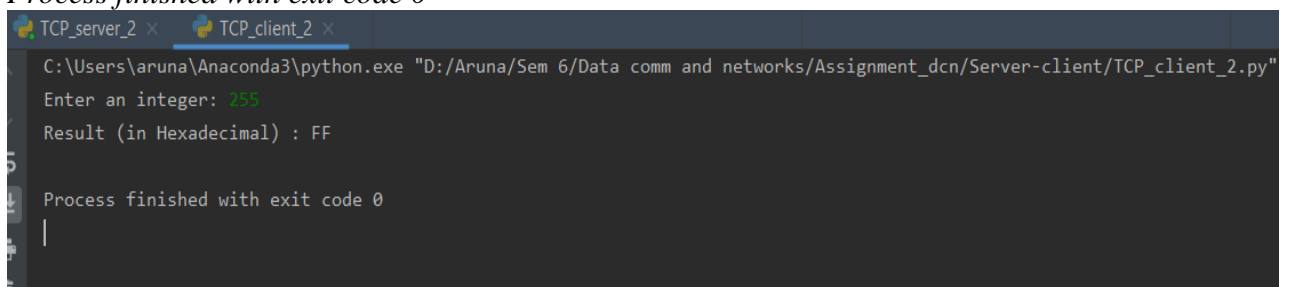


```
TCP_server_2 x TCP_client_2 x
C:\Users\aruna\Anaconda3\python.exe "D:/Aruna/Sem 6/Data comm and networks/Assignment_dcn/Server-client/TCP_server_2.py"
Socket has been successfully created
Socket is binded to 99
Socket is listening
```

TCP client:

Enter an integer: 255
Result (in Hexadecimal) : FF

Process finished with exit code 0



```
TCP_server_2 x TCP_client_2 x
C:\Users\aruna\Anaconda3\python.exe "D:/Aruna/Sem 6/Data comm and networks/Assignment_dcn/Server-client/TCP_client_2.py"
Enter an integer: 255
Result (in Hexadecimal) : FF

Process finished with exit code 0
|
```

UDP SERVER:

```
import socket
HOST_ADDRESS = '192.168.29.158'
PORT_NUMBER = 199
while True:
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.bind((HOST_ADDRESS, PORT_NUMBER))
        while True:
            data, addr = s.recvfrom(1024)
            #print('Connected By', addr)
            print('Successfully connected by', addr)
            if not data:
                break
            xx = data.decode("utf-8")
            n = int(xx)
            b = []
            hexaDeciNum = ['0'] * 100
            i = 0
            while (n != 0):
                temp = 0
                temp = n % 16

                if (temp < 10):
                    hexaDeciNum[i] = chr(temp + 48)
                    i = i + 1
                else:
                    hexaDeciNum[i] = chr(temp + 55)
                    i = i + 1
                n = int(n / 16)

            j = i - 1

            while (j >= 0):
                b.append(hexaDeciNum[j])
                j = j - 1

            str1 = ""
            for elem in b:
                str1 += elem

            res = str(str1)
            s.sendto(res.encode(), addr)
```

UDP CLIENT:

```
import socket

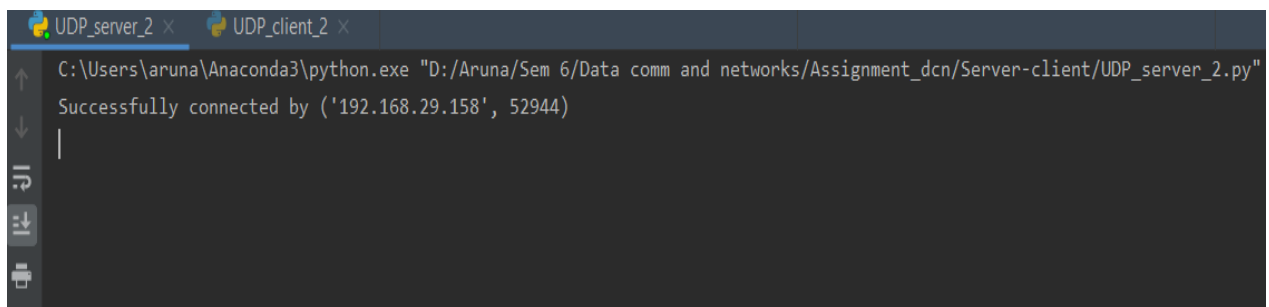
HOST_ADDRESS = '192.168.29.158'
PORT_NUMBER = 199

ss = input('Enter an integer: ')
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.sendto(bytes(ss, 'utf-8'), (HOST_ADDRESS, PORT_NUMBER))
    data = s.recv(1024)
    res = data.decode("utf-8")
    str_res = str(res)

print('Result (in Hexadecimal) :', str_res)
```

OUTPUT:

UDP server:

A screenshot of a terminal window with two tabs: 'UDP_server_2' and 'UDP_client_2'. The 'UDP_server_2' tab is active. The terminal shows the command prompt 'C:\Users\aruna\Anaconda3\python.exe "D:/Aruna/Sem 6/Data comm and networks/Assignment_dcn/Server-client/UDP_server_2.py"', followed by the output 'Successfully connected by ('192.168.29.158', 52944)' and a cursor on the next line.

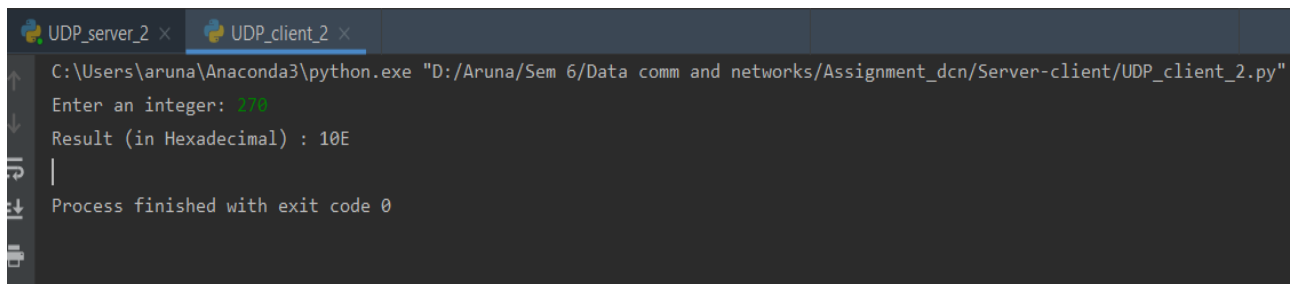
```
C:\Users\aruna\Anaconda3\python.exe "D:/Aruna/Sem 6/Data comm and networks/Assignment_dcn/Server-client/UDP_server_2.py"
Successfully connected by ('192.168.29.158', 52944)
|
```

UDP client:

Enter an integer: 270

Result (in Hexadecimal) : 10E

Process finished with exit code 0

A screenshot of a terminal window with two tabs: 'UDP_server_2' and 'UDP_client_2'. The 'UDP_client_2' tab is active. The terminal shows the command prompt 'C:\Users\aruna\Anaconda3\python.exe "D:/Aruna/Sem 6/Data comm and networks/Assignment_dcn/Server-client/UDP_client_2.py"', followed by the input 'Enter an integer: 270', the output 'Result (in Hexadecimal) : 10E', and finally 'Process finished with exit code 0'.

```
C:\Users\aruna\Anaconda3\python.exe "D:/Aruna/Sem 6/Data comm and networks/Assignment_dcn/Server-client/UDP_client_2.py"
Enter an integer: 270
Result (in Hexadecimal) : 10E
|
Process finished with exit code 0
```

CONCLUSION:

During the course of this assignment, I gained a better understanding of the notion of communication utilising TCP and UDP, as well as the differences between TCP and UDP. I learned about the benefits and drawbacks of these two protocols, as well as how to employ them effectively for various scenarios. I learned more about IP address classes and looked into the domains of texting programmes.

REFERENCES:

1. James Kurose Keith Ross - Computer Networks : A Top Down Approach
2. <https://www.geeksforgeeks.org/socket-programming-python/>
3. <https://www.britannica.com/technology/client-server-architecture>