Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

JUNAID GIRKAR
60004190057
TE COMPS A-4

Experiment No: 10

Aim: Implementation of Simple socket programming for message exchange between server and client.

Theory:

Sockets:

Sockets are the backbone of networking. They make the transfer of information possible between two different programs or devices. For example, when you open up your browser, you as a client are creating a connection to the server for the transfer of information.

In general terms, sockets are interior endpoints built for sending and receiving data. A single network will have two sockets, one for each communicating device or program. These sockets are a combination of an IP address and a Port. A single device can have 'n' number of sockets based on the port number that is being used. Different ports are available for different types of protocols. Take a look at the following image for more about some of the common port numbers and the related protocols:

Socket Programming in Python:

To achieve Socket Programming in Python, you will need to import the **socket** module or framework. This module consists of built-in methods that are required for creating sockets and help them associate with each other.

Some of the important methods are as follows:

Methods	Description
socket.socket()	used to create sockets (required on both server as well as client ends to create sockets)

socket.accept()	used to accept a connection. It returns a pair of values (conn, address) where conn is a new socket object for sending or receiving data and address is the address of the socket present at the other end of the connection
socket.bind()	used to bind to the address that is specified as a parameter
socket.close()	used to mark the socket as closed
socket.connect()	used to connect to a remote address specified as the parameter
socket.listen()	enables the server to accept connections

Server

A server is either a program, a computer, or a device that is devoted to managing network resources. Servers can either be on the same device or computer or locally connected to other devices and computers or even remote. There are various types of servers such as database servers, network servers, print servers, etc.

Servers commonly make use of methods like socket.socket(), socket.bind(), socket.listen(), etc to establish a connection and bind to the clients.

AF_INET refers to Address from the Internet and it requires a pair of (host, port) where the host can either be a URL of some particular website or its address and the port number is an integer. SOCK_STREAM is used to create TCP Protocols.

The bind() method accepts two parameters as a tuple (host, port). However, it's better to use 4-digit port numbers as the lower ones are usually occupied. The listen() method allows the server to accept connections. Here, 5 is the queue for multiple connections that come up simultaneously. The minimum value that can be specified here is 0 (If you give a lesser value, it's changed to 0). In case no parameter is specified, it takes a default suitable one.

The while loop allows accepting connections forever. ‘clt’ and ‘adr’ are the client object and address. The print statement just prints out the address and the port number of the client socket. Finally, clt.send is used to send the data in bytes.

Client

A client is either a computer or software that receives information or services from the server. In a client-server module, clients requests for services from servers. The best example is a web browser such as Google Chrome, Firefox, etc. These web browsers request web servers for the required web pages and services as directed by the user. Other examples include online games, online chats, etc.

gethostname is used when client and server are on on the same computer. (LAN – localip / WAN – publicip)

Here, the client wants to receive some information from the server and for this, you need to use the recv() method and the information is stored in another variable msg. Just keep in mind that the information being passed will be in bytes and in the client in the above program can receive up to 1024 bytes (buffer size) in a single transfer. It can be specified to any amount depending on the amount of information being transferred.

Finally, the message being transferred should be decoded and printed.

Code (Server Side):

```
import socket

# next create a socket object
s = socket.socket()
print ("Socket successfully created")

# reserve a port on your computer in our
# case it is 999 but it can be anything
port = 999

# Next bind to the port
s.bind(('', port))
print ("socket binded to %s" %(port))

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

```
# a forever loop until we interrupt it or
# an error occurs
while True:

# Establish connection with client.
    c, addr = s.accept()
    print ('Got connection from', addr )

    # send a thank you message to the client. encoding to send byte
type.
    c.send('Thank you for connecting'.encode())

    # Close the connection with the client
    c.close()

    # Breaking once connection closed
    break
```

Code (Client Side):

```
# Import socket module
import socket

# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
port = 999

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server and decoding to get the string.
print (s.recv(1024).decode())
# close the connection
s.close()
```

Output (Server Side)

```
> py server.py
Socket successfully created
socket binded to 999
socket is listening
Got connection from ('127.0.0.1', 60394)
```

Output (client side)

```
> py client.py
Thank you for connecting
```

Conclusion: There are multiple built in functions in python on socket programming which can faster the speed and reduce the complexity of coding. Hence complex Socket programming can also be done using python easily.