Lab Report No: 03

Lab Report Name: Introduction to Socket programming .

Name : Binodon

ID : IT-17046

Theory: Socket programming shows how to use socket APIs to establish communication links between remote and local processes. The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications.

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs. i5/OS™ sockets support multiple transport and networking protocols. Socket system functions and the socket network functions are threadsafe.

Server side : Server-side network programming involves designing and implementing programs to be run on a server. Server-side applications run as processes on a dedicated physical machine, virtual machine, or cloud infrastructure. Server-side applications receive requests from the clients and perform tasks as requested by the clients.

Server side code:

```
import socket # for 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:
    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 \

on port == %s" %(host_ip)
```

Output:

Client side: In a client environment, each computer still holds (or can still hold) its (or some) resources and files. Other computers can also access the resources stored in a computer, as in a peer-to-peer scenario. One of the particularities of a client/server network is that the files and resources are centralized. This means that a computer, the server, can hold them and other computers can access them. Since the server is always ON, the client machines can access the files and resources without caring whether a certain computer is ON.

Client side code:

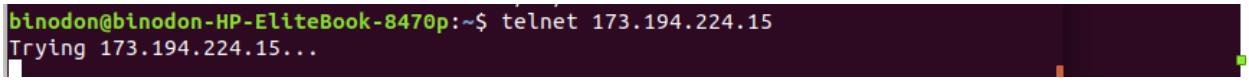
```
# standard Python
```

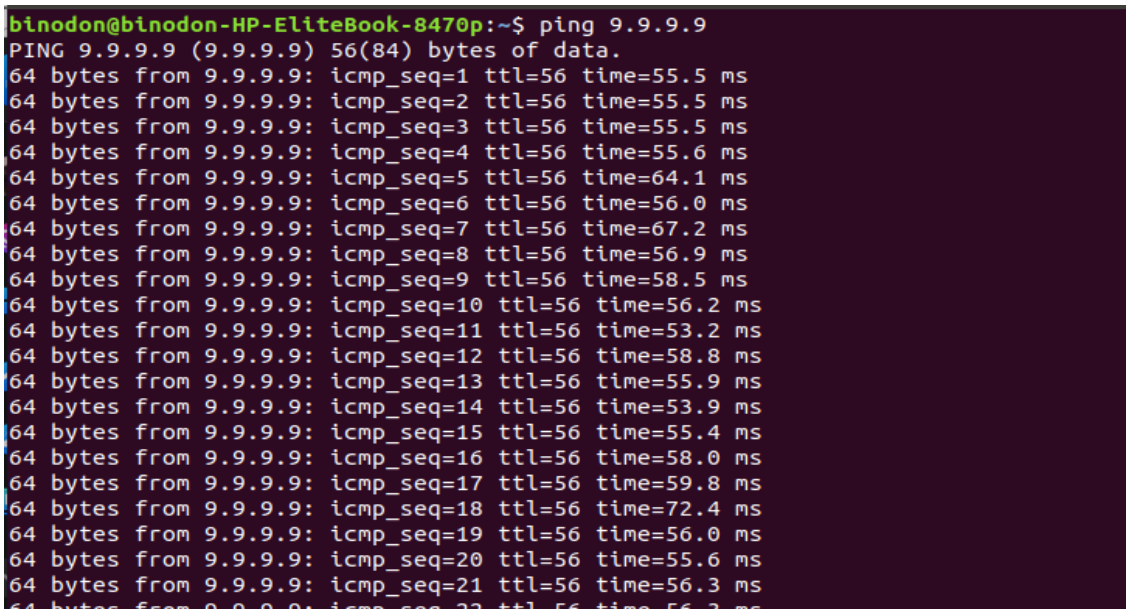
```
sio = socketio.Client()


# asyncio
sio = socketio.AsyncClient()
sio.connect('http://localhost:127.0.0.1')
await sio.connect('http://localhost:127.0.0.1')
sio.event(namespace='/chat')
def my_custom_event(sid, data):
    pass


@sio.on('connect', namespace='/chat')
def on_connect():
```

Output : Socket successfully created the socket has successfully connected to google on port == [173.194.224.15](173.194.224.15)





Conclusion: Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

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.