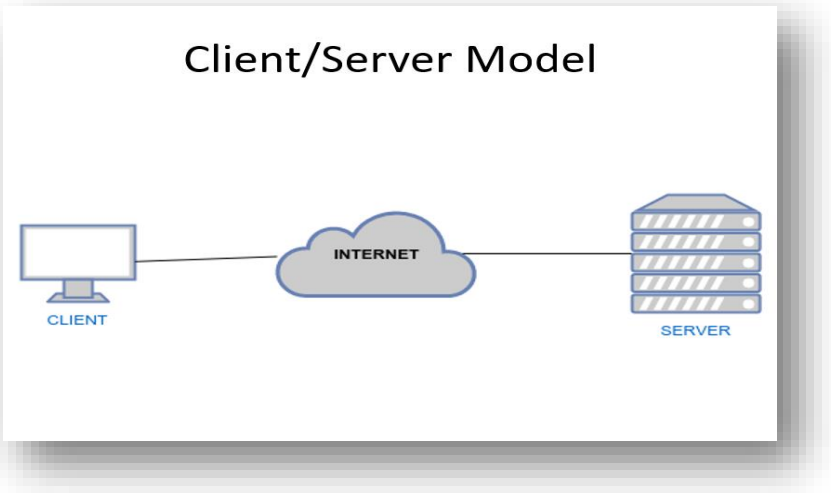
LAB # 3

Socket Programming

Introduction:

"Socket programming is a way of connecting two nodes on a network to communicate with each other."

A *socket* is one endpoint of a two-way communication link between two programs running on the network. Socket acts as an interface between application and network. It supports basic network communications on an IP network It is mostly used to **create a client-server environment**.



Network Socket is used to identify particular processes (programs) on particular machines.It is composed of 2 numbers:

- IP address – machine identifier
- Port number – process identifier

There are two socket types for two transport services:

- **UDP:**
  SOCK_DGRAM is a datagram-based protocol, that involves NO connection.
  It is an unreliable service.
- **TCP:**
  SOCK_STREAM is a "reliable" or "confirmed" service. It is based on TCP, data transmission is more secure.

Goal of Socket Programming:
Learn how to build client/server applications that communicate using sockets.

How Does It Work?

- The application creates a socket.
- The socket type dictates the style of communication.
- Once configured, the application can pass data to the socket for network transmission and receive data from the socket (transmitted through the network by some other host).

"Passive" Listening Socket VS Active "Connected" Socket:

Server:

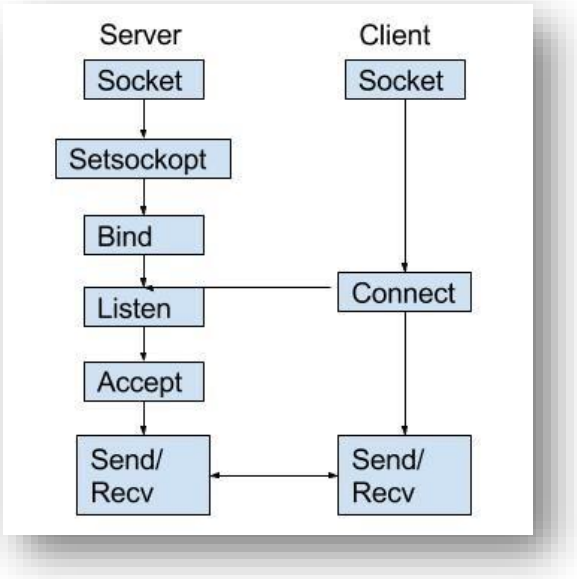
The server is a machine that is ready to receive connections. It holds the address and the port number.

Passive Listening Socket:

It DOES NOT represent any actual network conversation. No data can be received / sent by this kind of port! (Q: HOW it works then ???)

When the server get the request from the passive listening socket, server checks with the OS willingness to receive incoming connections. If NO, drop the request. If YES, an active connected socket is then created .This socket is bounded to one remote conversation partner

SOCKET PROGRAMMING USING PYTHON



Socket() : To create a socket.

Setsockopt() : This function provides an application program with the means to control socket behavior.

Bind() : This method binds the address (hostname, port number) to the socket.

Listen() : Indicates a readiness to accept client connection requests

Accept() : Blocks and waits for an incoming connection.

Connect() : Connects client socket to a TCP based server socket.

SIMPLE SERVER AND CLIENT CONNECTION

SERVER.py CLIENT.py

```python
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((socket.gethostname(), 1027))
s.listen(5)
print("lISTENING...")
while True:
    clt, adr=s.accept()
    print(f"Connection to {adr} established...")
    clt.send(bytes("Socket programming in python", "utf-8"))
    clt.close()
```

SERVER.py CLIENT.py

```python
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((socket.gethostname(),1027))
complete_info= ' '
while True:
    msg=s.recv(7)
    if len(msg)<=0:
        break
    complete_info+=msg.decode("utf-8")
print(complete_info)
```

AF_INET is an address family that is used to designate the type of addresses that your socket can communicate with.

SOCK_STREAM means that it is **a TCP socket (**SOCK_DGRAM for UDP**).**

Let's try another simple task for TCP server and client.

TCPserver.py TCPclient.py

```python
import socket,sys
s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
HOST='127.0.0.1'
PORT=8888

def recv_all(sock, length):
    data=''
    while len(data)<length:
        more=sock.recv(length - len(data))
        if not more:
            print("......end")
            raise EOFError('socket closed %d bytes into a %d-byte message'
                           %(len(data), length))
        data+=more.decode("utf-8")
    return data

s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
s.bind((HOST,PORT))
s.listen(1)
while True:
    print('Listening at',s.getsockname())
    sc, sockname=s.accept()
    print('We have accepted a connection from ',sockname)
    print('Socket connects',sc.getsockname(), 'and', sc.getpeername())
    message = recv_all(sc,16)
    print('The incoming 16-octect message says',repr(message))
    sc.sendall('BYE BYE CLIENT...'.encode("utf-8"))
    sc.close()
    print('Reply sent, socket closed')
```

```
Listening at ('127.0.0.1', 8888)
We have accepted a connection from  ('127.0.0.1', 50454)
Socket connects ('127.0.0.1', 8888) and ('127.0.0.1', 50454)
The incoming 16-octect message says 'Hello!! Server!!'
Reply sent, socket closed
Listening at ('127.0.0.1', 8888)
```

TCPserver.py | TCPclient.py

```python
import socket

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

HOST= '127.0.0.1'
PORT = 8888

def recv_all(sock,length):
    data=''
    while len(data)<length:
        more= sock.recv(length-len(data))
        if not more:
            raise EOFError('socket closed %d bytes into a %d-byte'
                           'message' %(len(data),length))
        data+=more.decode("utf-8")
    return data

s.connect((HOST, PORT))
print('Client has been assigned socket name',s.getsockname())
msg='Hello!! Server!!'
s.sendall(msg.encode('utf-8'))
reply = recv_all(s,16)
print('THE SERVER SAID', repr(reply))
s.close()
```

```
Client has been assigned socket name ('127.0.0.1', 50454)
THE SERVER SAID 'BYE BYE CLIENT..'

Process finished with exit code 0
```

TASK:

Above program comes with a limitation that the message must be fixed in 16 characters (octets)!

Your task is to modify the above program

- Determine the length of message L before sending it (from client side). [Assume max. number of length is 255 i.e. 3 digits], pad with leading zeros using zfill() method.
- Add L at the beginning of message.
- Send that message to server.
- Server should extract the length of message (first 3 characters) and read the rest of message with proper length.
