Name :- Rushikesh Kazbhazi Palve
Roll No. 31258

Date : / /20 | Page No :

Assignment No. 7

DOS :- 31-10-2021

Problem statement :-

Write a program using TCP socket for wired network for following
a.) Say Hello to Each other
b.) File transfer
c.) Calculator.

Objectives :-

(i) To learn TCP socket.
(ii) To implement a program using TCP socket for data transmission & communication.

Outcomes :-

After completion of the Assignment, students will be able to -
(i) Understand TCP protocol & socket.
(ii) Write a program using TCP socket for wired network.

Software & Hardware Requirements :-

Softwares : C/C++ compiler.

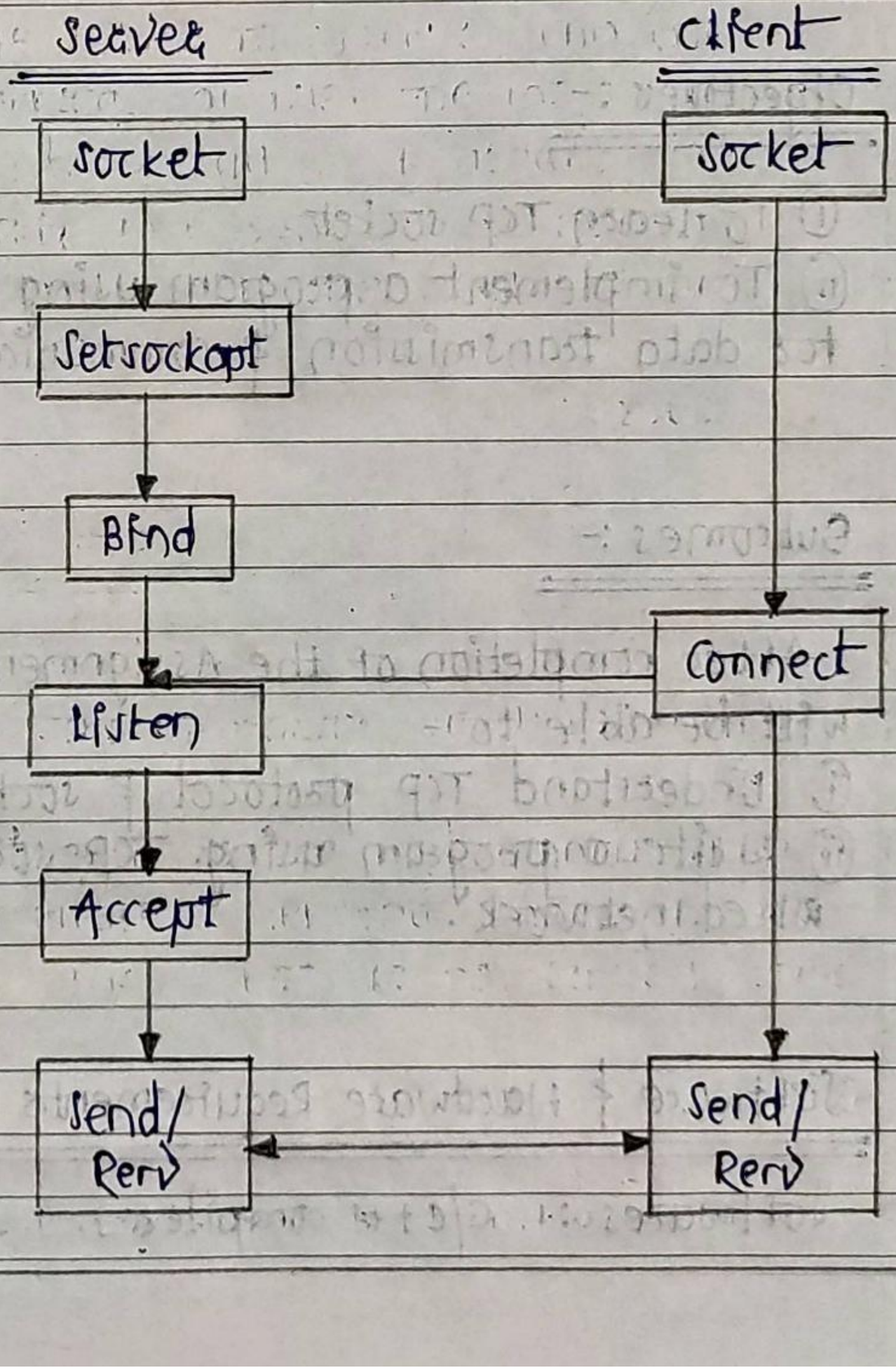
ROLL NO. 31258

: Date :	Page No :
/ /20	

THEORY :-

Socket Programming :-

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



Stages for server :-

- **Socket creation :**

int sockfd = socket(domain, type, protocol)

sockfd : socket descriptor, an integer (like a file-handle)

domain : integer, communication domain e.g. AF_INET (IPv4 protocol), AF_INET6 (IPv6 protocol)

type : communication type

SOCK_STREAM : TCP (reliable, connection oriented)

SOCK_DGRAM : UDP (unreliable, connectionless)

protocol : protocol value for Internet protocol (IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.

- **Setsockopt :**

int setsockopt (int sockfd, int level, int optname, const void *optval, socklen_t optlen);

this helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents errors such as : "address already in use.".

- **Bind :**

int bind (int sockfd, const struct sockaddr *addr, socklen_t addrlen);

After creation of the socket, bind function binds the socket to the address and port number specified in addr (custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

- **Listen :**

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

- **Accept :**

```
int new_socket = accept(int sockfd, struct
    sockaddr *addr, socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client :-

- **Socket connection :** Exactly same as that of server's socket creation.

- **Connect :**

```
int connect (int sockfd, const struct sockaddr
             *addr, socklen_t addrlen);
```

The connect() system call connects the socket referred to by the file descriptor ~~of~~ sockfd to the address specified by addr. Server's address and port is specified in addr.

TCP Protocol :-

The Transmission Control protocol (TCP) is a connection-oriented reliable protocol. It provides a reliable transport service between pairs of processes executing on End systems (ES) using the network layer service provided by the IP protocol.

TCP is stream oriented, that is, TCP protocol entities exchange streams of data. Individual bytes of data (e.g. from an application or session layer protocol) are placed in memory buffers and transmitted by TCP in transport protocol Data Units (for TCP these are usually known as "segments"). The reliable, flow-controlled TCP service is much more complex than UDP, which only provides a best

Effect service. To implement the service, TCP uses a number of protocol fimers that ensure reliable and synchronised communication between the two End systems:

ALGORITHM :-

- **Server :-**

1. Start
2. Create Server side socket using int sockfd = socket(domain, type, protocol)
3. Define parameters for Server socket.
4. Bind server socket using bind() method.
5. perform listen method using listen() method.
6. Accept data from client using accept() method.
7. close Server connection.
8. Stop.

- **Client :-**

1. Start
2. Create client side socket using int sockfd = socket(domain, type, protocol)
3. Define parameters for client socket.
4. Connect client to Server socket using connect() method.
5. Accept/send data from client to server using send/recv() method.
6. Stop.

CONCLUSION :-

Thus we have implemented a program that create socket in between client & server for data transmission.

CODE :-

Server.py

```
import socket
import sys

host=None
port=None
s=None
def create_socket():
    global host
    global port
    global s
    host=""
    port=9999
    try:
        s=socket.socket()
    except socket.error as e:
        pass

def bind_socket():
    global host
    global port
    global s
    try:
        s.bind((host,port))
        print("Listening...",end="")
        s.listen(5)
    except socket.error as e:
        bind_socket()

def socket_accept():
    global host
    global port
    global s
    try:
        connection,address=s.accept()
        print(f"connection address: {address}")
        send_commands(connection)
        if connection:
            connection.close()
    except socket.error as e:
        pass

def send_commands(connection):
    while True:
        cmd= input("What to solve?\t=>")
        if cmd=="q":
            connection.send(str.encode(cmd))
            connection.close()
            s.close()
            sys.exit()
        if len(str.encode(cmd)):
            connection.send(str.encode(cmd))
            response=str(connection.recv(1024),"utf-8")
```

```
            print("Solution: ",response)

def main():
    create_socket()
    bind_socket()
    socket_accept()

main()
```

Client.py

```
import socket,math

host="100.80.5.251"
port=9999
s=socket.socket()

s.connect((host,port))
print("Client ready to solve...")
while True:
    data=str(s.recv(1024),'utf-8')
    if data=='q':
        s.close()
        break
    print("Solving: ",data)
    try:
        output=eval(data)
    except:
        output="Not a proper calculation"
    s.send(str.encode(str(output),'utf-8'))
    print("Sent result...")
```

OUTPUT :-

Server

```
Listening...connection address: ('100.80.5.251', 60985)
What to solve?  =>5*5+5/5
Solution:  26.0
What to solve?  =>math.pi*2
Solution:  6.283185307179586
What to solve?  =>(10+6)*5
Solution:  80
What to solve?  =>math.sin(50)
Solution:  -0.2623748537039287 7
What to solve?  =>math.cos(50)
Solution:  0.964966028492113 3
What to solve?  =>math.tan(50)
Solution:  -0.2719006119976307 7
What to solve?  =>math.sin(50)/math.cos(50)
Solution:  -0.271900611997630 7
What to solve?  =>math.abs(-2.67)
Solution:  Not a proper calculation
What to solve?  =>abs(-2.67)
```

```
Solution:  2.67
What to solve?  =>2.69%1.5
Solution:  1.19
What to solve?  =>q
```

Client

```
Client ready to solve...
Solving:  5*5+5/5
Sent result...
Solving:  math.pi*2
Sent result...
Solving:  (10+6)*5
Sent result...
Solving:  math.sin(50)
Sent result...
Solving:  math.cos(50)
Sent result...
Solving:  math.tan(50)
Sent result...
Solving:  math.sin(50)/math.cos(50)
Sent result...
Solving:  math.abs(-2.67)
Sent result...
Solving:  abs(-2.67)
Sent result...
Solving:  2.69%1.5
Sent result...
```