Date - 07/05/2021

Experiment No. 5

Client Server using TCP and Socket Programming

Aim:

To write a program to implement Client-Server communication using Socket Programming and TCP as transport layer protocol

Theory:

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. They are the real backbones behind web browsing. In simpler terms there is a server and a client.

Socket programming is started by importing the socket library and making a simple socket.

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

Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address family ipv4. The SOCK_STREAM means connection oriented TCP protocol.

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

Some of the key functions from socket - Low-level networking interface:

1. **socket.socket():** Create a new socket using the given address family, socket type and protocol number.
2. **socket.bind(address):** Bind the socket to address.
3. **socket.listen(backlog):** Listen for connections made to the socket. The backlog argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.
4. **socket.accept():** The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.
   At accept(), a new socket is created that is distinct from the named socket. This new socket is used solely for communication with this particular client.
   For TCP servers, the socket object used to receive connections is not the same socket used to perform subsequent communication with the client. In particular, the accept() system call returns a new socket object that's actually used for the connection. This allows a server to manage connections from a large number of clients simultaneously.
5. **socket.send(bytes[, flags]):** Send data to the socket. The socket must be connected to a remote socket. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.
6. **socket.close():** Mark the socket closed. all future operations on the socket object will fail. The remote end will receive no more data (after queued data is flushed). Sockets are automatically closed when they are garbage-collected, but it is recommended to close() them explicitly.

Algorithm:

1. Client

```
Step 1. Start
Step 2. Connect to localhost: port
Step 3. Receive data
Step 4. Close connection
Step 5. Stop
```

2. Server

```
Step 1. Start
Step 2. Create a socket
Step 3. Bind it to an address and a port
Step 4. Wait for the client to connect
Step 5. If client:
        Initialize the connection
Stelt 6. Else:
        Connection timeout, goto step 10
Step 7. Transfer Bytes
Step 8. Stop
```

Program:

a. server.py

```
import socket
import time

# create a socket object
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9999

# bind to the port
serversocket.bind((host, port))

# queue up to 5 requests
serversocket.listen(5)

while True:
    # establish a connection
    clientsocket,addr = serversocket.accept()

    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n"
    clientsocket.send(currentTime.encode('ascii'))
    clientsocket.close()
```

b. client.py

```
import socket

# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9999
```

```
# connection to hostname on the port.
s.connect((host, port))

# Receive no more than 1024 bytes
tm = s.recv(1024)

s.close()

print("The time got from the server is %s" % tm.decode('ascii'))
```

Output:

```
$ python server.py &
Got a connection from ('127.0.0.1', 54597)

$ python client.py
The time got from the server is Fri May 07 20:36:52 2021
```

Result:

The program to implement Client-Server communication using Socket Programming and TCP as transport layer protocol has been executed successfully.