Computer Networks Lab 02

Course: Computer Networks (CL3001) **Semester:** Spring 2024
Instructor: Muhammad Nadeem Ghouri **Department:** SE

Note:

- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Socket Programming

Sockets and the socket API are used to send messages across a network. They provide a form of inter process communication (IPC). The network can be a logical, local network to the computer, or one that's physically connected to an external network, with its own connections to other networks. The obvious example is the Internet, which you connect to via your ISP.
Sockets have a long history. Their use originated with ARPANET in 1971 and later became an API in the Berkeley Software Distribution (BSD) operating system released in 1983 called Berkeley sockets.

When the Internet took off in the 1990s with the World Wide Web, so did network programming.
Web servers and browsers weren't the only applications taking advantage of newly connected networks and using sockets. Client-server applications of all types and sizes came into widespread use.

Today, although the underlying protocols used by the socket API have evolved over the years, and new ones have developed, the low-level API has remained the same.

The most common type of socket applications are client-server applications, where one side acts as the server and waits for connections from clients. This is the type of application that you'll be creating in this tutorial.
More specifically, you'll focus on the socket API for Internet sockets, sometimes called Berkeley or BSD sockets. There are also Unix domain sockets, which can only be used to communicate between processes on the same host.

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.

- **TCP:** TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

- **UDP:** UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

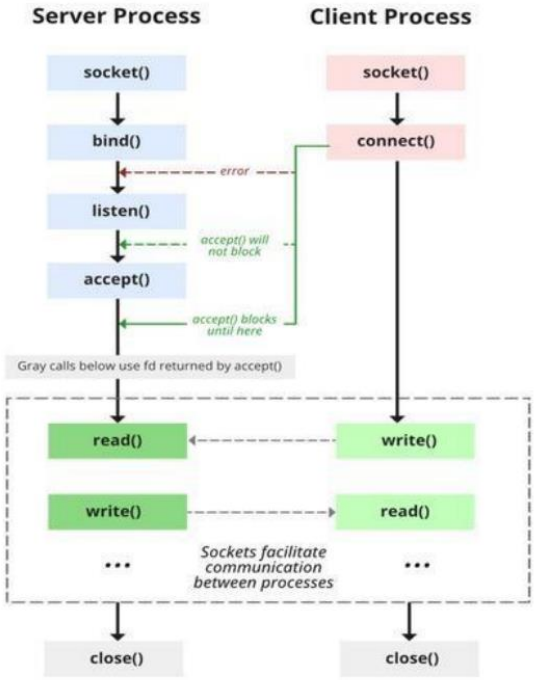

Fig-1: State diagram for server & client model of socket

Socket Programming in Python

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.

The following steps occur when establishing a TCP connection between two computers using sockets:

1. The server instantiates a Server Socket object, denoting which port number communication is to occur on.
2. The server invokes the accept() method of the Server Socket class. This method waits until a client connects to the server on the given port.
3. After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
4. The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an Output Stream and an Input Stream. The client's Output Stream is connected to the server's Input Stream, and the client's Input Stream is connected to the server's Output Stream. TCP is a two-way communication protocol, so data can be sent across both streams at the same time. There are following useful classes providing complete set of methods to implement sockets.

Important terms in socket vocabulary are as follow:

- **Domain:** The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
- **Type:** The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
- **Protocol:** Typically, zero, this may be used to identify a variant of a protocol within a domain and type.
- **Hostname:** The identifier of a network interface –
  - A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation.
  - A string "<broadcast>", which specifies an INADDR_BROADCAST address.
  - A zero-length string, which specifies INADDR_ANY, or
  - An Integer, interpreted as a binary address in host byte order.

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax:

s = socket.socket (socket_family, socket_type, protocol=0)

Here is the description of the parameters:

- **socket_family:** This is either AF_UNIX or AF_INET.
- **socket_type:** This is either SOCK_STREAM or SOCK_DGRAM.
- **protocol:** This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required:

Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to Socket.
s.listen()	This method sets up and start TCP listener.
s.accept	This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message.
s.send()	This method transmits TCP message.
s.recvfrom()	This method receives UDP message.
s.sendto()	This method transmits UDP message.
s.close()	This method closes the socket.
socket.gethostname()	Returns the hostname.

Examples

(I have used jupyter notebook. You can use any other IDE of your choice)

1. Getting IP address in python:

```
#importing the socket module:
import socket

#getting the hostname by socket.gethostname() method:
hostname=socket.gethostname()

#getting the ip address by socket.gethostbyname() method:
ip_address=socket.gethostbyname(hostname)

#printing the details:
print(f"Hostname: {hostname}")
print(f"IP_ADDRESS: {ip_address}")
```

```
Hostname: DESKTOP-7PC2P25
IP_ADDRESS: 172.18.224.1
```

2. Getting any host IP address in python:

```
#importing the socket module:
import socket

hostnames=["www.google.com","www.facebook.com"]

#getting the ip address by socket.gethostbyname() method:
#printing the details:
for i in hostnames:
    ip_address=socket.gethostbyname(i)
    print(f"Hostname: {i}")
    print(f"IP_ADDRESS: {ip_address}\n")
```

```
Hostname: www.google.com
IP_ADDRESS: 172.217.169.228

Hostname: www.facebook.com
IP_ADDRESS: 157.240.227.35
```

3. Getting hostname by IP address in python:

```
#importing the socket module:
import socket

#getting the hostname by using the ip address:
hname=socket.gethostbyaddr('172.16.5.43')
hname1=socket.gethostbyaddr('8.8.8.8')

#printing the details:
print(f"IP_ADDRESS: {hname}")
print(f"IP_ADDRESS: {hname1}")
```

```
IP_ADDRESS: ('Exam-Server.khifast.nu.edu.pk', [], ['172.16.5.43'])
IP_ADDRESS: ('dns.google', [], ['8.8.8.8'])
```

4. Getting service name, given port number & protocol in python:

```
#importing the socket module:
import socket

def find_service_name():
    protocol_name='tcp'
    for port in [80,25]:
        print("Port %s => Service name: %s" %(port, socket.getservbyport(port, protocol_name)))

    print("Port %s => Service name: %s" %(53, socket.getservbyport(53, 'udp')))

if __name__=='__main__':
    find_service_name()
```

```
Port 80 => Service name: http
Port 25 => Service name: smtp
Port 53 => Service name: domain
```

5. Port Scanner (This might take a few minutes to complete execution):

```
#importing the socket module:
from socket import *

#importing time module
import time

startTime=time.time()

if __name__=='__main__':
    target=input("Enter the hostname to be scanned: ")
    target_ip=gethostbyname(target)
    print("Starting scan on host: ",target_ip)

    for i in range(50,500):
        s=socket(AF_INET, SOCK_STREAM)

        conn=s.connect_ex((target_ip,i))

        if(conn==0):
            print("Port %d: OPEN" % (i,))
        s.close()

print("Time Taken in Seconds: ",time.time()-startTime)
print("Time in Minutes: ",(time.time()-startTime)/60)
```

```
Enter the hostname to be scanned: localhost
Starting scan on host:  127.0.0.1
Port 135: OPEN
Port 445: OPEN
Time Taken in Seconds:  914.0224618911743
Time in Minutes:  15.233707698186238
```

6. Simple Client Server Connecting Program (Execution takes a few minutes):

Server:

```
#importing the socket module:
import socket

#creating a socket object
s=socket.socket()
print("Socket Created")

#binding to the port
s.bind(('localhost',9999))

#now wait for client connection
s.listen(5)
print("Waiting for connection")

while True:
    c, addr=s.accept()     #establish connection with the client
    print("Got connection from",addr)
    c.send(bytes("Thankyou for connecting"))

    c.close()              #close the connection
```

Client:

```
#importing the socket module:
import socket

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

s.connect(('localhost',9999))
print(s.recv(1024).decode())

s.close()
```

Tasks

Design a simple Client Server chat application using the above concepts.