# Computer Networks Lab

## Lab-V

## Socket Programming

# Client-Server Model

## Introduction

- Standard model for developing network applications
- Notation of Client and Server
  - Server is a process that is offering some services
  - Client is a process that is requesting a service
  - Server or Client may be running on different machine
  - Server waits for requests from Client(s)

# Client-Server Model

## Typical scenario

- The server process starts on some computer system
- Server initialize itself , then goes to sleep waiting for a client request
- A client process starts , either on the same system or on some other system
- It sends a request to the server
- When the server process has finished providing its service to the client , the server goes back to sleep , waiting for the next client request to arrive

# Iterative Server

- It is used when server process knows in advance how long it takes to handle each request and it handles each request itself.

  - ✓ Single copy of server runs at all time
  - ✓ A client may have to wait if server is busy

# Concurrent Server

- It is used when amount of work required to handle a request is unknown ; the server starts another process to handle each request.

  - ✓ A copy of the server process to a client's request in a dedicated fashion
  - ✓ As many copies of server as there are clients requests

# What is Socket?

- The socket is the BSD(Berkeley sockets)method for achieving inter-process communication
- An interface between an application process and transport layer
- Used to allow one process to speak to another(on same or different machine)
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent

# What is Socket?

- **Socket is 3 tuple**
  - ✓ Protocol, local IP address, local Port number
  - ✓ Protocol, remote IP address, remote Port number

## Types of Socket Programming

1. Socket programming with TCP
2. Socket programming with UDP

# Transmission Control Protocol (TCP)

- Is a connection-oriented
- A client must connect a socket to a server
- TCP socket provides bidirectional channel between client and server
- Lost data is re-transmitted
- Data is delivered in-order
- Data is delivered as a stream of bytes
- TCP uses flow control

# User Datagram Protocol (UDP)

- Is a connectionless.
- A single socket can send and receive packets from many different computers.
- Best effort delivery.
- Some packets may be lost some packets may arrive out of order.

# Socket Programming using Python

- Sockets may be implemented over a number of different channel types: 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.

# Socket Programming using Python

## Sockets Vocabulary

| Sr.No. | Term & Description |
|---|---|
| 1 | 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. |
| 2 | Type: The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols. |
| 3 | Protocol: Typically zero, this may be used to identify a variant of a protocol within a domain and type. |
| 4 | Hostname: The identifier of a network interface:<br>• A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation<br>• A string "<broadcast>", which specifies an INADDR_BROADCAST address.<br>• A zero-length string, which specifies INADDR_ANY, or<br>• An Integer, interpreted as a binary address in host byte order. |
| 5 | Port: Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service. |

# Socket Programming using Python

## The *socket* Module

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

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

- Description of the parameters:
  - **socket_family:** This is either AF_UNIX or AF_INET, as explained earlier.
  - **socket_type:** This is either SOCK_STREAM or SOCK_DGRAM.
  - **Protocol:** This is usually left out, defaulting to 0.
- Once we have *socket* object, then we can use required functions to create the client or server program.

# Socket Programming using Python

## Server Socket Methods

| Sr.No. | Method & Description |
|---|---|
| 1 | s.bind()<br>This method binds address (hostname, port number pair) to socket. |
| 2 | s.listen()<br>This method sets up and start TCP listener. |
| 3 | s.accept()<br>This passively accept TCP client connection, waiting until connection arrives (blocking). |

# Socket Programming using Python

## Client Socket Methods

| Sr.No. | Method & Description |
|---|---|
| 1 | s.connect()<br><br>This method actively initiates TCP server connection. |

# Socket Programming using Python

## General Socket Methods

| Sr.No. | Method & Description |
|---|---|
| 1 | s.recv(): This method receives TCP message |
| 2 | s.send(): This method transmits TCP message |
| 3 | s.recvfrom(): This method receives UDP message |
| 4 | s.sendto(): This method transmits UDP message |
| 5 | s.close(): This method closes socket |
| 6 | socket.gethostname(): Returns the hostname. |

# Socket Programming using Python

## Python Internet Modules

| Protocol | Common function | Port No | Python module |
|---|---|---|---|
| HTTP | Web pages | 80 | httplib, urllib, xmlrpclib |
| NNTP | Usenet news | 119 | nntplib |
| FTP | File transfers | 20 | ftplib, urllib |
| SMTP | Sending email | 25 | smtplib |
| POP3 | Fetching email | 110 | poplib |
| IMAP4 | Fetching email | 143 | imaplib |
| Telnet | Command lines | 23 | telnetlib |
| Gopher | Document transfers | 70 | gopherlib, urllib |

# A Simple Server-Client Program

# Code for Server

```
# Import the socket library
import socket

# Next create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print ("Socket is created successfully")

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

# Reserve a port on the computer in this case it is 23456
# but it can be anything
port = 23456

# Next bind to the port
s.bind((host, port))
print ("Socket is binded to %s" %(port))
```

# Code for Server Contd.

```
# Put the socket into listening mode
s.listen(5)
print ("Socket is listening")

# A forever loop until we interrupt it or an error occurs
while True:
    # Establish connection with client.
    c, addr = s.accept()
    print ('Got connection from', addr )

    # Send a thank you message to the client.
    c.send(b'Thank you for connecting')

    # Close the connection with the client
    c.close()
```

# Code for Client

```
# Import socket module
import socket

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

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

# Define the port on which we want to connect
port = 23456

# connect to the server on local computer
s.connect((host, port))

# Receive data from the server
msg = s.recv(1024)
print (msg)

# Close the connection
s.close()
```

# Assignment-V

**Write socket program in Python to send a message from a client machine to a server machine and a server machine to a client machine using TCP.**