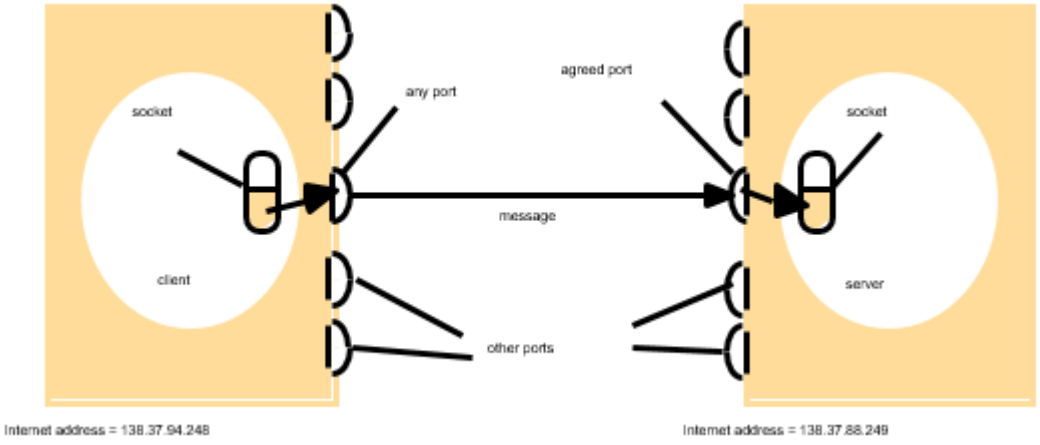
Socket Programming in Python

Socket Basics:

A *network socket* is an endpoint of an inter-process communication flow across a computer network. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Today, most communication between computers is based on the internet protocol; therefore most network sockets are *internet sockets*. To create a connection between machines, Python programs import the **socket** module, create a socket object, and call the object's methods to establish connections and send and receive data. Sockets are the endpoints of a bidirectional communications channel.



Socket 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, SMTP, and so on. 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.

Vocabulary of Sockets

Term	Description
domain	The family of protocols that will be 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.
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.

The *socket* Module:

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, as explained earlier.
- **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.

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).

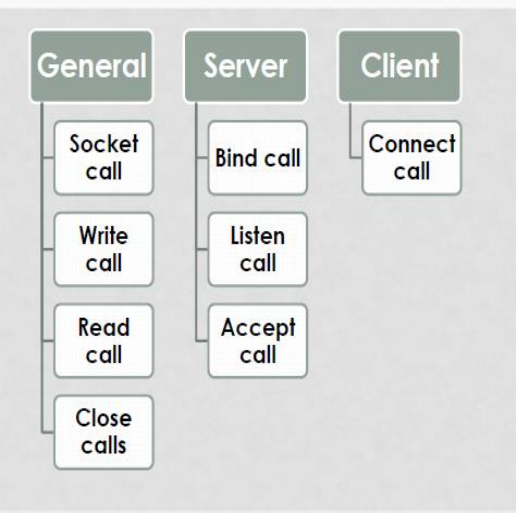
Client Socket Methods

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

General Socket Methods

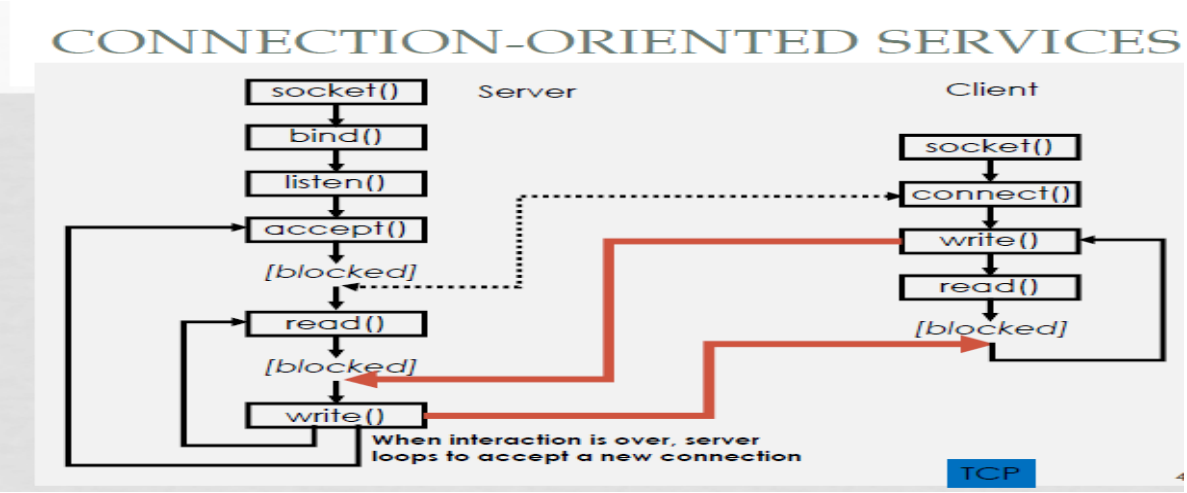
Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

MAJOR SYSTEM CALLS



MAJOR SYSTEM CALLS (SUMMARY)

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection



A Simple Server:

To write Internet servers, we use the **socket** function available in socket module to create a socket object.

A socket object is then used to call other functions to setup a socket server. Now call **bind(hostname, port)** function to specify a *port* for your service on the given host. Next, call the *accept* method of the returned object. This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

```
# Echo server program
import socket
host = socket.gethostname()
port = 12345
s = socket.socket()
s.bind((host, port))
s.listen(5)
conn, addr = s.accept()
print('Got connection from ', addr[0], '(', addr[1], ')')
print('Thank you for connecting')
while True:
    data = conn.recv(1024)
    if not data: break
    conn.sendall(data)
conn.close()
```

A Simple Client:

Now we will write a very simple client program which will open a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's *socket* module function.

The **socket.connect(hosname, port)** opens a TCP connection to *hostname* on the *port*. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits:

```
# Echo client program
import socket
host = socket.gethostname()
port = 12345
s = socket.socket()
s.connect((host, port))
s.sendall(b'Welcome User!')
data = s.recv(1024)
s.close()
print(repr(data))
```

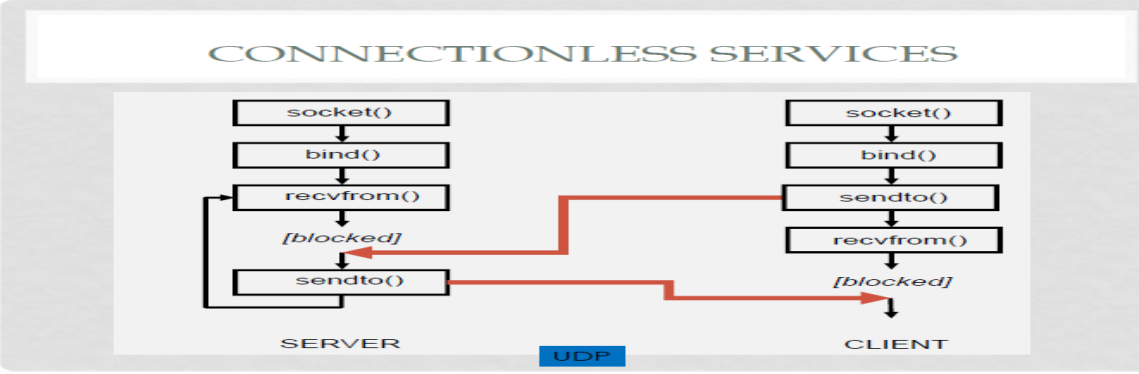
Note: The repr() function returns a printable representational string of the given object.

Now run this ***server.py*** in background and then run above ***client.py*** to see the result.

Output:

Step 1: Run ***server.py***. It would start a server in background.

Step 2: Run ***client.py***. Once server is started run client.



Simple Connectionless Server

import socket

```
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)      # For UDP

udp_host = socket.gethostname()                    # Host IP
udp_port = 12345                                   # specified port to connect

sock.bind((udp_host,  udp_port))

while True:
        print "Waiting for client..."
        data,addr = sock.recvfrom(1024)          #receive data from client
        print "Received Messages:",data.decode()," from",addr
```

Simple Connectionless Client:

```
import socket

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)      # For UDP

udp_host = socket.gethostname()                  # Host IP
udp_port = 12345                                 # specified port to connect

msg = "UDP Program!"
print "UDP target IP:", udp_host
print "UDP target Port:", udp_port

sock.sendto(msg.encode(),(udp_host,udp_port))
```

Practice Programs:

1A) Write a program where client can send a message to the server and the server can receive the message and send, or echo, it back to the client.

Echo Client:

```
import socket

HOST = '127.0.0.1'  # The server's hostname or IP address
PORT = 2053      # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
    print('Received Connection')
    print('Server:', data.decode())
```

Echo Server:

```
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 2053     # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()

    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if  data:
                print("Client: ",data.decode())
```

```
        data = input("Enter message to client:");
        if not data:
            break;
        # sending message as bytes to client.
        conn.sendall(bytearray(data, 'utf-8'));

    conn.close()
```

2A) Write a program to create TCP time server in Python

Time Client:

```
#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 = 9991
# connection to hostname on the port.
s.connect((host, port))
# Receive no more than 1024 bytes
tm = s.recv(1024)
print(' Current time from Sever :', tm.decode())
s.close()
```

Time Server:

```
# 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 = 9991

# 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()
```

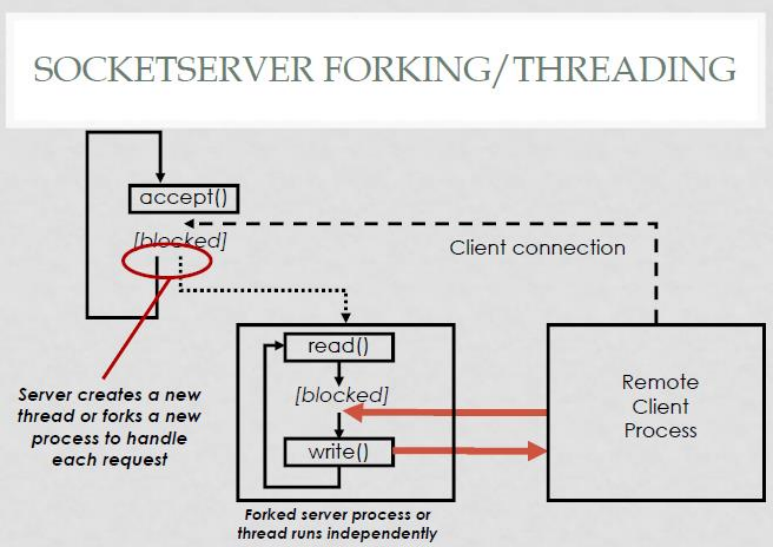
3A) Write a TCP chat server in python using socket programming.

Client Chat:

```
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 31621    # Port to listen on (non-privileged ports are > 1023)
s = socket.socket()
name = input(str("\nEnter your name: "))

print("\nTrying to connect to ", HOST, "(", PORT, ")\n")

s.connect((HOST, PORT))
print("Connected...\n")

s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")
while True:
    message = s.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        s.send(message.encode())
        print("\n")
        break
    s.send(message.encode())
```

Server Chat:

```
# server.py
import socket
HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 31621    # Port to listen on (non-privileged ports are > 1023)

s = socket.socket()

s.bind((HOST, PORT))

s.listen()
print("\nWaiting for incoming connections...\n")
conn, addr = s.accept()
print("Received connection from ", addr[0], "(", addr[1], ")\n")

s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the chat room\nEnter [e] to exit chat room\n")
name = input(str("Enter your name: "))
conn.send(name.encode())

while True:
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        conn.send(message.encode())
        print("\n")
        break
    conn.send(message.encode())
    message = conn.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
```



4A. Forking/ Threading (Concurrent Server)

```
 import socket

ClientSocket = socket.socket()
host = '127.0.0.1'
port = 11596

print('Waiting for connection')
try:
    ClientSocket.connect((host, port))
except socket.error as e:
    print(str(e))

Response = ClientSocket.recv(1024)
while True:
    Input = input('Client Say Something: ')
    ClientSocket.send(str.encode(Input))
```

```
    Response = ClientSocket.recv(1024)
    print('From Server : ' + Response.decode())

ClientSocket.close()
```

4B. Server Program

```
import socket
import os
from _thread import *

ServerSocket = socket.socket()
host = '127.0.0.1'
port = 11596
ThreadCount = 0
try:
    ServerSocket.bind((host, port))
except socket.error as e:
    print(str(e))

print('Waitiing for a Connection..')
ServerSocket.listen(5)

def threaded_client(connection):
    connection.send(str.encode('Welcome to the Server'))
    while True:
        data = connection.recv(2048)
        print('Received from client :' + str(ThreadCount) +data.decode())
        Inputs = input('Server Says: ')
        if not data:
            break
        connection.sendall(Inputs.encode())
    connection.close()
```

```
while True:
    Client, address = ServerSocket.accept()
    print('Connected to: ' + address[0] + ':' + str(address[1]))
    start_new_thread(threaded_client, (Client, ))
    ThreadCount += 1
    print('Thread Number: ' + str(ThreadCount))
ServerSocket.close()
```

Lab Exercise:

1. Write a UDP time server to display the current time and day.
2. Write a UDP simple chat program for message send and receive.

Sample Output:

Client Side

Do Ctrl+c to exit the program !!
Type some text to send =>Hi
1. Client Sent : Hi
2. Client received : Hello

Server Side:

Do Ctrl+c to exit the program !!
####### Server is listening #######
2. Server received: Hi
Type some text to send => Hello
1. Server sent : Hello
####### Server is listening #######

3. Write a TCP/UDP peer to peer chat system between two different machines.
