

NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCE

Computer Network Lab (CL3001) Lab Session 05

Objective:

- Introduction to Socket Programming
- Introduction to basic methods of socket programming in python
- Implementation of client server communication using python

Socket Programming

1. Introduction:

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.

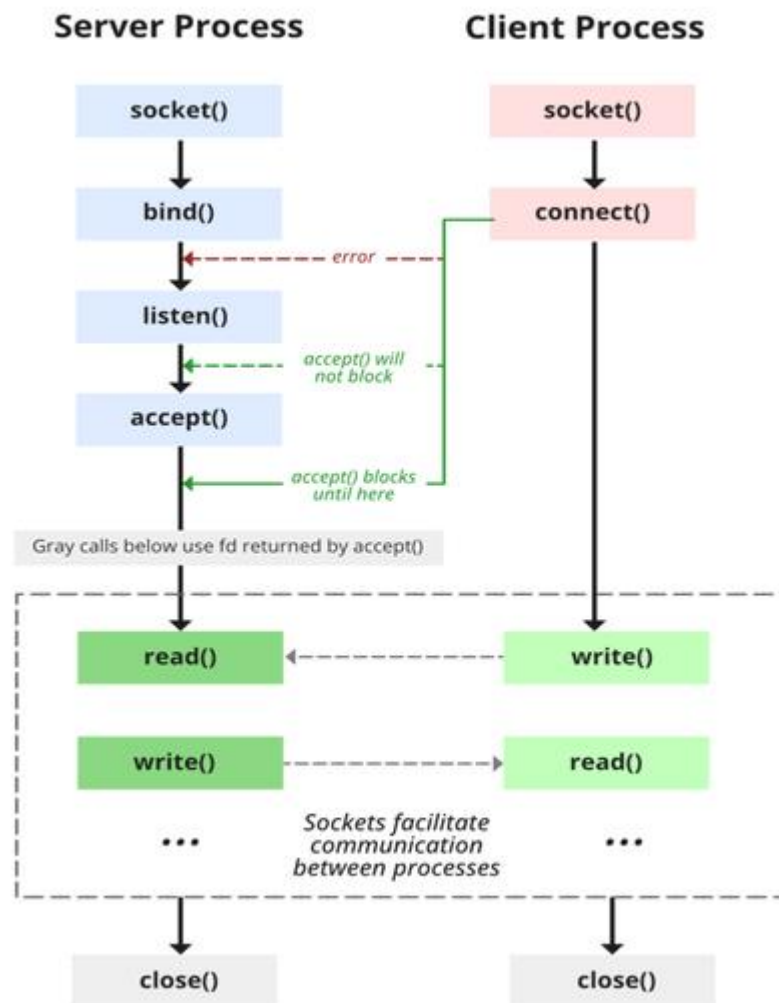


Fig-1: State diagram for server & client model of socket

2. 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 `ServerSocket` object, denoting which port number communication is to occur on.
2. The server invokes the `accept()` method of the `ServerSocket` 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).
-------------------	---

Table-1: Server socket methods in python & their description

Client Socket Method:

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

Table-2: Client socket method in python & their description

General Socket Method:

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.

Table-3: General socket methods in python & their description

3. Examples

Example 1: How to get IP address in python

```

1  ## importing socket module
2  import socket
3  ## getting the hostname by socket.gethostname() method
4  hostname = socket.gethostname()
5  ## getting the IP address using socket.gethostbyname() method
6  ip_address = socket.gethostbyname(hostname)
7  ## printing the hostname and ip_address
8  print(f"Hostname: {hostname}")
9  print(f"IP Address: {ip_address}")

```

Output:

```

Hostname: DESKTOP-1LI534L
IP Address: 192.168.1.101

```

Example 2: How to get Host IP address in python

```

1  ## importing socket module
2  import socket
3  ## getting the IP address using socket.gethostbyname() method
4  ip_address = socket.gethostbyname('google.com')
5  ## printing the hostname ip_address
6  print(f"IP Address: {ip_address}")

```

Output:

```

IP Address: 172.217.169.238
PS E:\FAST\Fall-22\CN\Lab> 

```

Example 3: How to get Host name using IP address in python

```

1  ## importing socket module
2  import socket
3  ## getting the Hostname using IP address
4  Hname = socket.gethostbyaddr('8.8.8.8')
5  ## printing the hostname
6  print(f"IP Address: {Hname}")

```

Output:

```

IP Address: ('dns.google', [], ['8.8.8.8'])
PS E:\FAST\Fall-22\CN\Lab> 

```

Example 4: How to find service name, given port number & protocol

```

1  ## importing socket module
2  import socket
3  def find_service_name():
4      protocolname = 'tcp'
5      for port in [80, 25]:
6          print ("Port: %s => service name: %s" %(port, socket.getservbyport(port, protocolname)))
7
8      print ("Port: %s => service name: %s" %(53, socket.getservbyport(53, 'udp')))
9
10 if __name__ == '__main__':
11     find_service_name()

```

Output:

```

Port: 80 => service name: http
Port: 25 => service name: smtp
Port: 53 => service name: domain

```

Example 5: Port scanner

```

1  ## importing socket module
2  from socket import *
3  import time
4  startTime = time.time()
5
6  if __name__ == '__main__':
7      target = input('Enter the host to be scanned: ')
8      t_IP = gethostbyname(target)
9      print ('Starting scan on host: ', t_IP)
10
11     for i in range(50, 500):
12         s = socket(AF_INET, SOCK_STREAM)
13
14         conn = s.connect_ex((t_IP, i))
15         if(conn == 0) :
16             print ('Port %d: OPEN' % (i,))
17         s.close()
18     print('Time taken:', time.time() - startTime)

```

Output:

```
Enter the host to be scanned: localhost
Starting scan on host: 127.0.0.1
Port 135: OPEN
Port 445: OPEN
Time taken: 911.266163110733
PS E:\FAST\Fall-22\CN\Lab> 
```

Example 6: Simple Client Server connecting program

Server:

```
1  import socket          # Import socket module
2
3  s = socket.socket()     # Create a socket object
4  print('Socket Created')
5  s.bind(('localhost', 9999)) # Bind to the port
6
7  s.listen(5)             # Now wait for client connection.
8  print('waititng for connection')
9  while True:
10     c, addr = s.accept() # Establish connection with client.
11     print ('Got connection from', addr)
12     c.send(bytes('Thank you for connecting'))
13     c.close()           # Close the connection
```

Client:

```
1  import socket          # Import socket module
2
3  s = socket.socket()     # Create a socket object
4  s.connect(('localhost', 9999))
5  print(s.recv(1024).decode())
6  s.close()              # Close the socket when done
```

4. Lab Task:

Design a simple Client Server chat application