# Socket Programming

| | |
|---|---|
| Chapter No. | 29 |
| Status | Completed |

▼ Sockets are a mechanism used to send data from one program to another and vice versa

- There are many kinds of sockets, but the kind that is most often discussed is the Internet socket

▼ IP Addresses & Ports

- Each end of a socket is associated with a running program and is uniquely identified by a combined IP address and port number
- The IP address identifies which device that end of the socket is attached to
- The port number identifies which program on that device is using the device

▼ Creating a Socket Connection

- Creating a socket connection is a multi-step process that requires one program to be the server and another program to be the client
- The server's IP address and port number for accepting connections must also be known ahead of time by the client
- Once a socket is established, it can send data both from the client to the server and from the server to the client

▼ Process

1. The server creates a passive socket, binds it to a pre-chosen port number and listens for an incoming connection. A passive socket is not connected to another socket and merely waits for an incoming connection.
2. The client initiates a connection request using the server's IP address and port number.
3. If the connection request does not reach an IP address and port number that the server is listening on, the connection will be refused.
4. If the connection request reaches an IP address and port number that the server is listening on, the server accepts and creates a new socket for the requesting client using a dynamically assigned port number.

5. The passive socket goes back to listening for new connections while the client and server can now exchange data using the newly-created socket.

▼ Byte Encoding

- As sockets work at a very basic level, they can only send and receive data in the form of raw bytes
- Thus, we must be able to encode the data into a sequence of 8-bit characters using Python's bytes data type

▼ Working with bytes and string Data Types in Python

▼ str.encode( )

- Converts a string to a bytes using UTF-8 encoding

▼ bytes.decode( )

- Converts bytes to a string using UTF-8 encoding

▼ b"raw bytes"

- To enter a sequence of bytes directly in code, we can use the bytes literal that starts with the letter b, followed by a sequence of bytes enclosed in matching single or double quotes

▼ Python socket Module

▼ bind((host, port))

- Binds a socket object to a given address tuple (host, port), where host is an IPv4 address and port is a port number

▼ listen( )

- Enables a socket to listen for incoming connections from clients

▼ accept( )

- Enables a socket to wait for an incoming connection and returns a tuple containing a new socket object for the connection and and an address tuple (host, port), where host is the IPv4 address of the connected client and port is its port number

▼ connect((host, port))

- Initiates a connection to the given address tuple (host, port), where host is the IPv4 address of the server and port is its port number

▼ recv(max_bytes)

- Receives and returns up the the given number of bytes from the socket

▼ sendall(bytes)

- Sends the given bytes to the socket

▼ Implementing an Iterative Server in Python

- An iterative server is a server that only handles one client at a time

▼ Designing a Protocol

- In general, we should never assume that socket.recv( ) will receive all the bytes that were sent over in one go
- The only way to be certain that any received data is complete is to agree beforehand on a protocol or a set of rules for how communication should take place

▼ Example - Newline Character Suffix Protocol

- For instance, we can agree beforehand that any data we transmit will always end with a newline character "\n" and that the data itself will never contain the "\n" character
- This simple protocol allows us to detect the end of a transmission easily by just searching for the "\n" character

▼ Example - Chat Program

```
#chat server
import socket

listen_socket = socket.socket()
listen_socket.bind(("127.0.0.1", 1234)) #port number is not fixed
listen_socket.listen()

chat_socket, address = listen_socket.accept() #.accept() returns (socket object, (ip, port))

while True:
  data = input("SERVER SAYS: ").encode()
  chat.socket.sendall(data + b"\n")

  data = b""
  while b"\n" not in data:
    data += chat_socket.recv(1024) #number of bytes is not fixed

  print("CLIENT SAYS:", data.decode())
```

```
#chat client
import socket

chat_socket = socket.socket()

ip = input("Enter IPv4 address of server: ") #127.0.0.1 for this server
port = int(input("Enter port number of server: ")) #1234 for this server
```

```
chat_socket.connect((ip, port))

while True:
  data = b""
  while b"\n" not in data:
    data += chat_socket.recv(1024) #number of bytes is not fixed

  print("SERVER SAYS:", data.decode())

  data = input("CLIENT SAYS: ").encode()
  chat.socket.sendall(data + b"\n")
```