

Experiment-7

Aim: Introduction to Socket Programming- Design and Implement client-server elements of a few network applications e.g., Echo client and server, Time client and server, Online Quiz and Buzzer Application, etc.

Prerequisite: Nil

Outcome: To impart knowledge of Socket Programming

Theory:

Socket programming within computer networks enables communication between two processes, whether they are on the same machine or across different machines in the network. A socket comprises an IP address and a software port number, facilitating this communication.

This type of programming can be categorized into two forms: connection-oriented and connectionless. Connection-oriented programming involves using TCP (Transmission Control Protocol), a reliable transport layer protocol that establishes a connection before data transmission. On the other hand, connectionless programming employs UDP (User Datagram Protocol), an unreliable and connectionless transport layer protocol that does not establish a connection before transmitting data.

There exist three main types of socket programming interfaces:

- Stream sockets: Commonly used for connection-oriented services.
- Datagram sockets: Provide connection-less services and are suitable when data loss or disorderly arrival is acceptable.
- Raw sockets: These bypass the built-in protocol support (like UDP and TCP) and are utilized for creating custom low-level protocols.

Regarding the classes used in socket programming, DatagramSocket and DatagramPacket classes are used for connectionless socket programming, whereas Socket and ServerSocket classes are used for connection-oriented socket programming.

The socket programming process involves various steps. On the client-side, functions like socket(), connect(), read(), write(), and close() are used. On the server-side, functions like socket(), bind(), listen(), read(), write(), and accept() are employed. For UDP socket programming, client-side functions include socket(), recvfrom(), and sendto(), while server-side functions encompass socket(), recvfrom(), sendto(), and bind().

Procedure:

1. Write Simple Client Server Program using Java/Python Programming Language.
2. Execute the program using appropriate compiler.
3. Verify the working of the program.

Steps:

Server.py

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

host = socket.gethostname()

port = 12345

server_socket.bind((host, port))

server_socket.listen(1)

print(f'Server listening on {host}:{port}...')

client_socket, addr = server_socket.accept()

print(f'Connection from: {addr}')

data = client_socket.recv(1024).decode('utf-8')

print(f'Received: {data}')

client_socket.close()
```

Client.py

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

host = socket.gethostname()

port = 12345

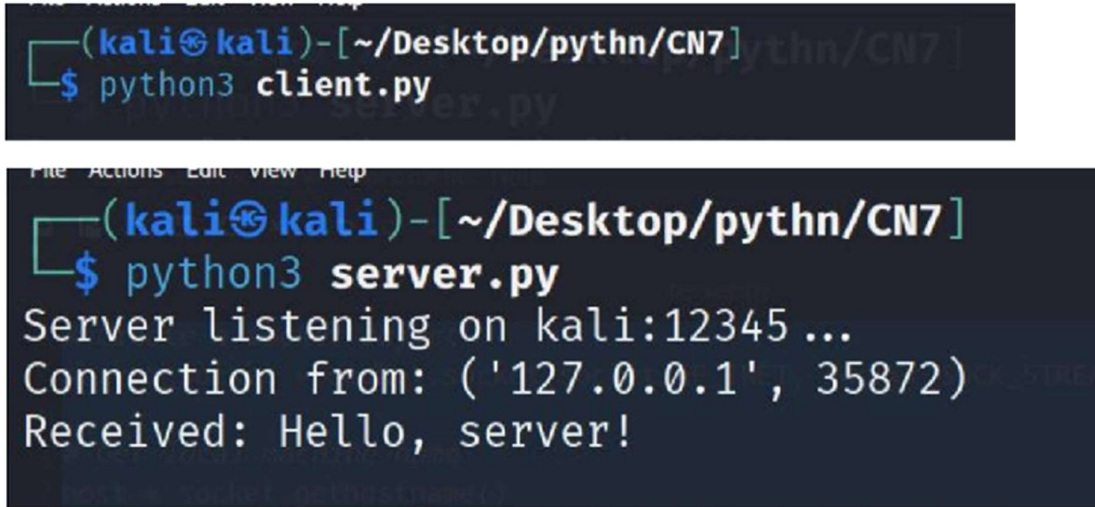
client_socket.connect((host, port))

message = "Hello, server!"

client_socket.send(message.encode('utf-8'))
```

```
client_socket.close()
```

Output:



The image contains two terminal window screenshots. The top screenshot shows a terminal with the prompt `(kali㉿kali)-[~/Desktop/pythn/CN7]` and the command `$ python3 client.py` being entered. The bottom screenshot shows the same terminal with the command `$ python3 server.py` entered. The output of the server program is displayed: `Server listening on kali:12345 ...`, `Connection from: ('127.0.0.1', 35872)`, and `Received: Hello, server!`.

Observation & Learning

Socket programming facilitates communication between devices over a network, allowing processes to transmit and receive data.

1. **Socket Varieties:** The Python socket library enables the creation of diverse socket types, including TCP/IP (stream) and UDP (datagram) sockets.
2. **Server-Client Interaction:** Servers actively listen for incoming connections, while clients initiate connections to servers. Once connected, they can engage in data exchange.
3. **Port and IP Address:** Communication takes place via ports. Servers are designed to listen on specific ports, and clients connect to a server using its IP address and port to establish a connection.
4. **Communication Protocols:** Socket programming relies on common protocols such as TCP (reliable and connection-oriented) and UDP (unreliable and connectionless) for various applications and communication needs.

Conclusion

In summary, socket programming stands as a crucial element in computer network development, facilitating communication between processes across a network. It offers a versatile and effective method for transmitting data among various machines or processes.

Questions:

1. What is Socket?

Ans:

- Sockets are established through a 'socket' system call, each bearing a specific address comprised of an IP address and a designated port number. This address serves as the connection point to the network. There are two primary types of sockets: datagram sockets, offering connection-less services, and stream sockets, providing connection-oriented services.
- Typically utilized in client-server applications, the server initializes a socket, assigns it to a network port address, and awaits contact from the client. Meanwhile, the client creates its socket and endeavors to establish a connection with the server socket. Once the connection is established, data transmission occurs.

2. Which socket is used for the communication between the client and server?

Ans: The prevalent socket types for client-server communication are:

- TCP (Transmission Control Protocol): TCP ensures reliable, connection-oriented communication between the client and server, ensuring accurate data delivery and sequence. It finds common usage in applications where data integrity is critical, such as web browsing, email, and file transfers.
- UDP (User Datagram Protocol): UDP operates as a connectionless protocol, offering faster communication, albeit with less reliability compared to TCP. It doesn't assure packet delivery or order, making it fitting for applications like real-time streaming, online gaming, and DNS, where speed takes precedence over guaranteed delivery.

3. What the different operation supported on sockets?

Ans:

- Socket Creation (socket()): Initiates a new socket and generates a unique socket descriptor, which serves as an identifier for the socket. It commonly involves specifying parameters like the address family (e.g., AF_INET for IPv4), socket type (e.g., SOCK_STREAM for TCP, SOCK_DGRAM for UDP), and optionally, a protocol (e.g., IPPROTO_TCP for TCP).

Example: `server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`

- Binding (`bind()`): Associates a socket with a particular address and port on the local machine. This step is crucial for servers as it designates the address and port they will be listening on.

Example: `server_socket.bind((host, port))``

- Listening (`listen()`): Specifically for server sockets, this action readies the socket to accept incoming connections and sets the maximum number of queued connections the system can handle.

Example: `server_socket.listen(5)`

- Accepting Connections (`accept()`): Pauses the server's operation until a client establishes a connection. It provides a new socket object symbolizing the connection with the client and the client's address.

Example: `client_socket, addr = server_socket.accept()`

- Connecting (`connect()`): For client sockets, this function establishes a connection to a remote server identified by its address and port.

Example: `client_socket.connect((host, port))`

- Sending Data (`send()`): Transmits data from one socket to another. For TCP sockets, it sends a byte stream, while for UDP sockets, it sends a datagram.

Example: `client_socket.send(message.encode('utf-8'))`

- Receiving Data (`recv()`): Gathers data from the socket. For TCP sockets, it retrieves a byte stream, while for UDP sockets, it collects a datagram.

Example for TCP: `data = client_socket.recv(1024).decode('utf-8')`

Example for UDP: `data, addr = client_socket.recvfrom(1024)`

- Closing a Socket (`close()`): Releases the resources connected to the socket and concludes the connection.

Example: `client_socket.close()`