



**School of Computing Sciences Pak-Austria Fachhochschule:  
Institute of Applied Sciences and Technology, Haripur, Pakistan**

# **Lab 05**

## **Iterated TCP Server**

Course: COMP-352L (Computer Networks Lab)

Lab Demonstrator: Jarullah Khan

## **Iterated Server**

- Server does not terminate after closing the connection with a client, it starts listening for the next connection.
- The server exits when it is closed from outside.
- It processes one client's request completely from start to finish before it can accept and handle a new request from another client.

### **How an Iterative Server Works**

- **Listen and Accept:** The server starts by listening for incoming client connections. When a connection request arrives, it accepts the connection.
- **Service the Request:** Once the connection is established, the server dedicates itself entirely to that one client. It reads the client's request, performs the necessary processing, and sends a response back.
- **Close and Repeat:** After the transaction is completed and the response is sent, the server closes the connection with that client. Only then does it return to its listening state, ready to accept the next incoming connection.

Iterated\_server.py

```
# Iterated TCP Server.
```

```
# Conversation with a single client at a time.
```

```
import socket
```

```
def start_server():
```

```
    host = "127.0.0.1"    # localhost
```

```
    port = 5000           # arbitrary/random port
```

```
    # Create socket object
```

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

```
    # Bind socket to host and port
```

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

```
    # Listen for incoming connections
```

```
    server_socket.listen(1)
```

```
    print(f"[*] Server started.")
```

```
    while True:
```

```
        print(f"Listening on {host}:{port}")
```

```
        # Accept client connection
```

```
        conn, addr = server_socket.accept()
```

```
        print(f"[+] Connected by {addr}")
```

```
        # Communicate with client
```

```
        data = conn.recv(1024).decode()
```

```
        print(f"[Client]: {data}")
```

```
        response = "Hello, client! I received your message."
```

```
        conn.send(response.encode())
```

```
        # Close connection
```

```
        conn.close()
```

```
        print("[-] Client disconnected.")
```

```
if __name__ == "__main__":
```

```
    start_server()
```

simple\_client.py

# Simple socket based TCP client

```
import socket
```

```
def start_client():
```

```
    host = input("Enter server IP Address: ")
```

```
    port = int(input("Enter server Port: "))
```

```
    # Create socket object
```

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

```
    # Connect to server
```

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

```
    print(f"Connected to server at {host}:{port}")
```

```
    # Send message
```

```
    message = input("Message: ")
```

```
    client_socket.send(message.encode())
```

```
    # Receive response
```

```
    response = client_socket.recv(1024).decode()
```

```
    print(f"[Server]: {response}")
```

```
    # Close connection
```

```
    client_socket.close()
```

```
if __name__ == "__main__":
```

```
    start_client()
```

Running the client and server programs/scripts

Server is started once.

Listen. Accept. Respond. Disconnect. Repeat

```
○ jarullah@saturn:lab05 | $ python3 iterated_server.py
[*] Server started.
Listening on 127.0.0.1:5000
[+] Connected by ('127.0.0.1', 56317)
[Client]: Hello Server!
[-] Client disconnected.
Listening on 127.0.0.1:5000
[+] Connected by ('127.0.0.1', 56319)
[Client]: Message from Client 1
[-] Client disconnected.
Listening on 127.0.0.1:5000
[+] Connected by ('127.0.0.1', 56320)
[Client]: Message from Client 2
[-] Client disconnected.
```

Client terminates after response from server.

```
● jarullah@saturn:lab05 | $ python3 simple_client.py
Enter server IP Address: 127.0.0.1
Enter server Port: 5000
Connected to server at 127.0.0.1:5000
Message: Message from Client 1
[Server]: Hello, client! I received your message.
● jarullah@saturn:lab05 | $ python3 simple_client.py
Enter server IP Address: 127.0.0.1
Enter server Port: 5000
Connected to server at 127.0.0.1:5000
Message: Message from Client 2
[Server]: Hello, client! I received your message.
○ jarullah@saturn:lab05 | $
```

## TASKS

### Task 01

Test the server by trying to connect multiple clients at a time.

- Set the back log to 3, "socket.listen(3)".
- Run 5 instances of the client program/script.
  - Open 5 terminal windows and in each one run the client program/script.
- Try connecting all 5 clients to the server.
- Each client should send a message to the server like, "Client 1", "Client 2", "Client 3",...
- The server should respond with "Connected to Client  $N$ ", where  $N$  is the number of the client.

## Task 02

Enhance the server to support **continuous conversation**. (You will have to modify the client as well)

### Requirements:

#### 1. Client connect

- Clients should ask the user for the server's **IP Address** and **Port**.
- Then establish a connection to the server using the given IP Address and Port.
- Immediately after the connection is established the server should respond with (without waiting for client message) "Connection established message" which the client should display in its console.
- E.g.

```
Enter server IP Address: 127.0.0.1
```

```
Enter server port: 8000
```

```
[Server]Connection Established.
```

#### 2. Client/Server Conversation(Echo)

- The client should ask the user for the message to be sent to the server.
- The server should append "**(Echoed)**" to the message and send it back to the client.
- This conversation continues until the client sends a "**disconnect**" message
- E.g.

```
[Client] Hello.
```

```
[Server] Hello (Echoed)
```

```
[Client] Testing server echo.
```

```
[Server] Testing server echo (Echoed)
```

### **3. Client disconnect**

- Client should be able to send a "disconnect" message at which point the server closes the connection with the client and is ready to accept a new connection.

```
[Client] disconnect
```

```
[Server] Connection terminated.
```

```
Listening on 127.0.0.1:8000
```



### Task 03

Enhance the server to support **user login and registration**.

#### Requirements:

##### 4. Registration

- Clients should be able to register with a **username and password**.
- Store credentials in a dictionary or a text file.

```
[Server] Welcome! Would you like to (1) Register or (2) Login?
```

```
[Client] 1
```

```
[Server] Username:
```

```
[Client] jarullah
```

```
[Server] password
```

```
[Client] 12345678
```

```
[Server] Retype password
```

```
[Client] 12345678
```

```
[Server] User 'jarullah' registered successfully!
```

##### 5. Login

- Clients must login using their registered credentials.
- If login is successful, display: *"Welcome <username>"*.

```
[Server] Welcome! Would you like to (1) Register or (2) Login?
```

```
[Client] 2
```

```
[Server] Username:
```

```
[Client] jarullah
```

```
[Server] password
```

```
[Client] 12345678
```

```
[Server] Welcome jarullah!
```

If login fails, allow retry.

[Server] Welcome! Would you like to (1) Register or (2) Login?

[Client] 2

[Server] Username:

[Client] jarulla

[Server] user unknown. Try again.

Username:

[Client] jarullah

[Server] Password

[Client] 12345679

[Server] Password incorrect. Try again

Password:

[Client] 12345678

[Server] Welcome jarullah!