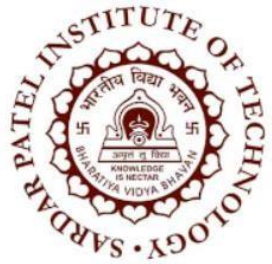



Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
An Autonomous Institute Affiliated To University Of Mumbai
Munshi Nagar,Andheri (W) Mumbai 400 058

# DISTRIBUTED COMPUTING
# EXPERIMENT 1

## Submitted By:

**Name: Adwait Purao**
**UID:** 2021300101
**Batch:** B2

## Submitted To:

Prof. Pramod Bide

## Aim:

Implementation of Client-Server Communication using Sockets

## Theory:

**Sockets Overview:**
Sockets are a fundamental communication mechanism that enables data exchange between processes or applications running on different devices over a network. In client-server architecture, sockets are used to establish connections and facilitate communication.

**Components:**

**Client:** The client is the application or system that initiates a connection request to the server. It typically requests services or data from the server.

**Server:** The server is responsible for listening to incoming connection requests from clients and providing services or data as requested.

**Socket Types:**
There are two primary socket types used in client-server communication:

**a. TCP Sockets:** Transmission Control Protocol (TCP) sockets provide reliable, connection-oriented communication. They ensure data integrity and order but may have higher overhead.

**b. UDP Sockets:** User Datagram Protocol (UDP) sockets offer faster, connectionless communication but do not guarantee data integrity or order. They are suitable for scenarios where speed is prioritized over reliability.

**Key Steps in Client-Server Communication:**

**Server Initialization:** The server creates a socket, binds it to a specific IP address and port, and starts listening for incoming client connections.

**Client Connection:** The client creates a socket and initiates a connection to the server's IP address and port.

**Data Exchange:**

1. **Server Reception:** The server accepts incoming client connections, creating a new socket for communication. It receives data from the client through this socket.
2. **Client Transmission:** The client sends data to the server using its socket.
3. **Data Processing**: The server processes the received data, performs requested operations (e.g., handling inventory requests), and may send a response back to the client.
4. **Connection Termination:** Both client and server can close their sockets when the communication is complete.
5. **Error Handling:** Effective error handling is crucial in socket communication. It involves dealing with issues such as connection failures, data transmission errors, and unexpected disconnections.
6. **Scalability and Performance:** The experiment may explore how well the client-server architecture scales under various loads, considering factors like concurrency, response times, and resource utilization.
7. **Security:** Security measures like encryption and authentication may be implemented to protect data privacy and system integrity.
8. **Socket Libraries:** Depending on the programming language, libraries like Python's socket, Java's Socket and Server Socket, or Node.js's net module can be used to implement sockets.

## Code:

### Server.py:

```
import socket
import pickle

# Sample inventory data (in-memory storage)
inventory = {}

def add_item(item_name, quantity):
    if item_name in inventory:
        inventory[item_name] += quantity
    else:
        inventory[item_name] = quantity

def view_inventory():
    return inventory

def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('127.0.0.1', 12345))
    server.listen(5)

    print("Server is listening...")

    while True:
        client_socket, client_address = server.accept()
        print(f"Accepted connection from {client_address}")

        request = client_socket.recv(1024)
        request = pickle.loads(request)

        if request['action'] == 'add':
            item_name = request['item_name']
            quantity = request['quantity']
            add_item(item_name, quantity)
            response = {'message': f"Added {quantity} {item_name}(s) to the
inventory."}
        elif request['action'] == 'view':
```

```
            response = {'inventory': view_inventory()}

        client_socket.send(pickle.dumps(response))
        client_socket.close()

if_name_== "_main__":
    main()
```

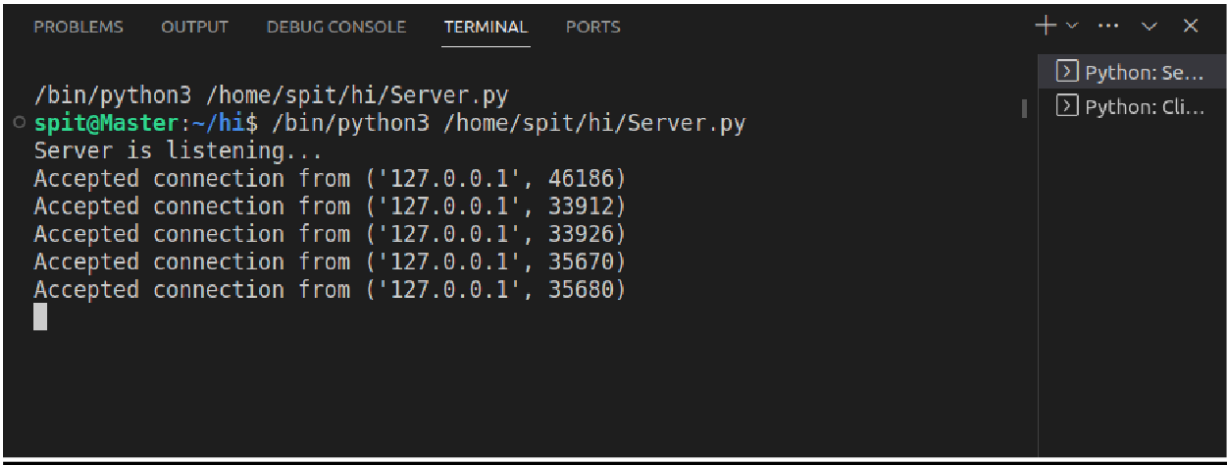
Client.py:

```
import socket
import pickle

def send_request(request):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 12345))

    client.send(pickle.dumps(request))
    response = client.recv(1024)

    response = pickle.loads(response)
    print(response)

    client.close()

def main():
    while True:
        print("\nOptions:")
        print("1. Add Item to Inventory")
        print("2. View Inventory")
        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            item_name = input("Enter item name: ")
            quantity = int(input("Enter quantity to add: "))
```

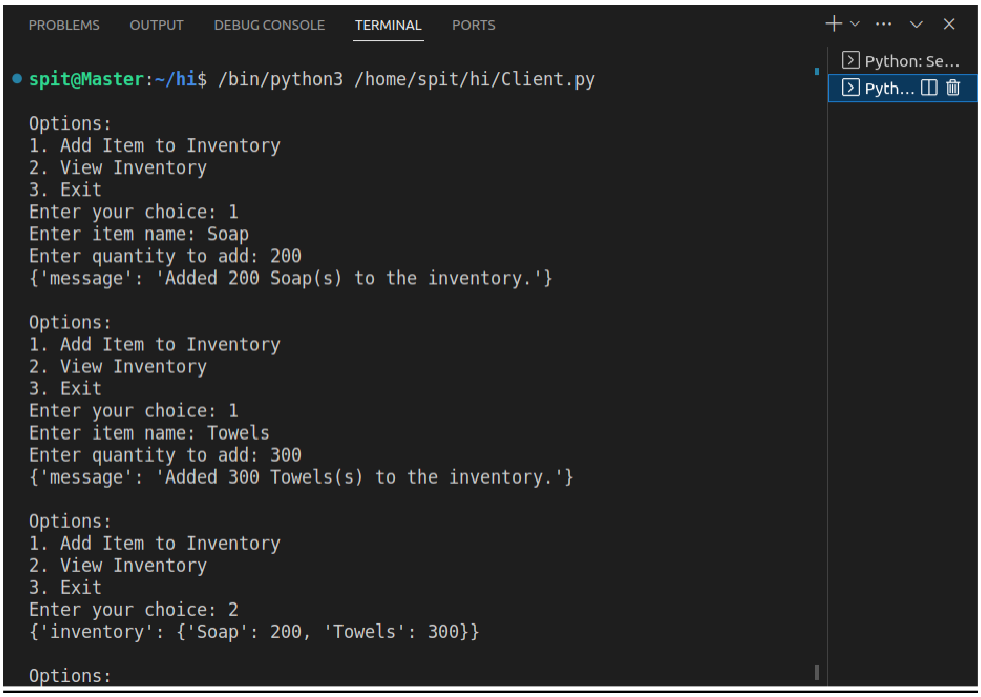
```
            request = {'action': 'add', 'item_name': item_name, 'quantity':
quantity}
            send_request(request)
        elif choice == '2':
            request = {'action': 'view'}
            send_request(request)
        elif choice == '3':
            break
        else:
            print("Invalid choice. Try again.")

if_name_== "_main__":
    main()
```

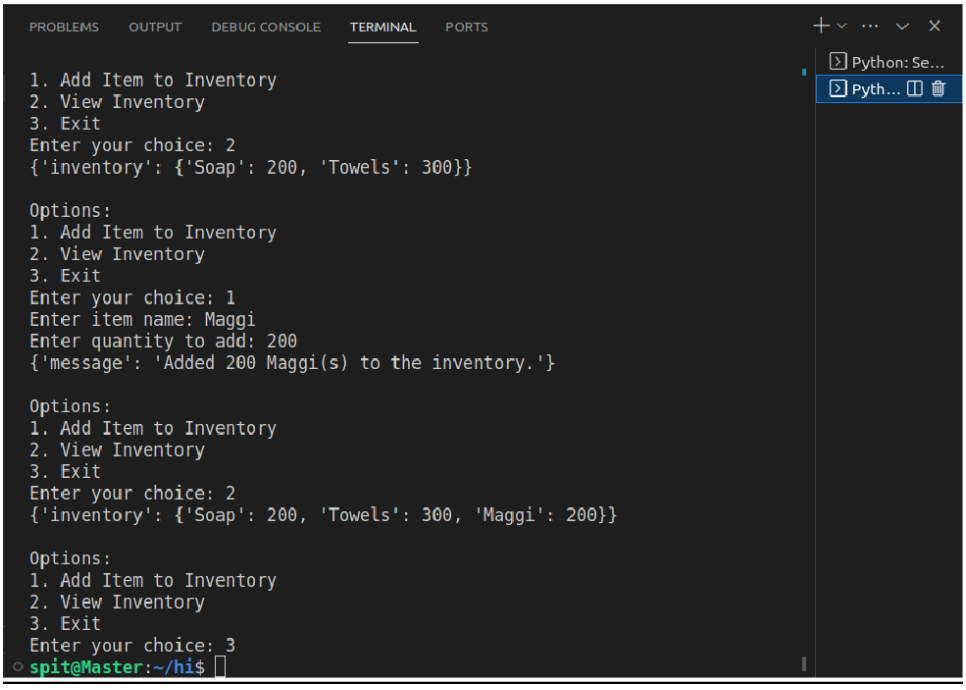
**Output:**

Server:



## Client:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Pyth...
spit@Master:~/hi$ /bin/python3 /home/spit/hi/Client.py
Options:
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 1
Enter item name: Soap
Enter quantity to add: 200
{'message': 'Added 200 Soap(s) to the inventory.'}
Options:
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 1
Enter item name: Towels
Enter quantity to add: 300
{'message': 'Added 300 Towels(s) to the inventory.'}
Options:
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 2
{'inventory': {'Soap': 200, 'Towels': 300}}
Options:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Pyth...
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 2
{'inventory': {'Soap': 200, 'Towels': 300}}
Options:
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 1
Enter item name: Maggi
Enter quantity to add: 200
{'message': 'Added 200 Maggi(s) to the inventory.'}
Options:
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 2
{'inventory': {'Soap': 200, 'Towels': 300, 'Maggi': 200}}
Options:
1. Add Item to Inventory
2. View Inventory
3. Exit
Enter your choice: 3
spit@Master:~/hi$

## Conclusion:

In summary, this experiment centered on implementing client-server communication using sockets, a crucial networking mechanism. It covered the core components of client-server architecture, socket types (TCP and UDP), and the essential communication steps.
Implementing client-server communication using sockets is a fundamental technique for building networked applications.
The Python code demonstrated a practical application of socket programming with a server managing an inventory system.
Socket programming remains a fundamental skill for networking and distributed applications, highlighting the importance of efficient communication in modern computing environments.