BHARATIYA VIDYA BHAVANS
SARDAR PATEL INSTITUTE OF TECHNOLOGY
MUNSHI NAGAR, ANDHERI (WEST), MUMBAI - 400 058
AUTONOMOUS INSTITUTE AFFILIATED TO MUMBAI UNIVERSITY

Adwait Purao - 2021300101

TE Comps B - Batch B

DC LAB

DISTRIBUTED COMPUTING EXPERIMENT 9

Aim:

Implementation of a Client Server based program to check data consistency.

Theory:

Client-Server Architecture:

Client-Server Model:
The client-server architecture is a common paradigm in distributed systems where tasks are divided between clients and servers. Clients request services or resources, and servers provide those services or resources. In the context of an inventory management system, clients can request operations like adding items, removing items, or checking the inventory status.

Inventory Management System:
Basic Operations:

Adding Items (ADD): Clients can add items to the inventory with a specified quantity.
Removing Items (REMOVE): Clients can remove items from the inventory, ensuring data consistency.
Viewing Inventory Status (STATUS): Clients can check the current status of the inventory.

Characteristics:

Communication: Clients and servers communicate over a network, typically using protocols such as TCP/IP.
Decentralized Processing: Clients and servers operate independently, allowing for better scalability and distribution of computational load.
Resource Sharing: Servers provide shared resources and services to multiple clients.

Data Consistency in Distributed Systems:
Definition:
Data consistency in distributed systems refers to the uniformity and accuracy of shared data across multiple nodes. Maintaining consistency becomes challenging due to factors such as network latency, node failures, and the need for synchronization among distributed entities.

Challenges:

Concurrency: Coordinating concurrent access to shared resources to prevent conflicts.
Fault Tolerance: Ensuring data consistency in the presence of node failures.
Scalability: Balancing the trade-off between consistency and system scalability.
Consistency Models:

Strong Consistency: Guarantees that all nodes see the same data at the same time.
Eventual Consistency: Allows for temporary inconsistencies but guarantees convergence over time.
Causal Consistency: Preserves causal relationships between related operations.
Consistency Protocols:

Two-Phase Commit (2PC): Coordinates the commit or rollback of transactions across distributed nodes.
Quorum-Based Systems: Decision-making based on a majority vote or agreement among a subset of nodes.
Vector Clocks: Tracking causality in distributed systems for partial ordering of events.

Code Relevance:
The provided code exemplifies a basic client-server system for an inventory management application. Key aspects related to distributed computing and data consistency in the code include:

Client-Server Interaction: The code implements a simple client-server communication model where the client sends requests, and the server processes and responds to those requests.

Data Consistency Checks: The server code includes basic data consistency checks during operations like adding and removing items from the inventory. This addresses the challenge of ensuring accurate updates to shared data.

Concurrency Control: While the code doesn't explicitly implement advanced concurrency control, it lays the foundation for understanding how such mechanisms could be integrated in a distributed environment.

Code:

Server.py

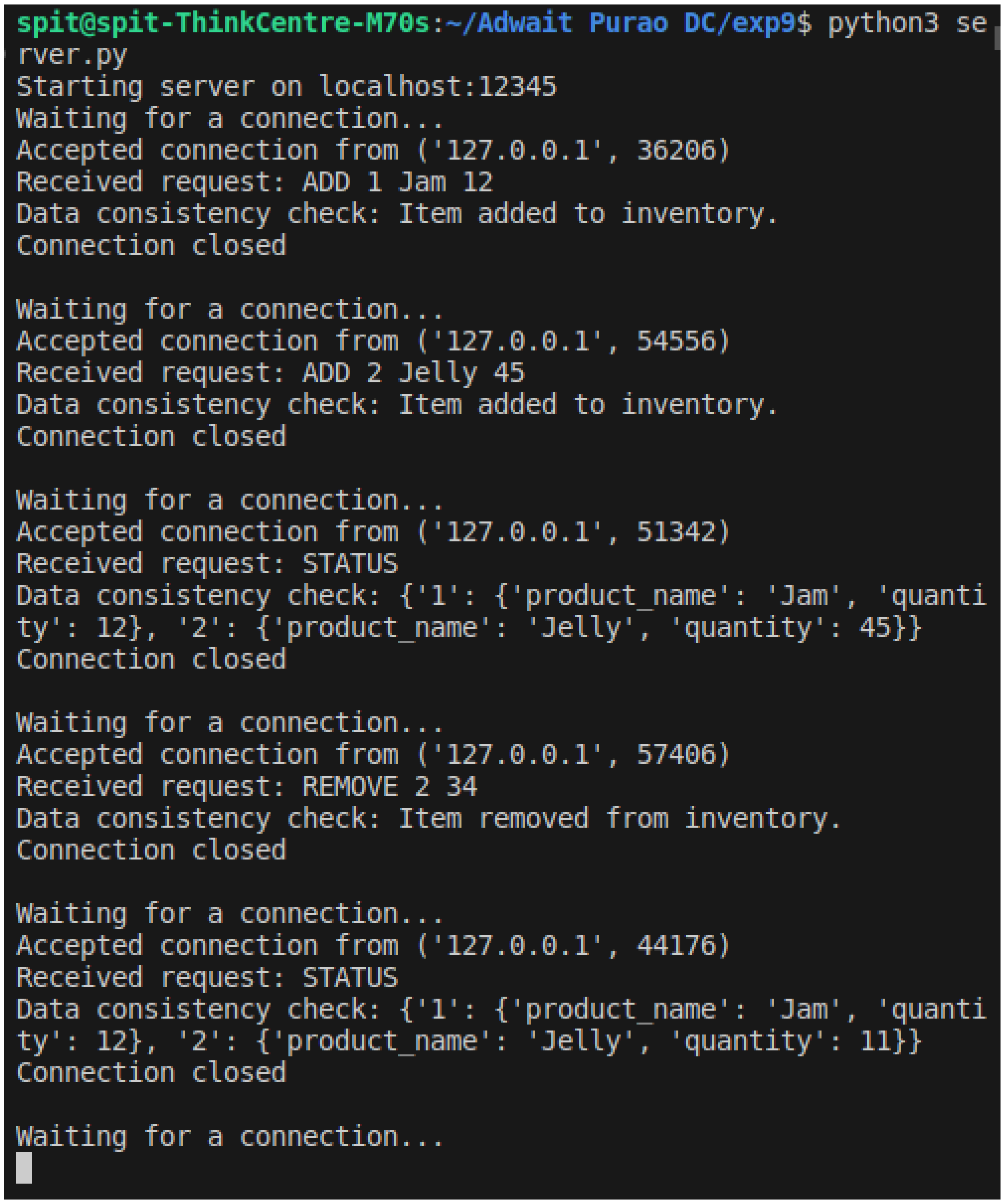
```python
import socket

inventory = {}

def add_item(product_id, product_name, quantity):
   if product_id in inventory:
       inventory[product_id]['quantity'] += quantity
   else:
       inventory[product_id] = {'product_name': product_name, 'quantity':
quantity}

def remove_item(product_id, quantity):
   if product_id in inventory and inventory[product_id]['quantity'] >=
quantity:
       inventory[product_id]['quantity'] -= quantity
       return True  # Indicate consistency
   else:
       print(f"Error: Inconsistent data. Not enough
{inventory.get(product_id, {}).get('product_name')} in the inventory.")
       return False  # Indicate inconsistency
```

```python
def get_inventory_status():
    return inventory

def handle_client_request(request):
    command, *params = request.split()

    if command == "ADD":
        product_id, product_name, quantity = params
        add_item(product_id, product_name, int(quantity))
        return "Item added to inventory."

    elif command == "REMOVE":
        product_id, quantity = params
        consistency_check_passed = remove_item(product_id, int(quantity))
        return f"Item removed from inventory." if consistency_check_passed else "Error: Inconsistent data."

    elif command == "STATUS":
        return str(get_inventory_status())

    else:
        return "Invalid command. Supported commands: ADD, REMOVE, STATUS"

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 12345)

    print(f"Starting server on {server_address[0]}:{server_address[1]}")

    server_socket.bind(server_address)
    server_socket.listen(1)

    while True:
        print("Waiting for a connection...")
        client_socket, client_address = server_socket.accept()
        print(f"Accepted connection from {client_address}")

        request = client_socket.recv(1024).decode('utf-8')
        print(f"Received request: {request}")
        response = handle_client_request(request)
        print(f"Data consistency check: {response}")
        client_socket.send(str(response).encode('utf-8'))

        client_socket.close()
        print("Connection closed\n")

if __name__ == "__main__":
    start_server()
```

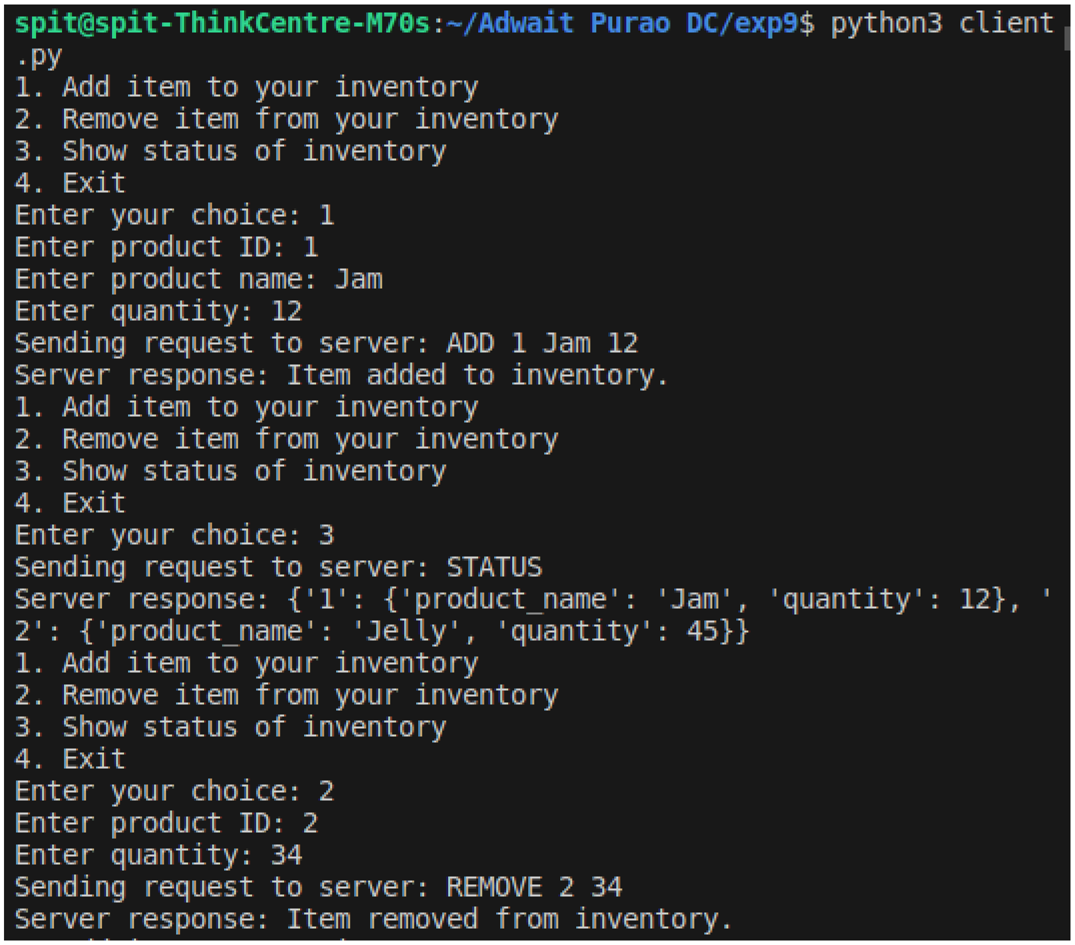
Client.py

```
import socket

def send_request_to_server(request):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 12345)

    client_socket.connect(server_address)

    print(f"Sending request to server: {request}")
    client_socket.send(request.encode('utf-8'))

    response = client_socket.recv(1024).decode('utf-8')
    print(f"Server response: {response}")

    client_socket.close()

if __name__ == "__main__":
    while True:
        print("1. Add item to inventory")
        print("2. Remove item from inventory")
        print("3. View detailed inventory status")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")
        if choice == "1":
            product_id = input("Enter product ID: ")
            product_name = input("Enter product name: ")
            quantity = int(input("Enter quantity: "))
            send_request_to_server(f"ADD {product_id} {product_name}
{quantity}")

        elif choice == "2":
            product_id = input("Enter product ID: ")
            quantity = int(input("Enter quantity: "))
            send_request_to_server(f"REMOVE {product_id} {quantity}")

        elif choice == "3":
            send_request_to_server("STATUS")

        elif choice == "4":
            print("Exiting the program.")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
```

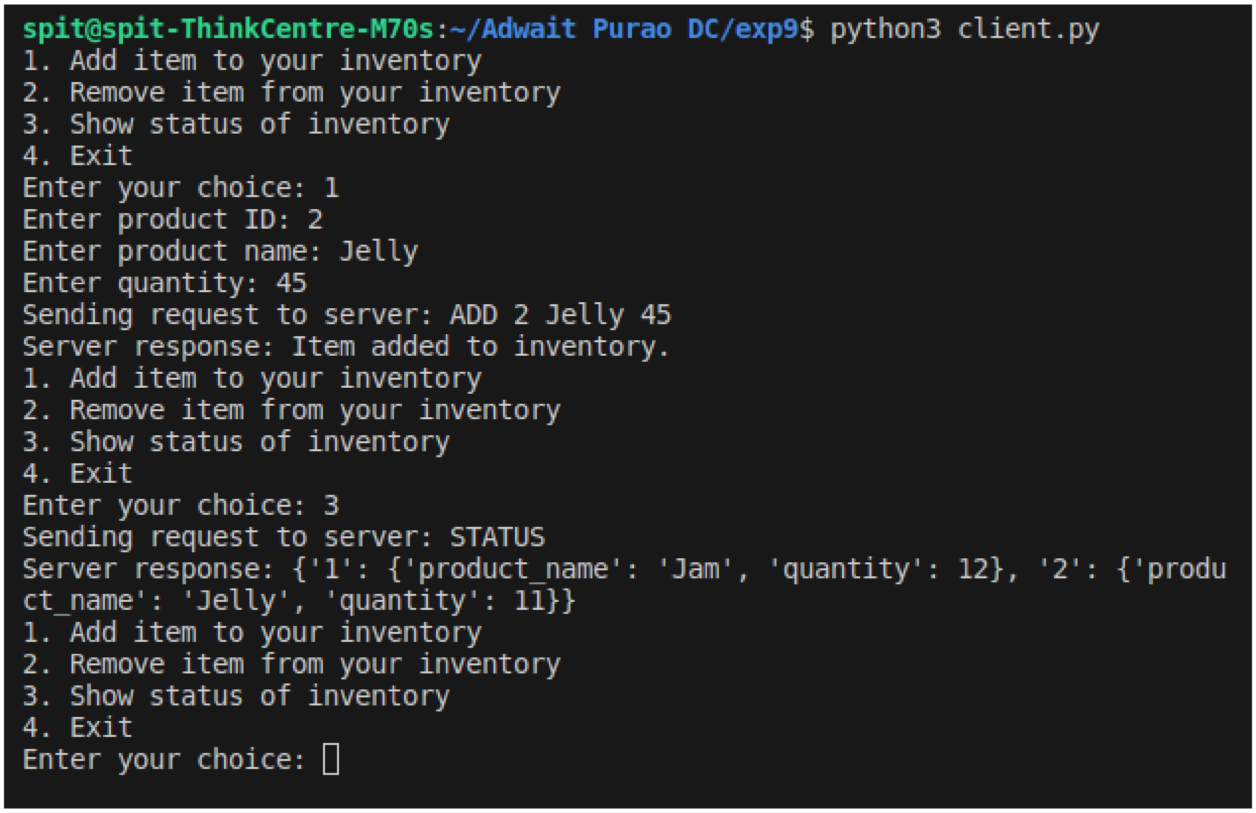
Output:



```
spit@spit-ThinkCentre-M70s:~/Adwait Purao DC/exp9$ python3 se
rver.py
Starting server on localhost:12345
Waiting for a connection...
Accepted connection from ('127.0.0.1', 36206)
Received request: ADD 1 Jam 12
Data consistency check: Item added to inventory.
Connection closed

Waiting for a connection...
Accepted connection from ('127.0.0.1', 54556)
Received request: ADD 2 Jelly 45
Data consistency check: Item added to inventory.
Connection closed

Waiting for a connection...
Accepted connection from ('127.0.0.1', 51342)
Received request: STATUS
Data consistency check: {'1': {'product_name': 'Jam', 'quanti
ty': 12}, '2': {'product_name': 'Jelly', 'quantity': 45}}
Connection closed

Waiting for a connection...
Accepted connection from ('127.0.0.1', 57406)
Received request: REMOVE 2 34
Data consistency check: Item removed from inventory.
Connection closed

Waiting for a connection...
Accepted connection from ('127.0.0.1', 44176)
Received request: STATUS
Data consistency check: {'1': {'product_name': 'Jam', 'quanti
ty': 12}, '2': {'product_name': 'Jelly', 'quantity': 11}}
Connection closed

Waiting for a connection...
```



```
spit@spit-ThinkCentre-M70s:~/Adwait Purao DC/exp9$ python3 client
.py
1. Add item to your inventory
2. Remove item from your inventory
3. Show status of inventory
4. Exit
Enter your choice: 1
Enter product ID: 1
Enter product name: Jam
Enter quantity: 12
Sending request to server: ADD 1 Jam 12
Server response: Item added to inventory.
1. Add item to your inventory
2. Remove item from your inventory
3. Show status of inventory
4. Exit
Enter your choice: 3
Sending request to server: STATUS
Server response: {'1': {'product_name': 'Jam', 'quantity': 12}, '
2': {'product_name': 'Jelly', 'quantity': 45}}
1. Add item to your inventory
2. Remove item from your inventory
3. Show status of inventory
4. Exit
Enter your choice: 2
Enter product ID: 2
Enter quantity: 34
Sending request to server: REMOVE 2 34
Server response: Item removed from inventory.
```



```
spit@spit-ThinkCentre-M70s:~/Adwait Purao DC/exp9$ python3 client.py
1. Add item to your inventory
2. Remove item from your inventory
3. Show status of inventory
4. Exit
Enter your choice: 1
Enter product ID: 2
Enter product name: Jelly
Enter quantity: 45
Sending request to server: ADD 2 Jelly 45
Server response: Item added to inventory.
1. Add item to your inventory
2. Remove item from your inventory
3. Show status of inventory
4. Exit
Enter your choice: 3
Sending request to server: STATUS
Server response: {'1': {'product_name': 'Jam', 'quantity': 12}, '2': {'produ
ct_name': 'Jelly', 'quantity': 11}}
1. Add item to your inventory
2. Remove item from your inventory
3. Show status of inventory
4. Exit
Enter your choice: 
```

Conclusion:

In summary, the assignment introduced a client-server inventory management system in the context of distributed computing. The code demonstrates essential concepts like client-server interaction, data consistency checks, and basic concurrency control. It provides a foundational understanding, encouraging further exploration into advanced distributed computing topics, such as consensus algorithms and fault tolerance. This assignment serves as a starting point for practical application and extension based on specific distributed computing requirements.