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 3

Submitted By:

Adwait Purao	2021300101

Submitted To:

Prof. Pramod Bide

Aim:

Implementation of RMI

Theory:

Remote Method Invocation (RMI) is a fundamental concept in distributed computing, allowing for the invocation of methods on objects located on remote servers as if they were local.
RMI plays a pivotal role in enabling communication and interaction between different components of a distributed system. Within the realm of RMI, Pyro4 is a Python library that simplifies the development of distributed systems, making it a powerful tool for building distributed applications.

Remote Method Invocation (RMI):

Remote Method Invocation (RMI) represents a foundational and transformative concept within the field of distributed computing. It serves as the linchpin for enabling communication and interaction among various components of a distributed system, transcending geographical boundaries and networked infrastructures.
RMI empowers developers by providing them with the capability to seamlessly execute methods on objects residing on remote servers, effectively collapsing the distinctions between local and remote execution.

Core principles that underpin RMI:

1. **Remote Objects:**

- At the heart of RMI lies the concept of remote objects. These objects, often encapsulated within specially designed classes, serve as emissaries representing entities that reside on remote servers. These remote object classes encapsulate not only data but also methods that can be invoked remotely by clients.
- The ability to invoke methods on these remote objects as if they were local is a hallmark of RMI.
- This abstraction allows developers to create distributed systems where the complexities of remote communication are hidden beneath a straightforward method invocation, promoting modularity and code organization.

2. **Network Communication:**

- RMI acts as a powerful intermediary that simplifies the intricate nuances of network communication.
- It assumes the responsibility of managing the entire data transmission process, starting with the serialization of data.
- Serialization involves converting complex objects, often containing intricate data structures, into a format that is suitable for efficient transmission across the network.
- On the receiving end, deserialization reverses this process, restoring the transmitted data into its original object form.
- By orchestrating these operations, RMI ensures that data flows seamlessly between remote components, regardless of the underlying complexities of network protocols, addressing, and data serialization formats.

3. **Client-Server Model:**

- RMI thrives within the framework of a client-server model, a paradigm that is foundational in distributed computing.
- In this model, a server exposes its remote objects, replete with their methods, to clients.
- Clients, in response, create proxy objects that act as intermediaries for interacting with these remote objects on the server. The key innovation here is that these proxies abstract away the intricacies of low-level networking details, including socket programming, addressing, and routing.
- As a result, developers are liberated from having to grapple with these complexities, allowing them to concentrate on the core logic and functionality of their applications.
- This abstraction significantly enhances code readability and maintainability.

In essence, Remote Method Invocation (RMI) is not merely a technical mechanism but a transformative approach to distributed computing.
It empowers developers to build distributed systems that function as cohesive wholes, despite the physical separation of their components.
By abstracting the complexities of network communication and seamlessly bridging the gap between local and remote execution, RMI plays a pivotal role in the development of scalable, collaborative, and resilient distributed applications across a wide spectrum of domains.

<u>Pyro4:</u>

- Pyro4 is a versatile Python library that plays a pivotal role in simplifying the practical implementation of Remote Method Invocation (RMI) and the development of distributed systems.
- It offers a comprehensive set of tools that empower developers to create distributed applications efficiently, enhancing productivity and enabling seamless communication between remote components.

Object Registration:

- One of the cornerstones of Pyro4's capabilities is its ability to streamline the registration of remote objects with a naming service.
- This registration process is critical in making remote objects discoverable and accessible to clients over a network. The naming service acts as a directory or registry, allowing clients to locate and invoke remote objects.
- By registering objects, Pyro4 enables a clean separation of concerns, where the server-side components are registered centrally and clients can discover them dynamically.

Network Communication:

- Pyro4 abstracts many of the complexities associated with network communication, offering a straightforward and developer-friendly means of transmitting data between remote objects.
- It bridges the physical separation between distributed components, enabling them to exchange information as if they were co-located.

Proxy Generation:

- In Pyro4-based systems, clients create proxies for remote objects. These proxies act as intermediaries that manage the complexities of network communication on behalf of the client.
- Proxies serve as a bridge between the client and the remote server, allowing the client to invoke methods on the proxy as if it were interacting with a local object. Pyro4 automates the creation of these proxies, ensuring that clients can seamlessly interact with remote services.
- This abstraction simplifies the development process and enhances code readability by making the interaction with remote components as intuitive as possible.

Exception Handling:

- Pyro4 places a strong emphasis on robust exception handling mechanisms.
- These mechanisms are essential for addressing potential issues that may arise during remote method invocation. Since networked communication introduces additional points of failure, it's crucial to handle exceptions gracefully to ensure the reliability and resilience of distributed communication.
- Pyro4 provides tools for capturing and propagating exceptions that occur during remote method calls, allowing developers to implement error-handling strategies that suit their specific use cases.

In summary, Pyro4 stands as a powerful and developer-friendly Python library for building distributed applications and implementing RMI.

Its features, including object registration, simplified network communication, proxy generation, and robust exception handling, make it a versatile choice for developing scalable, collaborative, and fault-tolerant distributed systems across various domains.

Code:

Server:

```
import Pyro4

# Define the remote object class
@Pyro4.expose
class InventoryManager:
  def __init__(self):
      self.inventory = {}  # Simulate inventory data with a dictionary
      self.item_id_counter = 1  # Counter for generating unique item IDs

  def _print_request_info(self, request_type, item_id=None,
item_name=None, quantity=None):
      print(f"Received {request_type} request.")
      if item_id:
          print(f"Item ID: {item_id}")
```

```
        if item_name:
            print(f"Item Name: {item_name}")
        if quantity:
            print(f"Quantity: {quantity}")

    def add_item(self, item_name, quantity):
        self._print_request_info("Add", item_name=item_name,
quantity=quantity)
        item_id = self.item_id_counter
        self.inventory[item_id] = {"name": item_name, "quantity": quantity}
        self.item_id_counter += 1
        return f"Item '{item_name}' with ID {item_id} added successfully."

    def update_item(self, item_id, quantity):
        self._print_request_info("Update", item_id=item_id,
quantity=quantity)
        if item_id in self.inventory:
            self.inventory[item_id]["quantity"] = quantity
            return f"Quantity for item ID {item_id} updated successfully."
        else:
            return f"Item with ID {item_id} not found."

    def delete_item(self, item_id):
        self._print_request_info("Delete", item_id=item_id)
        if item_id in self.inventory:
            del self.inventory[item_id]
            return f"Item with ID {item_id} deleted successfully."
        else:
            return f"Item with ID {item_id} not found."

    def get_item(self, item_id):
        self._print_request_info("Get", item_id=item_id)
        if item_id in self.inventory:
            return self.inventory[item_id]["quantity"]
        else:
            return None

    def get_inventory(self):
        self._print_request_info("Get Inventory")
        return self.inventory
```

```python
# Create and register the server
if __name__ == "__main__":
  inventory_manager = InventoryManager()
  daemon = Pyro4.Daemon()
  uri = daemon.register(inventory_manager)

  with Pyro4.locateNS() as ns:
      ns.register("InventoryManagerServer", uri)  # Use a different name

  print("InventoryManager server is running.")
  daemon.requestLoop()
```

Client:

```python
import Pyro4


# Create an RMI proxy to access the remote InventoryManager
inventory_manager = Pyro4.Proxy("PYRONAME:InventoryManagerServer")  # Use
the same name as registered in server.py


# Define client functions to interact with the server
def add_item(item_name, quantity):
  return inventory_manager.add_item(item_name, quantity)


def update_item(item_id, quantity):
  return inventory_manager.update_item(item_id, quantity)


def delete_item(item_id):
  return inventory_manager.delete_item(item_id)
```

```python
def get_item(item_id):
  return inventory_manager.get_item(item_id)


def display_all_items():
  all_items = inventory_manager.get_inventory()
  if all_items:
      print("All Items in Inventory:  ")
      for product_id, product_info in all_items.items():
          print(f"Product ID: {product_id}, Name: {product_info['name']},
Quantity: {product_info['quantity']}")
  else:
      print("Inventory is empty.")
  print()

# Function to interactively manage inventory using product IDs
def interactively_manage_inventory():
  while True:
      print("Choose an action:")
      print("  1. Add item")
      print("  2. Update item quantity")
      print("  3. Delete item")
      print("  4. Get item quantity")
      print("  5. Display all items")
      print("  6. Quit")
      print()

      choice = input("Enter your choice: ")


      if choice == "1":
          item_name = input("Enter the item name: ")
          quantity = int(input("Enter the quantity to add: "))
          result = add_item(item_name, quantity)
          print(result)
          print()
      elif choice == "2":
          item_id = int(input("Enter the product ID to update quantity:
"))
          quantity = int(input("Enter the new quantity: "))
```

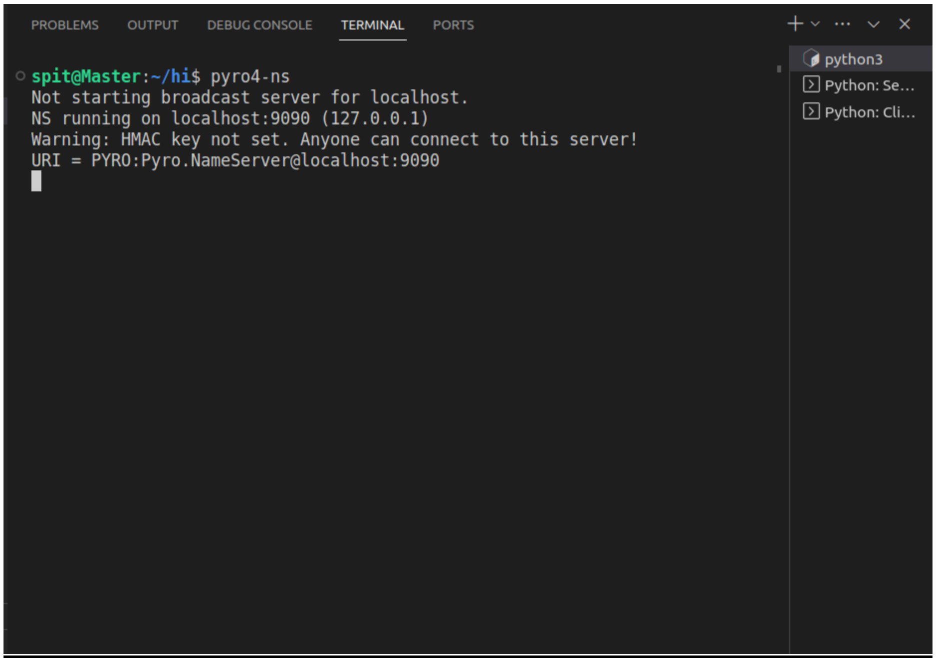
```python
            result = update_item(item_id, quantity)
            print(result)
            print()
        elif choice == "3":
            item_id = int(input("Enter the product ID to delete: "))
            result = delete_item(item_id)
            print(result)
            print()
        elif choice == "4":
            item_id = int(input("Enter the product ID to get quantity: "))
            item_info = get_item(item_id)
            if item_info is not None:
                print(f"Item ID: {item_id}, Quantity: {item_info}")
            else:
                print("Item not found.")
            print()
        elif choice == "5":
            display_all_items()
        elif choice == "6":
            print("Thank You for using our services ")
            break
        else:
            print("Invalid choice. Please try again.")


# Example usage:
if __name__ == "__main__":
  interactively_manage_inventory()
```

How to run:

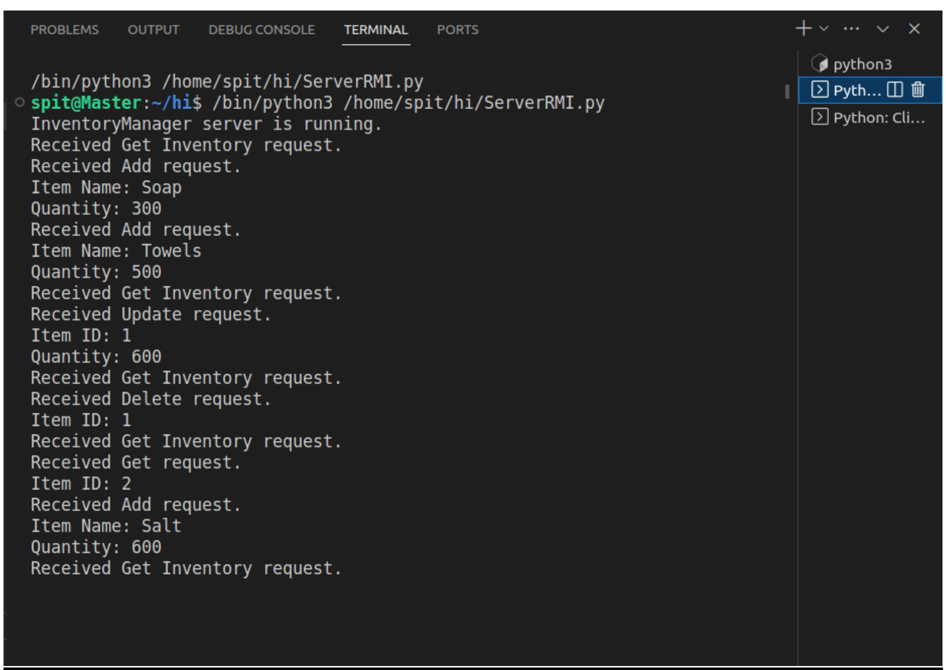
Step 1. Open 3 terminals:
Step 2. 1st terminal: Run pyro4-ns
Step 3. 2nd terminal: Run python3 server.py
Step 4: 3rd terminal: Run python3 client.py

Output:

Step 1: Pyro4ns



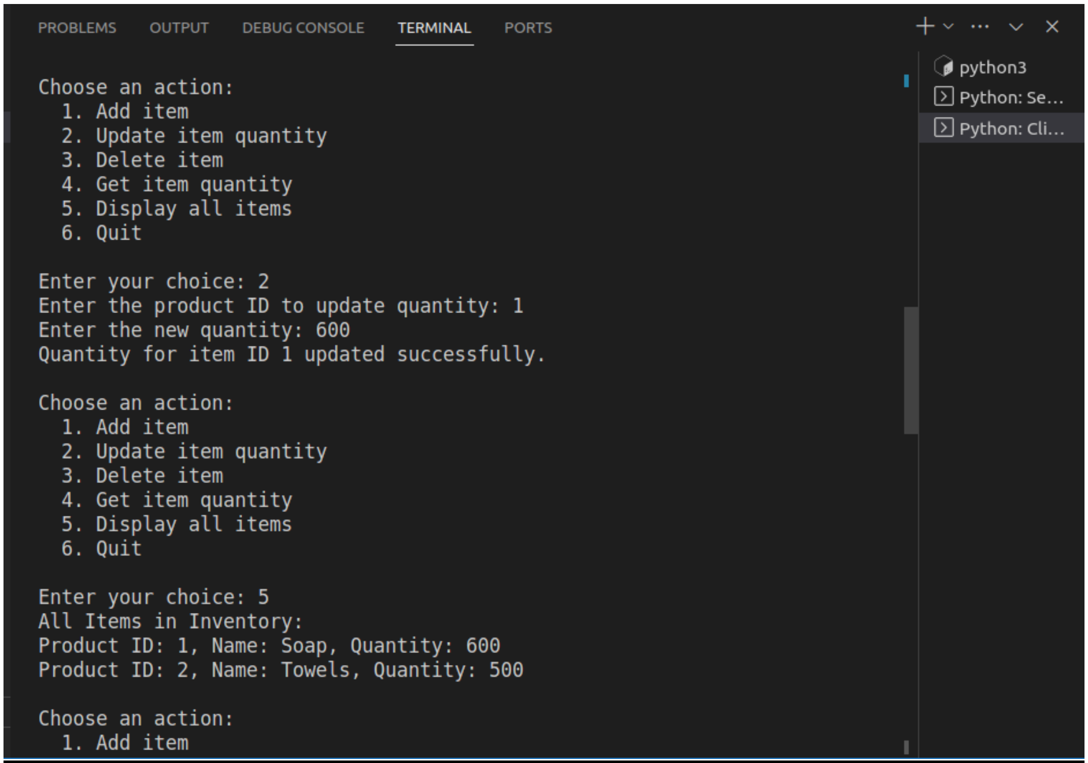
Step 2: Server



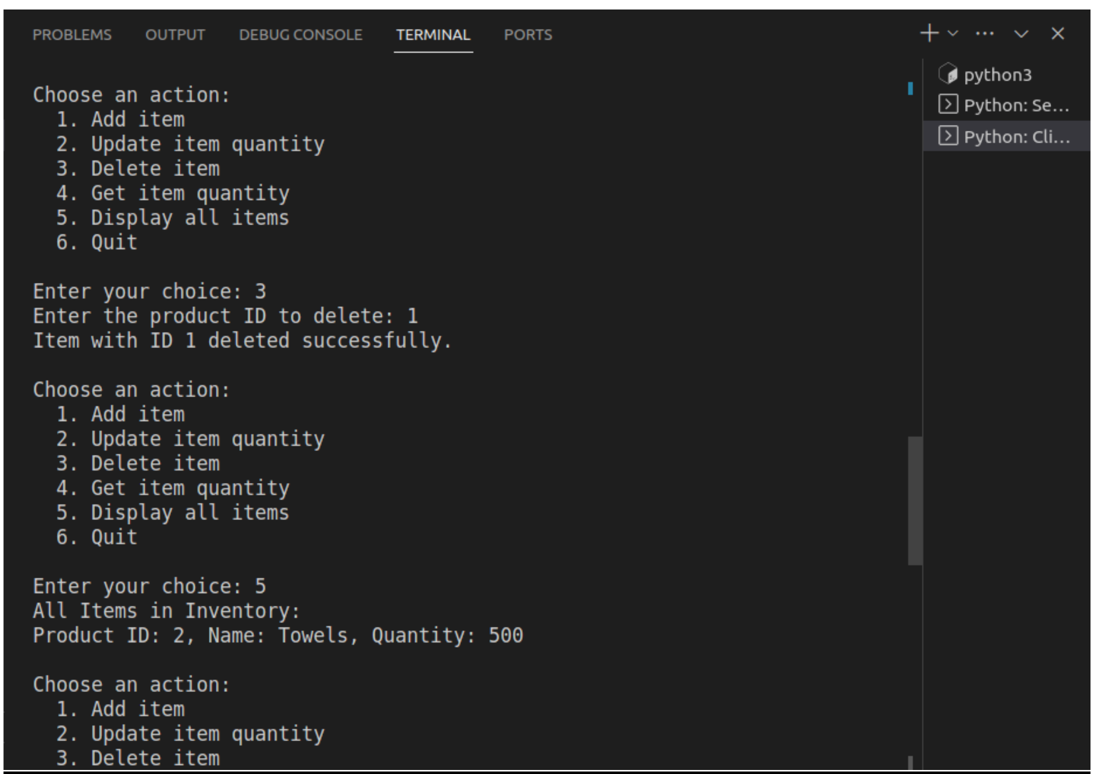
Client:

```
/bin/python3 /home/spit/hi/ClientRMI.py
spit@Master:~/hi$ /bin/python3 /home/spit/hi/ClientRMI.py
Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 5
Inventory is empty.

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 1
Enter the item name: Soap
Enter the quantity to add: 300
Item 'Soap' with ID 1 added successfully.

Choose an action:
  1. Add item
```

```
Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 1
Enter the item name: Towels
Enter the quantity to add: 500
Item 'Towels' with ID 2 added successfully.

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 5
All Items in Inventory:
Product ID: 1, Name: Soap, Quantity: 300
Product ID: 2, Name: Towels, Quantity: 500

Choose an action:
  1. Add item
```




```
Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 2
Enter the product ID to update quantity: 1
Enter the new quantity: 600
Quantity for item ID 1 updated successfully.

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 5
All Items in Inventory:
Product ID: 1, Name: Soap, Quantity: 600
Product ID: 2, Name: Towels, Quantity: 500

Choose an action:
  1. Add item
```




```
Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 3
Enter the product ID to delete: 1
Item with ID 1 deleted successfully.

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 5
All Items in Inventory:
Product ID: 2, Name: Towels, Quantity: 500

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
```

```
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 4
Enter the product ID to get quantity: 2
Item ID: 2, Quantity: 500

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 1
Enter the item name: Salt
Enter the quantity to add: 600
Item 'Salt' with ID 3 added successfully.

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
```

```
Enter the item name: Salt
Enter the quantity to add: 600
Item 'Salt' with ID 3 added successfully.

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 5
All Items in Inventory:
Product ID: 2, Name: Towels, Quantity: 500
Product ID: 3, Name: Salt, Quantity: 600

Choose an action:
  1. Add item
  2. Update item quantity
  3. Delete item
  4. Get item quantity
  5. Display all items
  6. Quit

Enter your choice: 6
Thank You for using our services
spit@Master:~/hi$
```

Conclusion:

In this experiment, we effectively employed Pyro4 to implement RMI, resulting in a distributed inventory management system. The server exposed remote methods for managing an inventory, including adding, updating, deleting, and retrieving items. Clients accessed these methods as if they were local, demonstrating the power of RMI in building distributed applications.

Pyro4 simplified the development process by abstracting many of the complexities related to remote method invocation, allowing seamless communication between clients and the server. The experiment showcased the significance of distributed computing and how RMI can streamline the creation of scalable, collaborative applications in distributed environments.