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 EXP 4

Aim:

Implementation of multi-threading in distributed systems

Theory:

Introduction:

Distributed systems are composed of multiple interconnected computers that work together to achieve a common goal. They are often designed to handle large-scale applications and require efficient utilization of computing resources. Multi-threading is a programming technique that can enhance the performance and responsiveness of distributed systems by allowing concurrent execution of tasks within a single process or across multiple processes.

Multi-Threading:

Multi-threading is a mechanism that enables a single process to have multiple threads of execution. Each thread is a lightweight unit of execution that shares the same memory space, allowing them to communicate and coordinate with each other efficiently. In the context of distributed systems, multi-threading offers several advantages:

1. **Parallelism:** Multi-threading allows different threads to execute tasks simultaneously. This parallelism can significantly improve the overall system throughput, especially when dealing with computationally intensive or I/O-bound operations.
2. **Responsiveness:** In distributed systems, responsiveness is crucial. Multi-threading can keep a system responsive by allowing certain threads to handle user interactions, while others handle background tasks, such as network communication or data processing.
3. **Resource Utilization:** Multi-threading enables efficient utilization of CPU cores and memory resources. This is essential in distributed systems where resource optimization is a primary concern.
4. **Synchronization:** To maintain data integrity and consistency in a distributed environment, threads need to synchronize their activities. Proper synchronization mechanisms, such as locks, semaphores, or message passing, are essential to ensure that multiple threads do not interfere with each other's operations.

Challenges in Multi-Threading in Distributed Systems:

While multithreading offers numerous benefits, it also introduces challenges in the context of distributed systems:

1. **Concurrency Control:** Coordinating threads across multiple distributed nodes can be complex. Ensuring that threads do not access shared resources concurrently and maintaining consistency requires careful design and synchronization.
2. **Fault Tolerance:** Distributed systems are susceptible to failures, such as network outages or node crashes. Managing threads in a way that maintains system integrity and recovers gracefully from failures is a non-trivial task.

3. **Scalability:** As distributed systems grow in size, managing an increasing number of threads can become challenging. Ensuring scalability and efficient load balancing is vital.

Best Practices for Multi-Threading in Distributed Systems: To successfully implement multithreading in distributed systems, consider the following best practices:

1. **Thread Pooling:** Use thread pools to manage and control the number of threads. This helps prevent resource exhaustion and simplifies thread lifecycle management.
2. **Isolation:** Isolate critical sections of code and use appropriate synchronization mechanisms to prevent data corruption and race conditions.
3. **Error Handling:** Implement robust error handling and fault tolerance mechanisms to handle failures gracefully and recover from them.
4. **Load Balancing:** Implement load balancing algorithms to distribute tasks evenly across nodes and threads, ensuring efficient resource utilization.
5. **Monitoring and Profiling:** Use monitoring and profiling tools to identify performance bottlenecks and fine-tune the multi-threaded code for optimal performance.

Multi-threading is a powerful technique that can enhance the performance and responsiveness of distributed systems. However, it comes with challenges that require careful consideration and design. When implemented correctly, multi-threading can lead to more efficient and scalable distributed systems, ultimately delivering a better user experience and improved resource utilization.

Code:

Server:

```
from xmlrpc.server import SimpleXMLRPCServer, SimpleXMLRPCRequestHandler
from socketserver import ThreadingMixIn
import threading


# Inventory management class
class InventoryManager:
  def __init__(self):
      self.inventory = {}
      self.product_id_counter = 1


  def add_item(self, item_name, quantity):
      product_id = self.product_id_counter
      self.inventory[product_id] = {"name": item_name, "quantity":
quantity}
      self.product_id_counter += 1
      return f"Product added with ID {product_id}\n"


  def update_item(self, product_id, quantity):
      if product_id in self.inventory:
          self.inventory[product_id]["quantity"] = quantity
          return f"Product with ID {product_id} quantity updated
successfully\n"
      else:
          return f"Product with ID {product_id} not found\n"


  def delete_item(self, product_id):
      if product_id in self.inventory:
          del self.inventory[product_id]
          return f"Product with ID {product_id} deleted successfully\n"
      else:
          return f"Product with ID {product_id} not found\n"
```

```python
  def get_item(self, product_id):
      if product_id in self.inventory:
          product = self.inventory[product_id]
          return f"Product ID: {product_id}, Name: {product['name']},
Quantity: {product['quantity']}\n"
      else:
          return f"Product with ID {product_id} not found"


  def get_all_items(self):
      # Convert product IDs to strings for compatibility with XML-RPC
      str_inventory = {str(key): value for key, value in
self.inventory.items()}
      return str_inventory


# Custom request handler class with threading enabled
class ThreadedXMLRPCRequestHandler(SimpleXMLRPCRequestHandler):
  rpc_paths = ('/RPC2',)




# Threaded XML-RPC server class
class ThreadedXMLRPCServer(ThreadingMixIn, SimpleXMLRPCServer):
  pass


# Function to handle client requests
def handle_client(client, addr):
  print(f"Accepted connection from {addr}")
  try:
      while True:
          client.handle_request()
  except Exception as e:
      print(f"Error: {e}")
  finally:
      print(f"Connection with {addr} closed.")
```

```python
# Server configuration
server = ThreadedXMLRPCServer(("localhost", 9090),
requestHandler=ThreadedXMLRPCRequestHandler)
server.register_introspection_functions()


# Create an instance of the InventoryManager class
inventory_manager = InventoryManager()


# Register the InventoryManager methods for RPC
server.register_function(inventory_manager.add_item, "add_item")
server.register_function(inventory_manager.update_item, "update_item")
server.register_function(inventory_manager.delete_item, "delete_item")
server.register_function(inventory_manager.get_item, "get_item")
server.register_function(inventory_manager.get_all_items, "get_all_items")




print("Server is listening on port 9090...")


# Start the server in a separate thread
server_thread = threading.Thread(target=server.serve_forever)
server_thread.start()


# Wait for the server thread to finish
server_thread.join()


print("Server closed.")
```

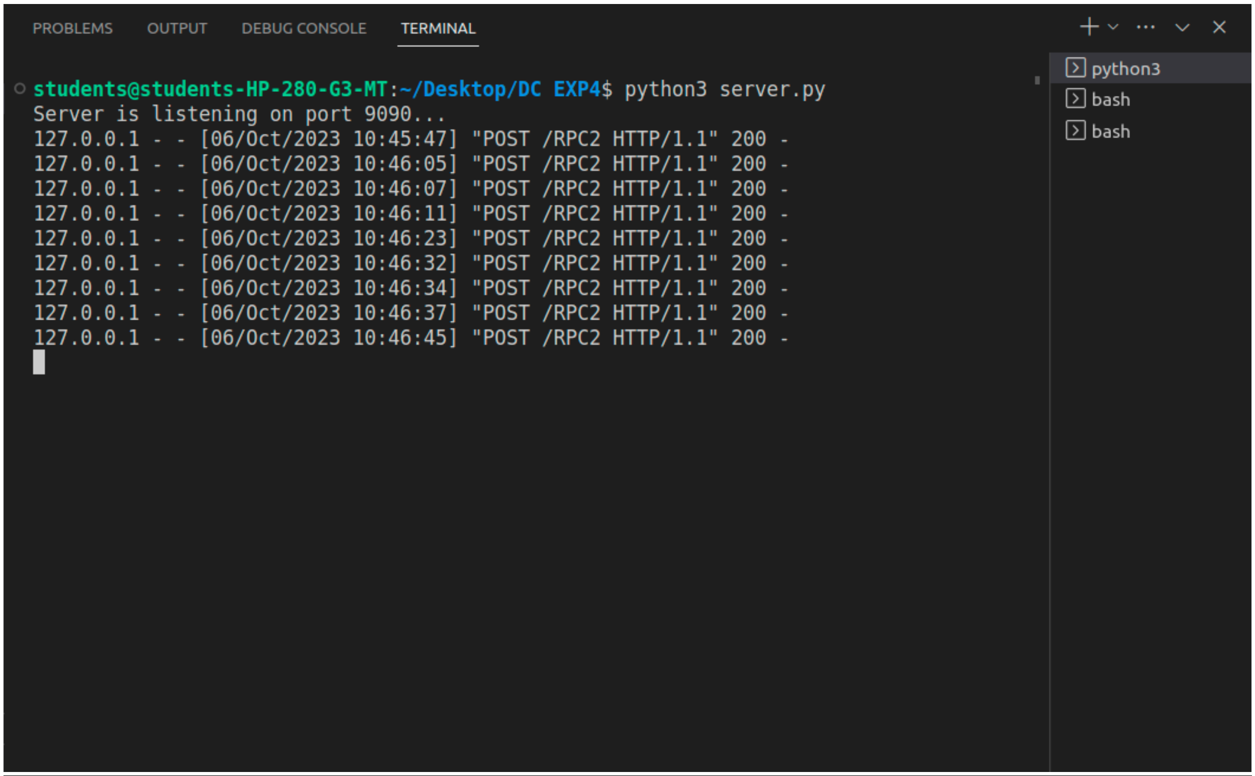
Client:

```
import xmlrpc.client


import threading


# Function to interactively manage inventory using product IDs
def interactively_manage_inventory(proxy):
  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 = proxy.add_item(item_name, quantity)
          print(result)
      elif choice == "2":
          item_id = int(input("Enter the product ID to update quantity:
"))
          quantity = int(input("Enter the new quantity: "))
          result = proxy.update_item(item_id, quantity)
          print(result)
      elif choice == "3":
          item_id = int(input("Enter the product ID to delete: "))
          result = proxy.delete_item(item_id)
          print(result)
      elif choice == "4":
```

```
            item_id = int(input("Enter the product ID to get quantity: "))
            item_info = proxy.get_item(item_id)
            print(item_info)
        elif choice == "5":
            all_items = proxy.get_all_items()
            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']}")
                print("")

            else:
                print("Inventory is empty.")
        elif choice == "6":
            print("Thank You for using our Services\n")
            break
        else:
            print("Invalid choice. Please try again.")


# Function to handle client communication in a separate thread
def client_thread():
  proxy = xmlrpc.client.ServerProxy("http://localhost:9090/RPC2",
allow_none=True)


  interactively_manage_inventory(proxy)

# Example usage
if __name__ == "__main__":
  # Start a separate thread for client communication
  client_thread = threading.Thread(target=client_thread)
  client_thread.start()

  # Wait for the client thread to finish
  client_thread.join()


  print("Client session ended.")
```

Output:

Server:



Client 1:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

students@students-HP-280-G3-MT:~/Desktop/DC EXP4$ python3 client.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: 1
Enter the item name: Salt
Enter the quantity to add: 400
Product added with ID 1

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: Salt, Quantity: 400
Product ID: 2, Name: Sugar, Quantity: 100

Choose an action:

python3
bash
bash
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

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: 2
Enter the new quantity: 4000
Product with ID 2 quantity 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: 3
Enter the product ID to delete: 2
Product with ID 2 deleted successfully

Choose an action:
1. Add item
2. Update item quantity

python3
bash
bash
```

```
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: Salt, Quantity: 400

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: 4
Enter the product ID to get quantity: 1
Product ID: 1, Name: Salt, Quantity: 400

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

```
All Items in Inventory:
Product ID: 1, Name: Salt, Quantity: 400

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: 4
Enter the product ID to get quantity: 1
Product ID: 1, Name: Salt, Quantity: 400

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

Client session ended.
students@students-HP-280-G3-MT:~/Desktop/DC EXP4$
```

Client 2:

```
students@students-HP-280-G3-MT:~/Desktop/DC EXP4$ python3 client.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: 1
Enter the item name: Sugar
Enter the quantity to add: 100
Product added with ID 2

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: Salt, Quantity: 400
Product ID: 2, Name: Sugar, Quantity: 100

Choose an action:
```

```
Product ID: 1, Name: Salt, Quantity: 400
Product ID: 2, Name: Sugar, Quantity: 100

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: Salt, Quantity: 400

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

Client session ended.
students@students-HP-280-G3-MT:~/Desktop/DC EXP4$
```

Conclusion:

Multi-threading in distributed systems using Python is a powerful technique for achieving parallelism, responsiveness, and efficient resource utilization. However, it requires careful consideration of concurrency control, fault tolerance, and load balancing. Implementing thread pools, robust error handling, and monitoring tools is crucial for success in building scalable and reliable distributed systems, ultimately leading to improved performance and a better user experience.