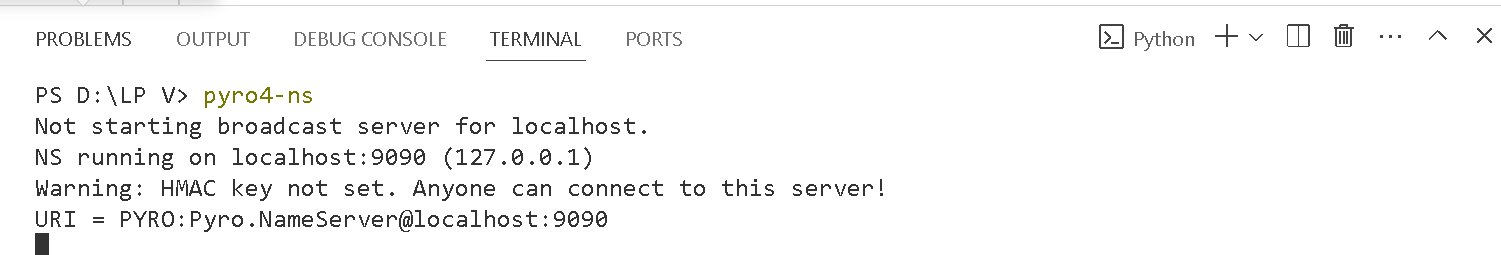
1. Implement multi-threaded client/server Process communication using RMI.

pyro4-ns  ---> tarminal command to run sever

Running server



MyServer.py

import Pyro4

@Pyro4.expose

class MyRemoteClass(object):

    def addition(self, x, y):

        return x + y

    def mult(self, x, y):

        return x \* y

def main():

    daemon = Pyro4.Daemon()

    ns = Pyro4.locateNS()

    uri = daemon.register(MyRemoteClass)

    ns.register("MyRemoteClass", uri)

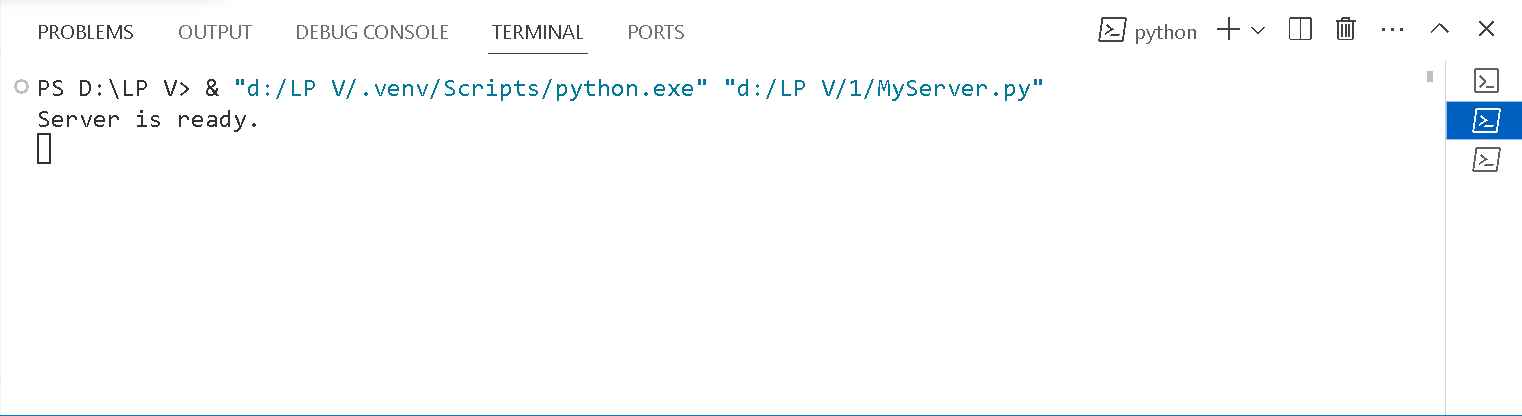
    print("Server is ready.")

    daemon.requestLoop()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output



MyClient.py

import Pyro4

def main():

    try:

        uri = "PYRONAME:MyRemoteClass"

        obj = Pyro4.Proxy(uri)

        a = int(input("Enter first number: "))

        b = int(input("Enter second number: "))

        print("The Addition is:", obj.addition(a, b))

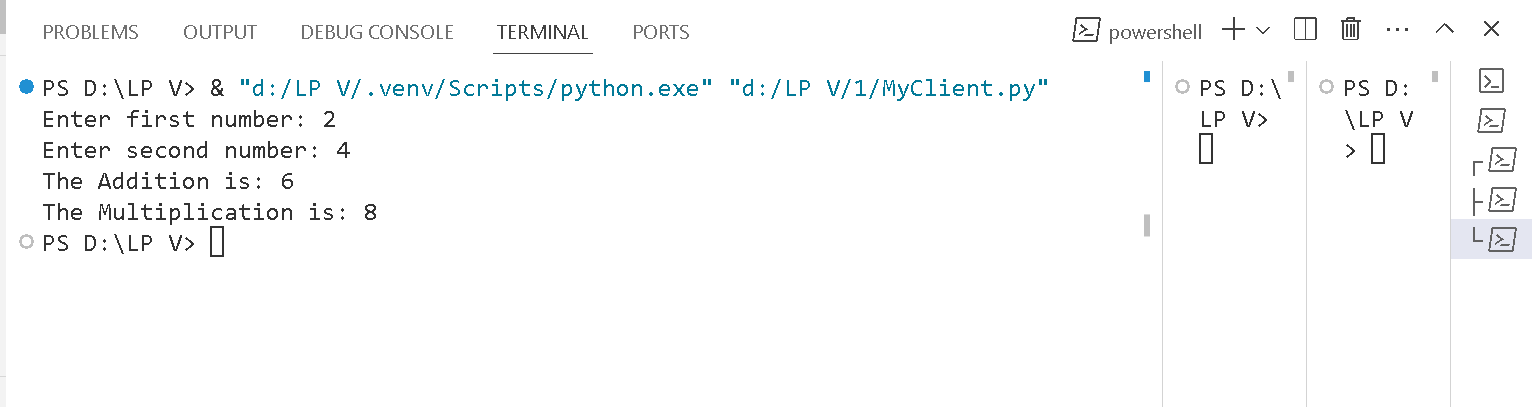
        print("The Multiplication is:", obj.mult(a, b))

    except Exception as e:

        print("Error:", e)

if \_\_name\_\_ == "\_\_main\_\_":

    main()



'''3. Develop a distributed system, to find sum of N elements in an array

by distributing N/n elements to n number of processors MPI or

OpenMP. Demonstrate by displaying the intermediate sums calculated

at different processors.'''

from mpi4py import MPI

def distribute\_array(arr, comm):

    rank = comm.Get\_rank()

    size = comm.Get\_size()

    chunk\_size = len(arr) // size

    start = rank \* chunk\_size

    end = start + chunk\_size if rank < size - 1 else len(arr)

    return arr[start:end]

def compute\_sum(arr):

    return sum(arr)

def main():

    comm = MPI.COMM\_WORLD

    rank = comm.Get\_rank()

    size = comm.Get\_size()

    # Define the array to be summed (Assuming same array on all processors)

    array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Distribute the array among processors

    local\_array = distribute\_array(array, comm)

    # Compute local sum

    local\_sum = compute\_sum(local\_array)

    # Gather all local sums on root process (rank 0)

    all\_sums = comm.gather(local\_sum, root=0)

    # Display intermediate sums calculated at different processors

    print("Processor", rank, "computed local sum:", local\_sum)

    # Root process combines sums to get the final result

    if rank == 0:

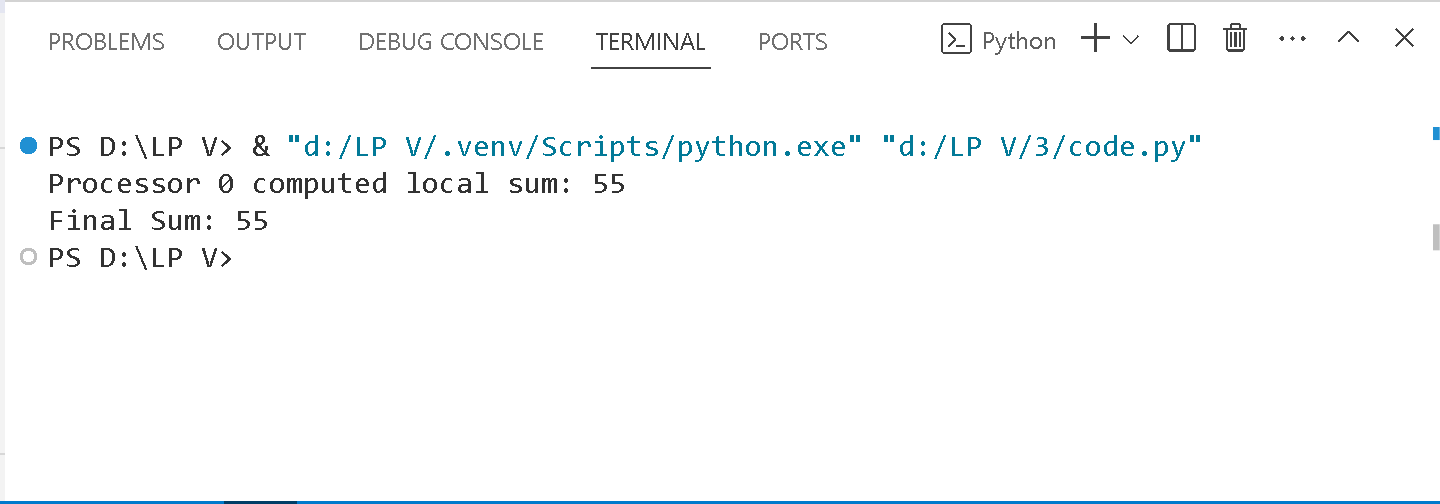
        final\_sum = sum(all\_sums)

        print("Final Sum:", final\_sum)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output



4. Implement Berkeley algorithm for clock synchronization.

import time

import random

# Function to calculate the clock offset

def calculate\_offset(remotes):

    local\_time = time.time()

    offsets = [remote - local\_time for remote in remotes]

    return sum(offsets) / len(offsets)

# Function to synchronize clocks using the Berkeley algorithm

def synchronize\_clocks():

    num\_peers = int(input("Enter the number of peers: "))

    local\_time = time.time()

    # Simulate remote clocks with random offsets

    remote\_times = [local\_time + random.uniform(-1, 1) for \_ in range(num\_peers)]

    print("Local time:", local\_time)

    print("Remote times:", remote\_times)

    # Calculate the clock offset

    offset = calculate\_offset(remote\_times)

    # Adjust local clock

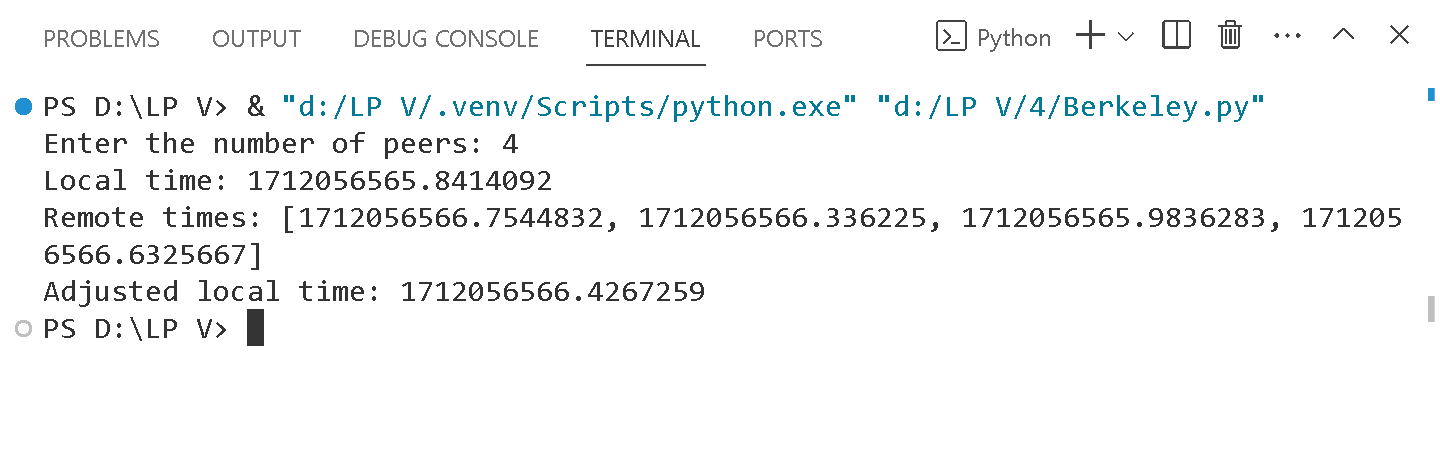
    adjusted\_time = local\_time + offset

    print("Adjusted local time:", adjusted\_time)

# Execute the clock synchronization

synchronize\_clocks()

Output



Implement token ring based mutual exclusion algorithm.

class TokenRing:

    def \_\_init\_\_(self, num\_nodes):

        self.num\_nodes = num\_nodes

        self.token = 0

    def send\_data(self, sender, receiver, data):

        print("Token passing:", end="")

        for i in range(self.token, sender):

            print(f" {i % self.num\_nodes}->", end="")

        print(f" {sender}")

        print(f"Sender {sender} sending data: {data}")

        for i in range(sender + 1, receiver):

            print(f"Data {data} forwarded by {i}")

        print(f"Receiver {receiver} received data: {data}\n")

        self.token = sender

if \_\_name\_\_ == "\_\_main\_\_":

    num\_nodes = int(input("Enter the number of nodes: "))

    token\_ring = TokenRing(num\_nodes)

    while True:

        sender = int(input("Enter sender: "))

        receiver = int(input("Enter receiver: "))

        data = input("Enter data: ")

        token\_ring.send\_data(sender, receiver, data)

        send\_again = input("Do you want to send again? (yes/no): ")

        if send\_again.lower() != "yes":

            break

Output

6. Implement Bully and Ring algorithm for leader election.

class Node:

    def \_\_init\_\_(self, node\_id):

        self.id = node\_id

        self.is\_coordinator = False

    def initiate\_election(self, nodes):

        for node in nodes:

            if node.id > self.id:

                print(f"Node {self.id} sends election message to Node {node.id}")

                node.start\_election(nodes)

        self.is\_coordinator = True

        print(f"Node {self.id} becomes the coordinator.")

    def start\_election(self, nodes):

        for node in nodes:

            if node.id > self.id:

                print(f"Node {self.id} sends election message to Node {node.id}")

                node.start\_election(nodes)

        self.is\_coordinator = True

        print(f"Node {self.id} becomes the coordinator.")

if \_\_name\_\_ == "\_\_main\_\_":

    # Create nodes

    nodes = [Node(i) for i in range(1, 6)]

    # Simulate Bully Algorithm

    print("Bully Algorithm:")

    # Node with highest ID starts the election

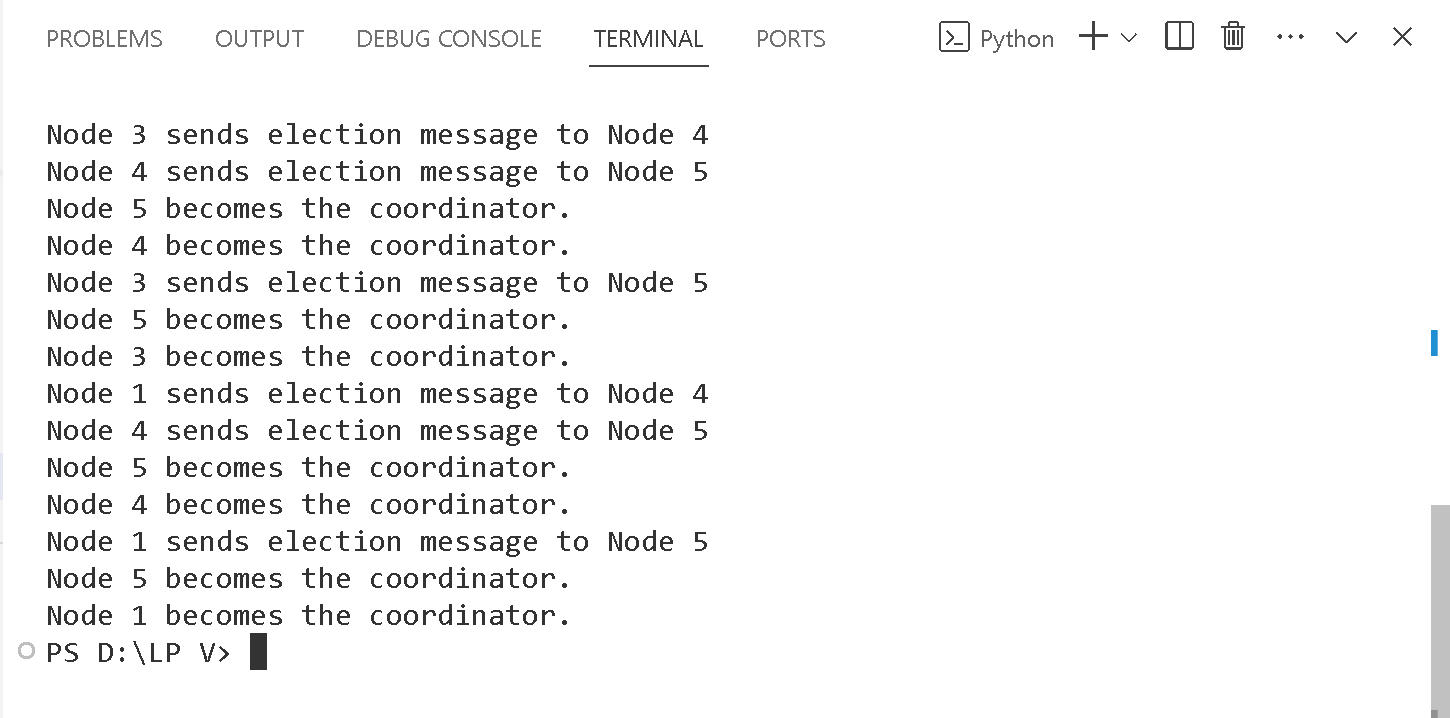
    nodes[-1].initiate\_election(nodes)

    # Simulate Ring Algorithm

    print("\nRing Algorithm:")

    # Node with lowest ID starts the election

    nodes[0].start\_election(nodes)

Output

7. Create a simple web service and write any distributed application to

App.py

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

@app.route('/add', methods=['POST'])

def add():

    data = request.get\_json()

    a = data['a']

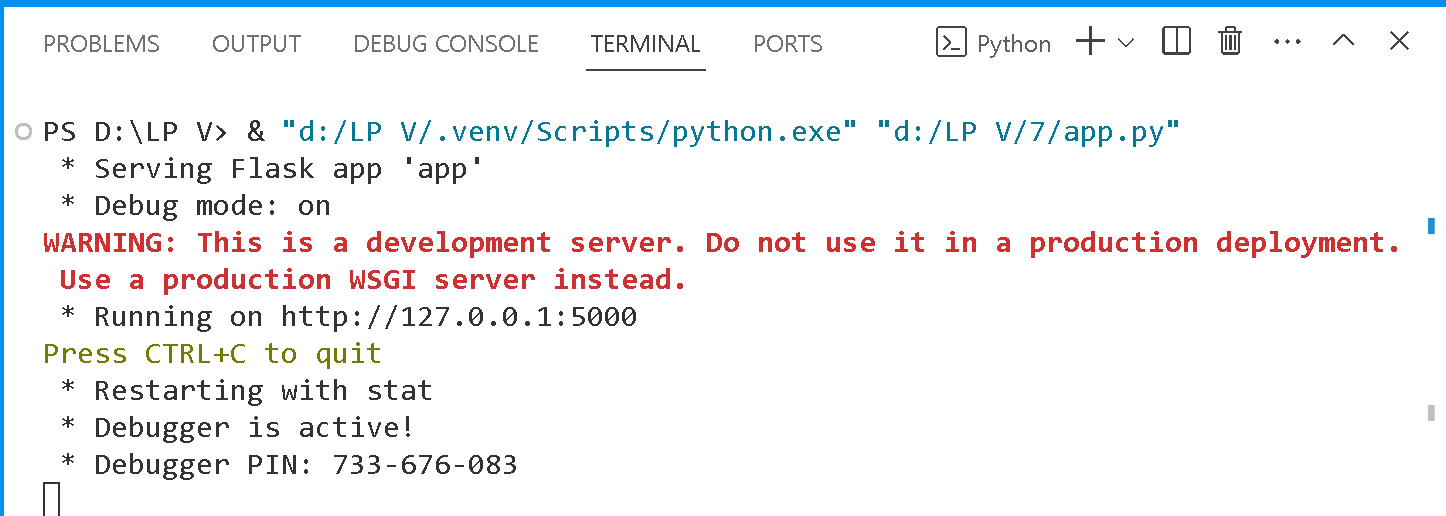
    b = data['b']

    result = a + b

    return jsonify({'result': result})

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

output

Client.py

import requests

data = {

    'a': 10,

    'b': 20

}

response = requests.post('http://localhost:5000/add', json=data)

result = response.json()

print("Result of addition:", result['result'])

Output

