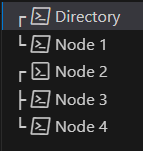
Programming Assignment 3

Bhavesh Naidu kulluru

Structure:  
  


CODE:   
  
directory\_server.py

import socket

import threading

import time

PORT = 5000

nodes = {}

leader\_id = -1

election\_in\_progress = False

# Broadcast message to all nodes

def broadcast(msg):

    for node\_id, node\_address in nodes.items():

        try:

            with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

                s.connect((node\_address[0], node\_address[1]))

                s.sendall(msg.encode())

                print(f"Directory Server: Broadcasted to Node {node\_id}: {msg}")

        except Exception as e:

            print(f"Failed to send message to {node\_id}: {e}")

# Handle incoming client connections (nodes)

def handle\_client(client\_socket, client\_address):

    global leader\_id, election\_in\_progress

    try:

        msg = client\_socket.recv(1024).decode()

        print(f"Directory Server: Received from {client\_address}: {msg}")

        # Node Join

        if msg.startswith("JOIN"):

            node\_id = int(msg.split()[1])

            nodes[node\_id] = client\_address

            print(f"Directory Server: Node {node\_id} joined. Active nodes: {list(nodes.keys())}")

            broadcast(f"New Node {node\_id} has joined the network.")

            if leader\_id == -1 or node\_id > leader\_id:

                leader\_id = node\_id

                broadcast(f"LEADER {leader\_id}")

                print(f"Directory Server: New leader elected: Node {leader\_id}")

            client\_socket.send("JOIN\_ACK".encode())

        # Node Leave

        elif msg.startswith("LEAVE"):

            node\_id = int(msg.split()[1])

            if node\_id in nodes:

                del nodes[node\_id]

                print(f"Directory Server: Node {node\_id} left. Active nodes: {list(nodes.keys())}")

                broadcast(f"Node {node\_id} has left the network.")

                if node\_id == leader\_id:

                    initiate\_election()

        # Heartbeat from Node

        elif msg.startswith("HEARTBEAT"):

            node\_id = int(msg.split()[1])

            print(f"Directory Server: Heartbeat received from Node {node\_id}")

            client\_socket.send("HEARTBEAT\_ACK".encode())

        # Election Process Triggered

        elif msg.startswith("ELECTION"):

            node\_id = int(msg.split()[1])

            print(f"Directory Server: Election message received from Node {node\_id}")

            if not election\_in\_progress:

                election\_in\_progress = True

                initiate\_election()

    except Exception as e:

        print(f"Error with client {client\_address}: {e}")

    finally:

        client\_socket.close()

# Listen for client connections (nodes)

def listen\_for\_clients():

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server\_socket.bind(("", PORT))

    server\_socket.listen(5)

    print(f"Directory Server running on port {PORT}")

    while True:

        client\_socket, client\_address = server\_socket.accept()

        print(f"Directory Server: New connection from {client\_address}")

        threading.Thread(target=handle\_client, args=(client\_socket, client\_address), daemon=True).start()

# Initiate Leader Election (Bully Algorithm)

def initiate\_election():

    global leader\_id, election\_in\_progress

    print("Directory Server: Leader election triggered...")

    if nodes:

        election\_in\_progress = True

        higher\_nodes = [node\_id for node\_id in nodes.keys() if node\_id > leader\_id]

        if higher\_nodes:

            for node in higher\_nodes:

                send\_election\_message(node)

        else:

            leader\_id = max(nodes.keys())

            broadcast(f"LEADER {leader\_id}")

            print(f"Directory Server: New leader elected: Node {leader\_id}")

            election\_in\_progress = False

# Send Election Message to Node

def send\_election\_message(node\_id):

    try:

        with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

            s.connect(nodes[node\_id])

            s.sendall(f"ELECTION {leader\_id}".encode())

            print(f"Directory Server: Sent ELECTION to Node {node\_id}")

    except Exception as e:

        print(f"Failed to send ELECTION to Node {node\_id}: {e}")

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

    listen\_for\_clients()

node.py

import socket

import threading

import time

class Node:

    def \_\_init\_\_(self, node\_id, ip\_address='127.0.0.1', port=5000):

        self.node\_id = node\_id

        self.ip\_address = ip\_address

        self.port = 5000 + node\_id

        self.is\_leader = False

        self.server\_address = (ip\_address, port)

    # Join the network by connecting to the directory server

    def join\_network(self):

        try:

            print(f"Node {self.node\_id}: Attempting to join network at {self.server\_address}")

            with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

                s.connect(self.server\_address)

                s.sendall(f"JOIN {self.node\_id}".encode())

                response = s.recv(1024).decode()

                if response == "JOIN\_ACK":

                    print(f"Node {self.node\_id} has successfully joined the network.")

        except Exception as e:

            print(f"Node {self.node\_id}: Error joining network: {e}")

    # Leave the network

    def leave\_network(self):

        try:

            with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

                s.connect(self.server\_address)

                s.sendall(f"LEAVE {self.node\_id}".encode())

                print(f"Node {self.node\_id} has left the network.")

        except Exception as e:

            print(f"Node {self.node\_id}: Error leaving network: {e}")

    # Send heartbeat to the directory server

    def send\_heartbeat(self):

        while True:

            time.sleep(5)

            try:

                with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

                    s.connect(self.server\_address)

                    s.sendall(f"HEARTBEAT {self.node\_id}".encode())

                    response = s.recv(1024).decode()

            except Exception as e:

                print(f"Node {self.node\_id}: Error sending heartbeat: {e}")

    # Receive messages from the directory server

    def receive\_messages(self):

        with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

            s.bind((self.ip\_address, self.port))

            s.listen(5)

            while True:

                client\_socket, client\_address = s.accept()

                message = client\_socket.recv(1024).decode()

                print(f"Node {self.node\_id}: Received message: {message}")

                if message.startswith("LEADER"):

                    new\_leader = int(message.split()[1])

                    print(f"Node {self.node\_id}: New leader elected: Node {new\_leader}")

                    self.is\_leader = (new\_leader == self.node\_id)

                elif message.startswith("New Node"):

                    print(f"Node {self.node\_id}: {message}")

                elif message.startswith("ELECTION"):

                    self.handle\_election(message.split()[1])

                client\_socket.close()

    # Handle the election message

    def handle\_election(self, sender\_id):

        print(f"Node {self.node\_id}: Handling election from Node {sender\_id}")

        if sender\_id < self.node\_id:

            self.respond\_to\_election(sender\_id)

    # Respond to an election

    def respond\_to\_election(self, sender\_id):

        try:

            with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

                s.connect(self.server\_address)

                s.sendall(f"OK {self.node\_id}".encode())

                print(f"Node {self.node\_id}: Responded with OK to Node {sender\_id}")

        except Exception as e:

            print(f"Node {self.node\_id}: Error responding to election: {e}")

    # Start the node processes

    def start(self):

        threading.Thread(target=self.send\_heartbeat, daemon=True).start()

        threading.Thread(target=self.receive\_messages, daemon=True).start()

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

    node\_id = int(input("Enter Node ID: "))

    node = Node(node\_id)

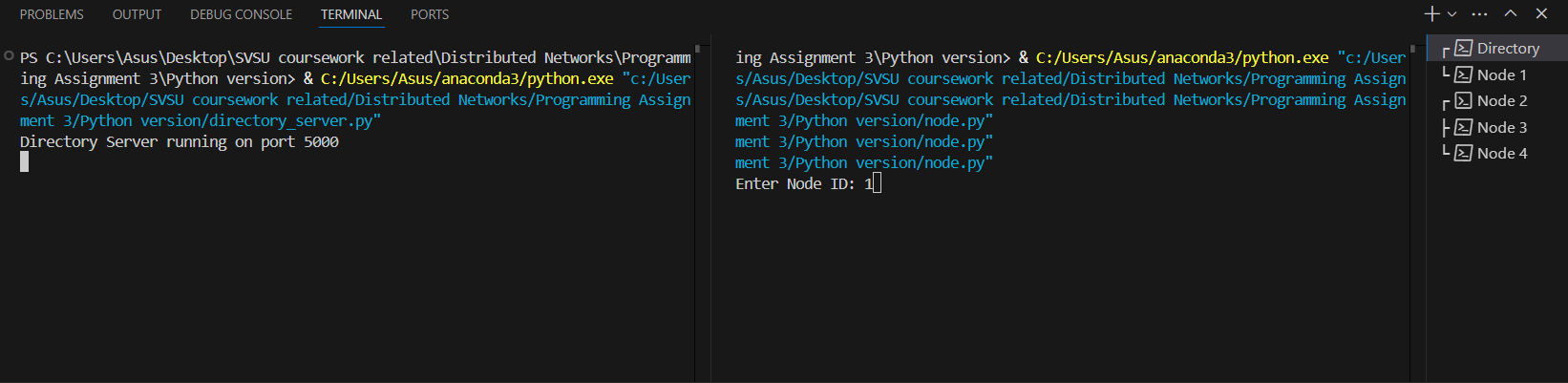
    node.join\_network()

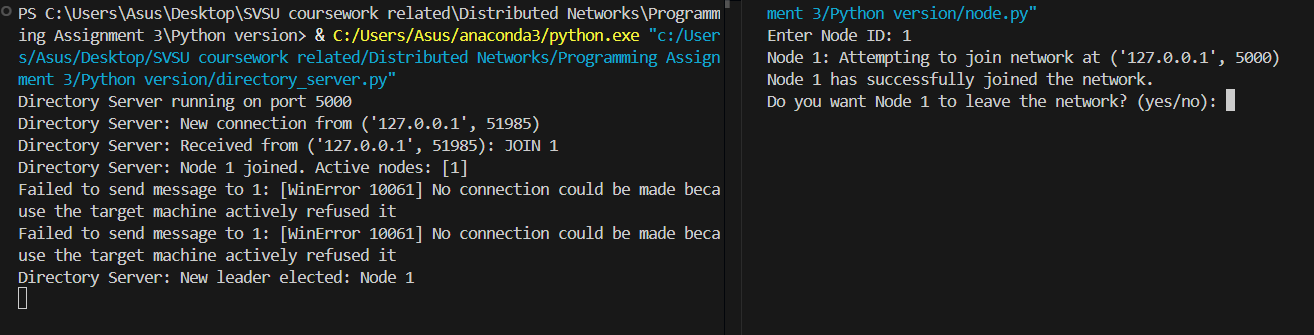
    node.start()

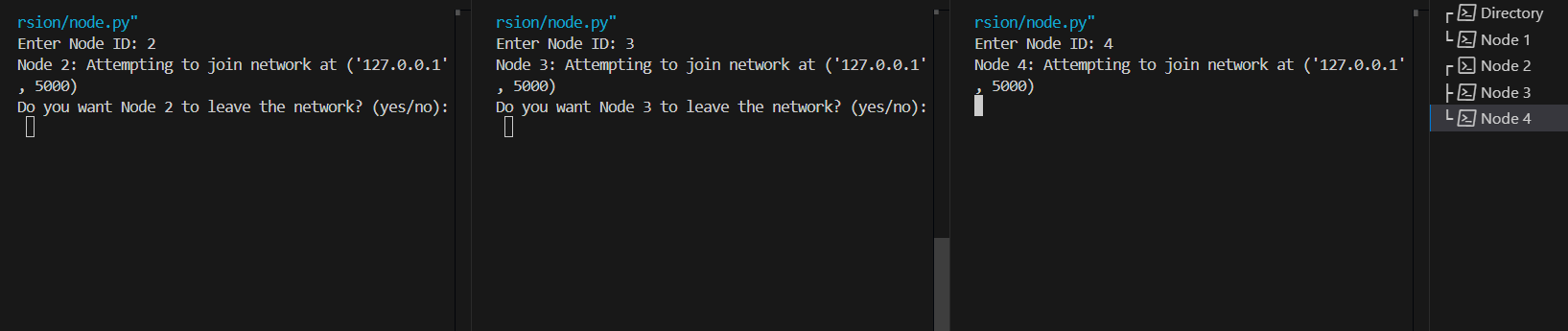
    leave = input(f"Do you want Node {node\_id} to leave the network? (yes/no): ")

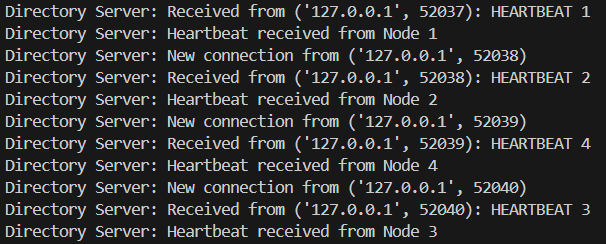
    if leave.lower() == 'yes':

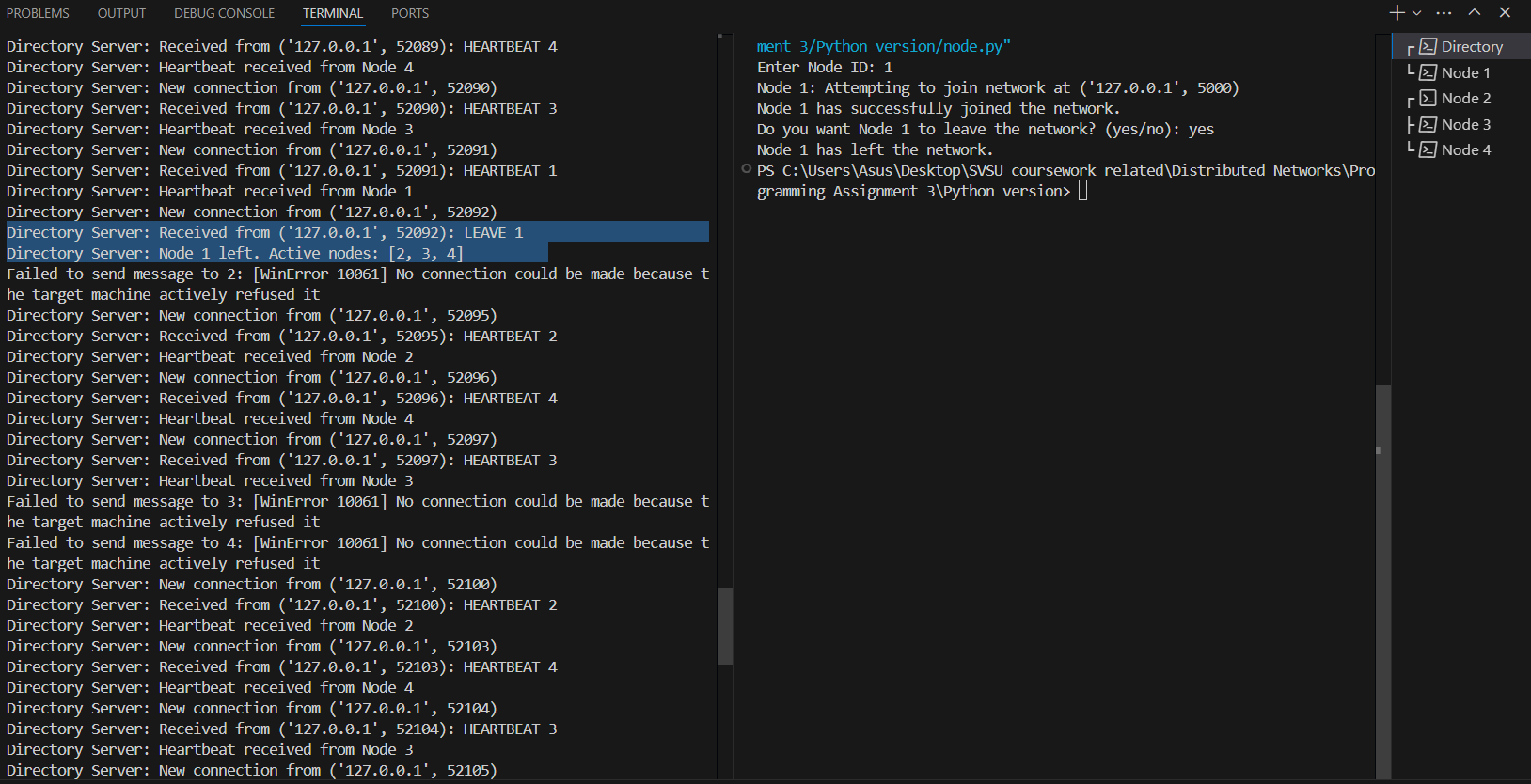
        node.leave\_network()

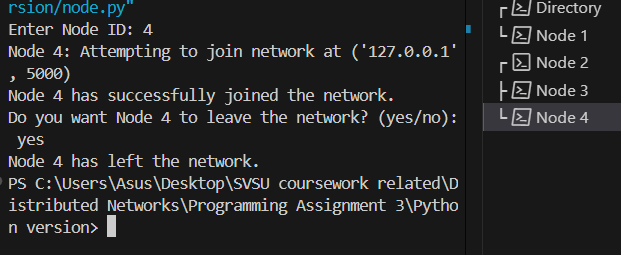
OUTPUTS:  
  


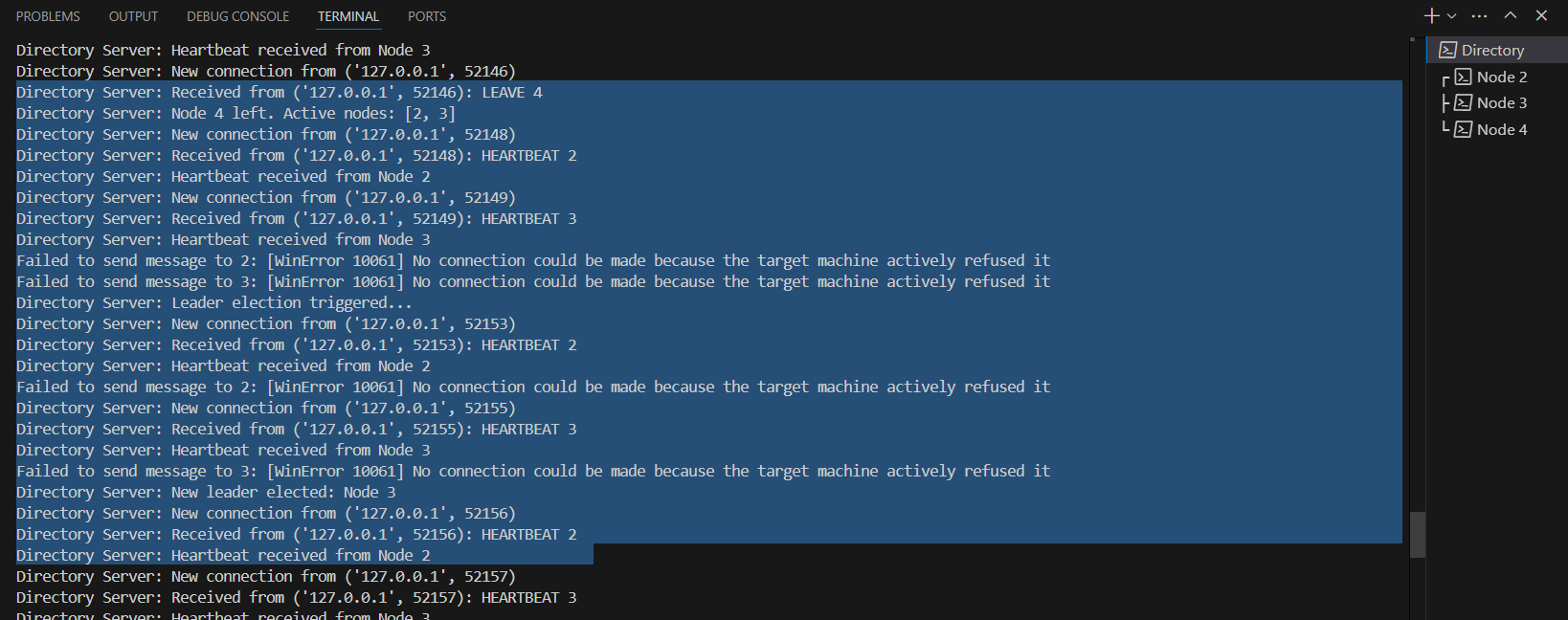












Thank YOU !!!