Exp 2 – rmi

Client

import java.rmi.Naming;

import java.util.Scanner;

public class RMI\_Client {

    public static *void* main(String[] *args*) {

        Scanner sc = null;

        try {

            RMI\_interface remoteObject = (RMI\_interface) Naming.lookup("rmi://localhost:1878/hello");

            sc = new Scanner(System.in);

            System.out.print(" Enter a number to calculate its square root : ");

*double* number = sc.nextDouble();

*double* result = remoteObject.calculateSquareRoot(number);

            System.out.println("Square root of " + number + " is : " + result);

        } catch (Exception e) {

            System.out.println("The RMI APP is Not running...");

            e.printStackTrace();

        } finally {

            if (sc != null) {

                sc.close();

            }

        }

    }

}

Server

import java.nio.channels.AlreadyBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

public class RMI\_Server extends UnicastRemoteObject implements RMI\_interface {

    protected RMI\_Server() throws RemoteException {

        super();

    }

    public static *void* main(String[] *args*)throws RemoteException, AlreadyBoundException {

        try {

            Registry registry = LocateRegistry.createRegistry(1878);

            registry.bind("hello", new RMI\_Server());

            System.out.println("The RMI\_Server is running and ready...");

        }

        catch (Exception e) {

            System.out.println("The RMI\_Server is not running...");

        }

    }

    @*Override*

    public *double* calculateSquareRoot(*double* *number*) throws RemoteException {

*double* result = Math.sqrt(*number*);

        System.out.println("Square Root of "+ *number*+ " is : "+result);

        return result;

    }

}

Interface

import java.rmi.Remote;

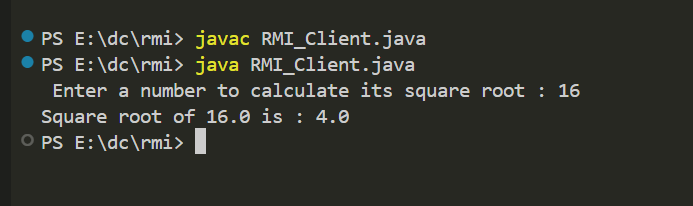
import java.rmi.RemoteException;

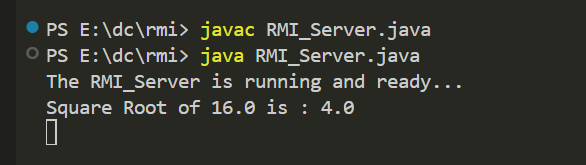
public interface RMI\_interface extends *Remote* {

*double* calculateSquareRoot(*double* *number*) throws RemoteException;

}

Output





Experiment 3 – Inter-process Communication

client

# client code

import socket

HOST = '127.0.0.1'

PORT = 12345

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

    client\_socket.connect((HOST, PORT))

    while True:

        message = input("Enter message to send to server: ")

        if message.lower() == 'exit':

            break

        client\_socket.sendall(message.encode())

        data = client\_socket.recv(1024)

        print("Received from server:", data.decode())

server

# server code

import socket

HOST = '127.0.0.1'

PORT = 12345

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

    server\_socket.bind((HOST, PORT))

    server\_socket.listen()

    print("Server listening on", (HOST, PORT))

    while True:

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

        print("Connected by", client\_address)

        with client\_socket:

            while True:

                data = client\_socket.recv(1024)

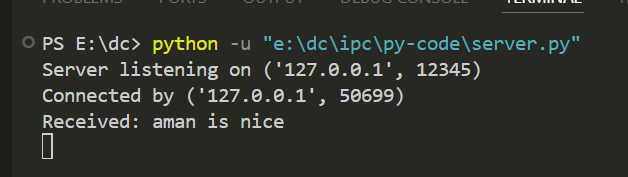
                if not data:

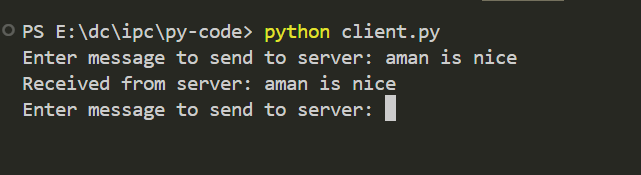
                    break

                print("Received:", data.decode())

                client\_socket.sendall(data)

code





Exp 4 group communication

Client

# Run server and client seperately copy the below server code

# First run server

# client code

import socket

import struct  # Import the struct module for packing TTL

# Define the multicast address and port

MULTICAST\_GROUP = '224.3.29.71'

MULTICAST\_PORT = 10000

# Create a socket

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Set the time-to-live (TTL) for multicast packets

ttl = struct.pack('b', 1)

sock.setsockopt(socket.IPPROTO\_IP, socket.IP\_MULTICAST\_TTL, ttl)

# Prompt the user to enter messages and send them as multicast

while True:

    message = input("Enter message to send (type 'exit' to quit): ")

    if message.lower() == 'exit':

        break

    sock.sendto(message.encode(), (MULTICAST\_GROUP, MULTICAST\_PORT))

# Close the socket

sock.close()

server

# server code

import socket

import struct

import threading  # Import the threading module

# Define the multicast address and port

MULTICAST\_GROUP = '224.3.29.71'

MULTICAST\_PORT = 10000

# Create a socket

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Bind the socket to the server address

server\_address = ('', MULTICAST\_PORT)

sock.bind(server\_address)

# Join the multicast group

group = socket.inet\_aton(MULTICAST\_GROUP)

mreq = struct.pack('4sL', group, socket.INADDR\_ANY)

sock.setsockopt(socket.IPPROTO\_IP, socket.IP\_ADD\_MEMBERSHIP, mreq)

# Function to receive messages

*def* receive\_messages():

    while True:

        data, address = sock.recvfrom(1024)

        print("Received message from {}: {}".format(address, data.decode()))

# Start a thread to receive messages

receive\_thread = threading.Thread(*target*=receive\_messages)

receive\_thread.start()

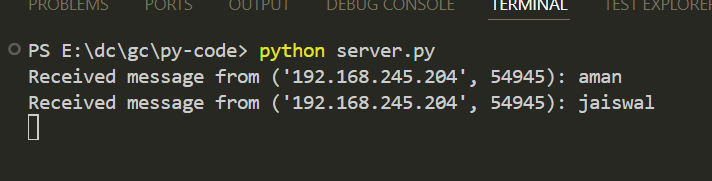
# Wait for the thread to finish

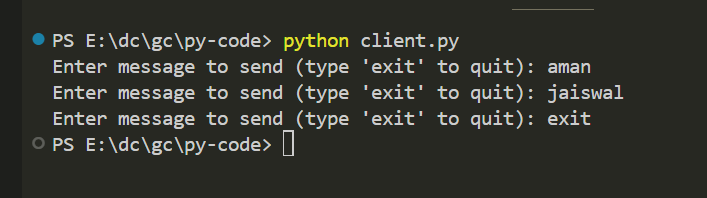
receive\_thread.join()

# Close the socket

sock.close()

output





Election-Algo-ring-algo

Code

*class* Pro:

*def* \_\_init\_\_(*self*, *id*):

        self.id = id

        self.act = True

*class* GFG:

*def* \_\_init\_\_(*self*):

        self.TotalProcess = 0

        self.process = []

*def* initialiseGFG(*self*):

        print("No of processes 5")

        self.TotalProcess = 5

        self.process = [Pro(i) for i in range(self.TotalProcess)]

*def* Election(*self*):

        max\_id\_process\_index = self.FetchMaximumActive()

        print("Process no " + *str*(self.process[max\_id\_process\_index].id) + " fails")

        self.process[max\_id\_process\_index].act = False

        # Initiating election by the highest ID process among active processes

        initialized\_process = self.FetchMaximumActive()

        print("Election Initiated by " + *str*(self.process[initialized\_process].id))

        # If there are no active processes, end the election

        if initialized\_process == -1:

            print("No active processes. End of Election.")

            return

        for newer in range(initialized\_process + 1, initialized\_process + self.TotalProcess):

            newer %= self.TotalProcess

            if self.process[newer].act:

                print("Process " + *str*(self.process[initialized\_process].id) + " pass Election(" + *str*(self.process[initialized\_process].id) + ") to " + *str*(self.process[newer].id))

                if self.process[newer].id > self.process[initialized\_process].id:

                    print("Process " + *str*(self.process[newer].id) + " responds 'OK'")

                else:

                    print("Process " + *str*(self.process[newer].id) + " doesn't respond")

        # Find the new coordinator among the active processes

        coord = self.FetchMaximumActive()

        if coord != -1:

            print("Process " + *str*(self.process[coord].id) + " becomes coordinator")

            old = coord

            newer = (old + 1) % self.TotalProcess

            while True:

                if self.process[newer].act:

                    print("Process " + *str*(self.process[old].id) + " pass Coordinator(" + *str*(coord) + ") message to process " + *str*(self.process[newer].id))

                    old = newer

                newer = (newer + 1) % self.TotalProcess

                if newer == coord:

                    print("End Of Election ")

                    break

        else:

            print("No active processes. End of Election.")

*def* FetchMaximumActive(*self*):

        max\_id = -1

        max\_id\_process\_index = -1

        for i in range(self.TotalProcess):

            if self.process[i].act and self.process[i].id > max\_id:

                max\_id = self.process[i].id

                max\_id\_process\_index = i

        return max\_id\_process\_index

*def* main():

    obj = GFG()

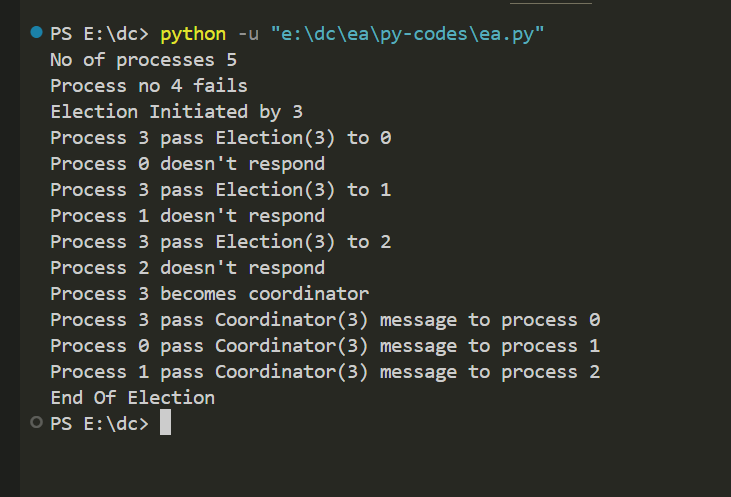
    obj.initialiseGFG()

    obj.Election()

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

    main()

output



Exp 6 – clock synchronization - berkeley\_algorithm

Code

from datetime import datetime, timedelta

*def* berkeley\_algorithm(*nodes*):

    # Calculate the average time (converted to timestamps)

    average\_time = sum(node['time'].timestamp() for node in nodes) / len(nodes)

    # Calculate the time difference for each node

    for node in nodes:

        node['offset'] = average\_time - node['time'].timestamp()

        node['synchronized\_time'] = node['time'] + timedelta(*seconds*=node['offset'])

*def* synchronize\_clocks(*nodes*):

    for node in nodes:

        # Synchronize the clock for each node

        node['synchronized\_time'] = node['time'] + timedelta(*seconds*=node['offset'])

*def* print\_node\_times(*nodes*):

    for node in nodes:

        print(*f*"Node {node['id']} - Local Time: {node['time']}, Synchronized Time: {node.get('synchronized\_time', 'Not synchronized')}")

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

    # Example with three nodes

    nodes = [

        {'id': 1, 'time': datetime.now()},

        {'id': 2, 'time': datetime.now() + timedelta(*seconds*=5)},

        {'id': 3, 'time': datetime.now() - timedelta(*seconds*=3)}

    ]

    print("Original Node Times:")

    print\_node\_times(nodes)

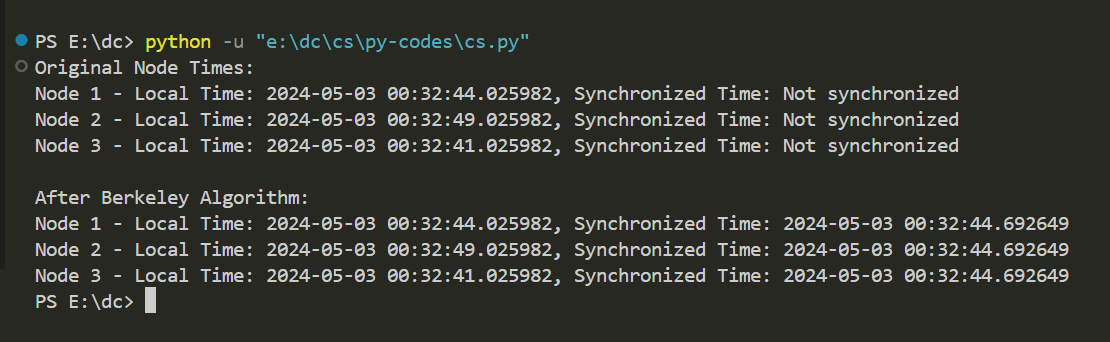
    berkeley\_algorithm(nodes)

    synchronize\_clocks(nodes)

    print("\nAfter Berkeley Algorithm:")

    print\_node\_times(nodes)

output



Exp 7 token-base-Suzuki-Kasami-Algo

Code

class CriticalSection implements Runnable {

    private static *int* counter = 0;

    private final *int* id;

    public CriticalSection(*int* *id*) {

        this.id = *id*;

    }

    @*Override*

    public *void* run() {

        while (counter < 10) {

            synchronized (CriticalSection.class) {

                if (counter % 5 == id) {

                    System.out.println("Node " + id + " is in critical section");

                    counter++;

                }

            }

        }

    }

}

public class Suzuki\_Kasami {

    public static *void* main(String[] *args*) {

        Thread[] threads = new Thread[5];

        for (*int* i = 0; i < 5; i++) {

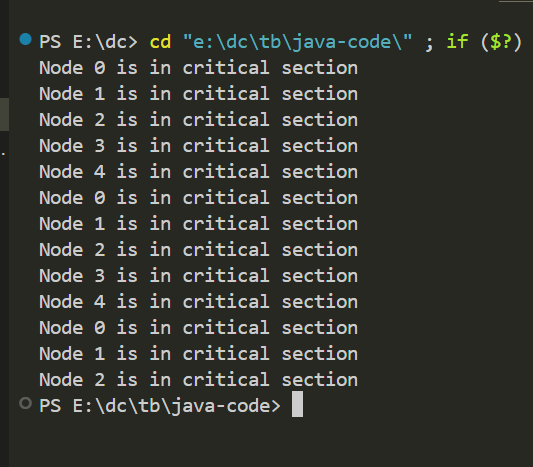
            threads[i] = new Thread(new CriticalSection(i));

            threads[i].start();

        }

    }

}



non token based – Ricart Agrawala Algorithm.

Code - python

*class* Message:

*def* \_\_init\_\_(*self*, *messageType*, *timestamp*, *siteId*):

        self.messageType = messageType

        self.timestamp = timestamp

        self.siteId = siteId

*class* Site:

*def* \_\_init\_\_(*self*, *siteId*):

        self.siteId = siteId

        self.requesting = False

        self.executing = False

        self.timestamp = 0

        self.deferredQueue = []

*def* requestCriticalSection(*self*, *sites*):

        self.requesting = True

        self.timestamp += 1

        for site in sites:

            if site.siteId != self.siteId:

                requestMessage = Message("REQUEST", self.timestamp, self.siteId)

                self.sendMessage(requestMessage, site)

        self.waitForReplies(sites)

*def* sendMessage(*self*, *message*, *destination*):

        print(*f*"Site {self.siteId} sends {message.messageType} message to Site {destination.siteId}")

        destination.receiveMessage(message, self)

*def* receiveMessage(*self*, *message*, *sender*):

        print(*f*"Site {self.siteId} receives {message.messageType} message from Site {sender.siteId}")

        if message.messageType == "REQUEST":

            if not self.requesting and not self.executing:

                self.sendMessage(Message("REPLY", 0, self.siteId), sender)

            elif self.requesting and message.timestamp < self.timestamp:

                self.deferredQueue.append(message)

        elif message.messageType == "REPLY":

            if self.requesting:

                self.deferredQueue = [m for m in self.deferredQueue if m.siteId != sender.siteId]

                if not self.deferredQueue:

                    self.executing = True

                    print(*f*"Site {self.siteId} enters critical section.")

*def* waitForReplies(*self*, *sites*):

        repliesExpected = len(sites) - 1

        repliesReceived = 0

        while repliesReceived < repliesExpected:

            pass  # Wait for replies

*def* releaseCriticalSection(*self*, *sites*):

        self.requesting = False

        self.executing = False

        for site in sites:

            if site.siteId != self.siteId:

                for message in self.deferredQueue:

                    self.sendMessage(Message("REPLY", 0, self.siteId), site)

        self.deferredQueue.clear()

        print(*f*"Site {self.siteId} releases critical section.")

*def* main():

    numberOfSites = *int*(input("Enter the number of sites: "))

    sites = [Site(i + 1) for i in range(numberOfSites)]

    for site in sites:

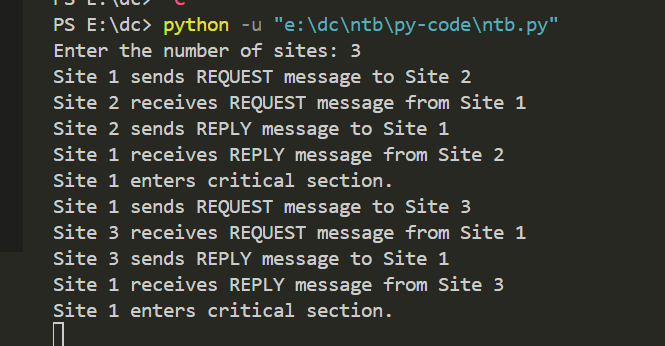
        site.requestCriticalSection(sites)

        site.releaseCriticalSection(sites)

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

    main()

output



Round robin - load balancer

Code - python

*class* RoundRobinLoadBalancer:

*def* \_\_init\_\_(*self*, *numServers*):

        self.numServers = numServers

        self.servers = [[] for \_ in range(numServers)]

*def* addProcesses(*self*, *processes*):

        currentIndex = 0

        for process in processes:

            self.servers[currentIndex].append(process)

            currentIndex = (currentIndex + 1) % self.numServers  # Round robin distribution

*def* printProcesses(*self*):

        for i, server in enumerate(self.servers):

            print(*f*"Server {i + 1} Processes: {server}")

*def* main():

    # Initial processes in the servers

    initialProcesses = [1, 2, 3, 4, 5, 6, 7]

    # Number of servers

    numServers = 4

    loadBalancer = RoundRobinLoadBalancer(numServers)

    print("Processes before balancing:")

    print(\*initialProcesses)

    loadBalancer.addProcesses(initialProcesses)

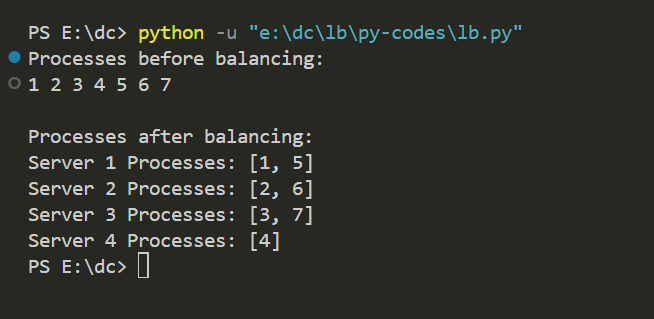
    print("\nProcesses after balancing:")

    loadBalancer.printProcesses()

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

    main()

output



Code - java

import java.util.ArrayList;

import java.util.List;

class Server {

    private *int* id;

    private *int* load;

    public Server(*int* *id*) {

        this.id = *id*;

        this.load = 0;

    }

    public *int* getId() {

        return id;

    }

    public *int* getLoad() {

        return load;

    }

    public *void* incrementLoad() {

        load++;

    }

}

public class RoundRobinLoadBalancer {

    private List<Server> servers;

    private *int* currentIndex;

    public RoundRobinLoadBalancer() {

        servers = new ArrayList<>();

        currentIndex = 0;

    }

    public *void* addServer(Server *server*) {

        servers.add(*server*);

    }

    public Server getNextServer() {

        Server nextServer = servers.get(currentIndex);

        currentIndex = (currentIndex + 1) % servers.size(); // Move to the next server in a circular manner

        return nextServer;

    }

    public static *void* main(String[] *args*) {

        RoundRobinLoadBalancer loadBalancer = new RoundRobinLoadBalancer();

        // Add some servers

        loadBalancer.addServer(new Server(1));

        loadBalancer.addServer(new Server(2));

        loadBalancer.addServer(new Server(3));

        // Simulate requests

        for (*int* i = 0; i < 10; i++) {

            Server server = loadBalancer.getNextServer();

            server.incrementLoad();

            System.out.println("Request assigned to Server " + server.getId());

        }

        // Print server loads

        System.out.println("\nServer Loads:");

        for (Server server : loadBalancer.servers) {

            System.out.println("Server " + server.getId() + ": " + server.getLoad());

        }

    }

}

Output

