Exp 2 – rmi

Client

```java
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();
            }
        }
    }
}
```

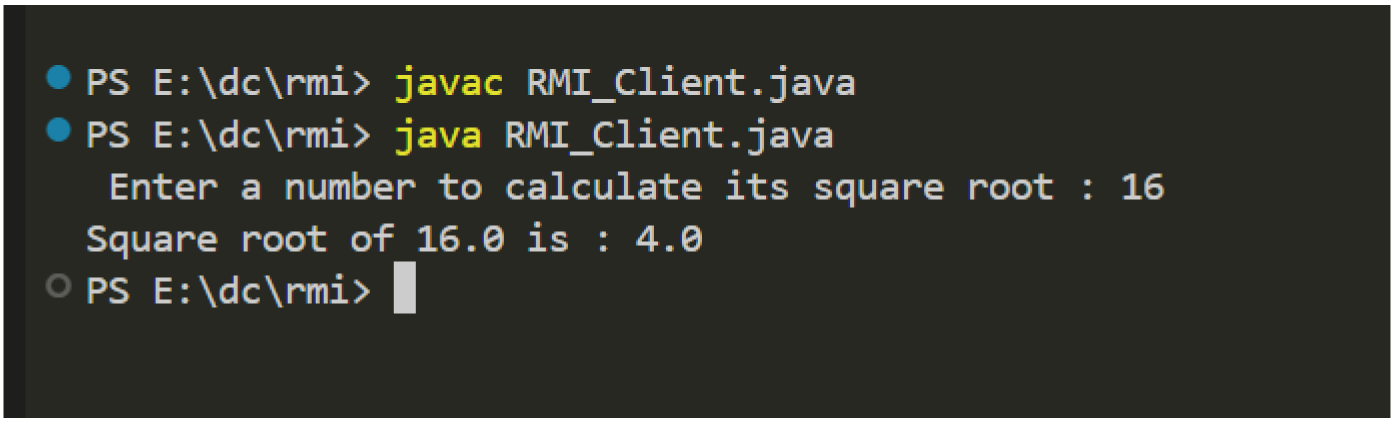
Server

```java
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;
    }
}
```

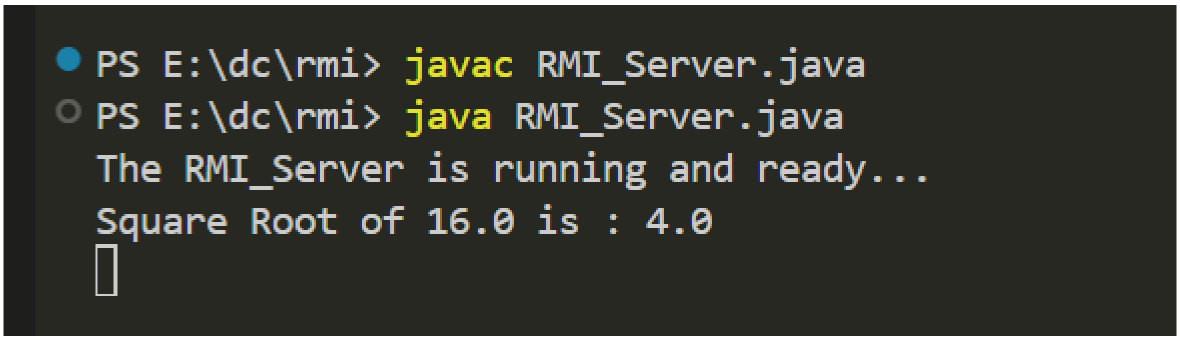
Interface

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMI_interface extends Remote {
    double calculateSquareRoot(double number) throws RemoteException;
}
```

Output



```
PS E:\dc\rmi> javac RMI_Client.java
PS E:\dc\rmi> java RMI_Client.java
 Enter a number to calculate its square root : 16
Square root of 16.0 is : 4.0
PS E:\dc\rmi>
```



```
PS E:\dc\rmi> javac RMI_Server.java
PS E:\dc\rmi> java RMI_Server.java
The RMI_Server is running and ready...
Square Root of 16.0 is : 4.0
```

Experiment 3 – Inter-process Communication

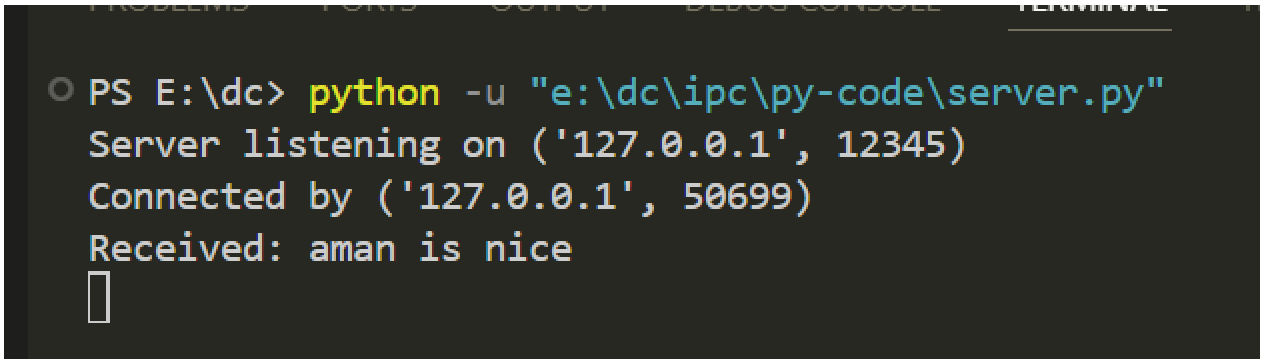
client

```python
# 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())
```

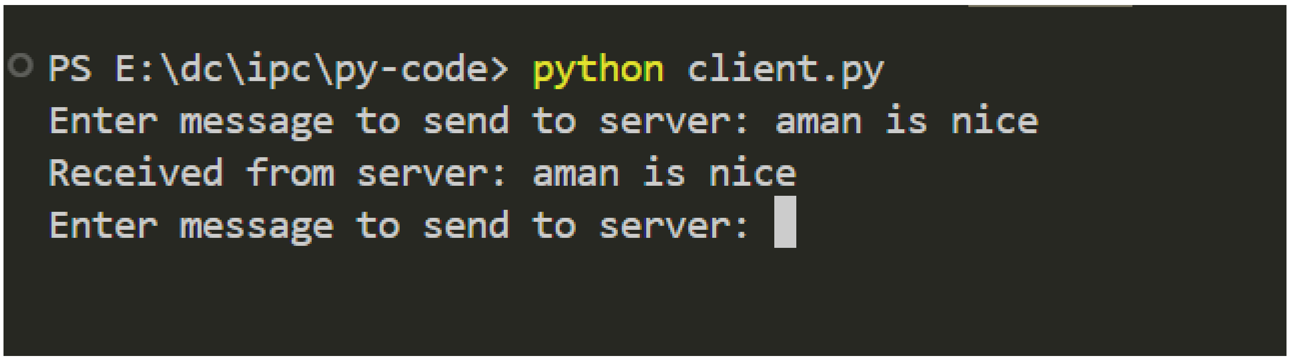
server

```python

# 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



```
PS E:\dc> python -u "e:\dc\ipc\py-code\server.py"
Server listening on ('127.0.0.1', 12345)
Connected by ('127.0.0.1', 50699)
Received: aman is nice
```



```
PS E:\dc\ipc\py-code> python client.py
Enter message to send to server: aman is nice
Received from server: aman is nice
Enter message to send to server:
```

Exp 4 group communication

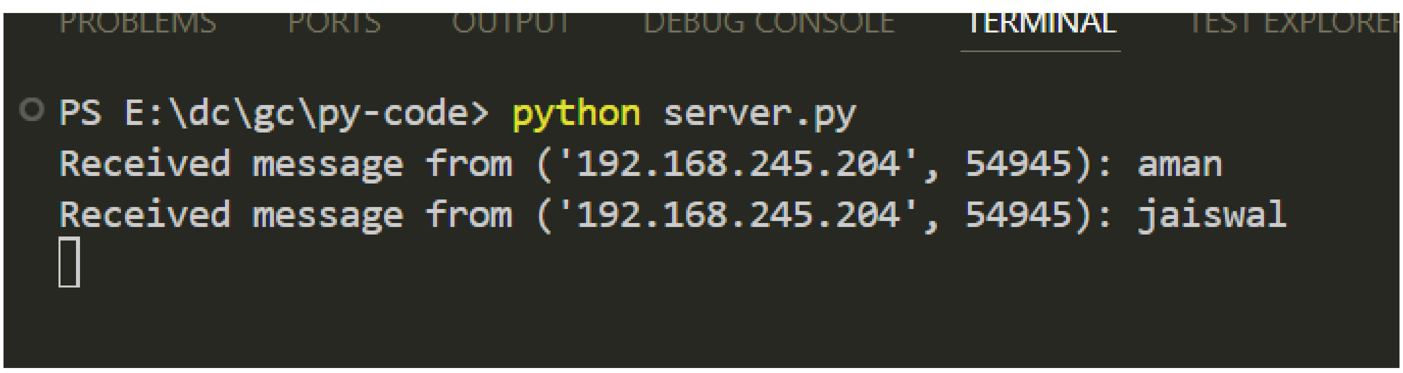
Client

```
# 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()
```

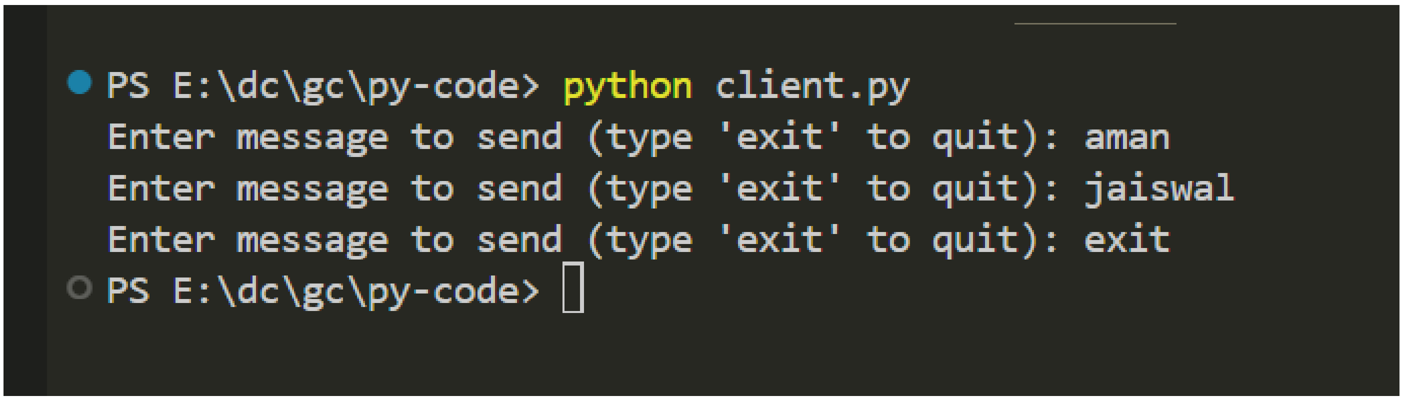
server

```python
# 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



```
PS E:\dc\gc\py-code> python server.py
Received message from ('192.168.245.204', 54945): aman
Received message from ('192.168.245.204', 54945): jaiswal
```



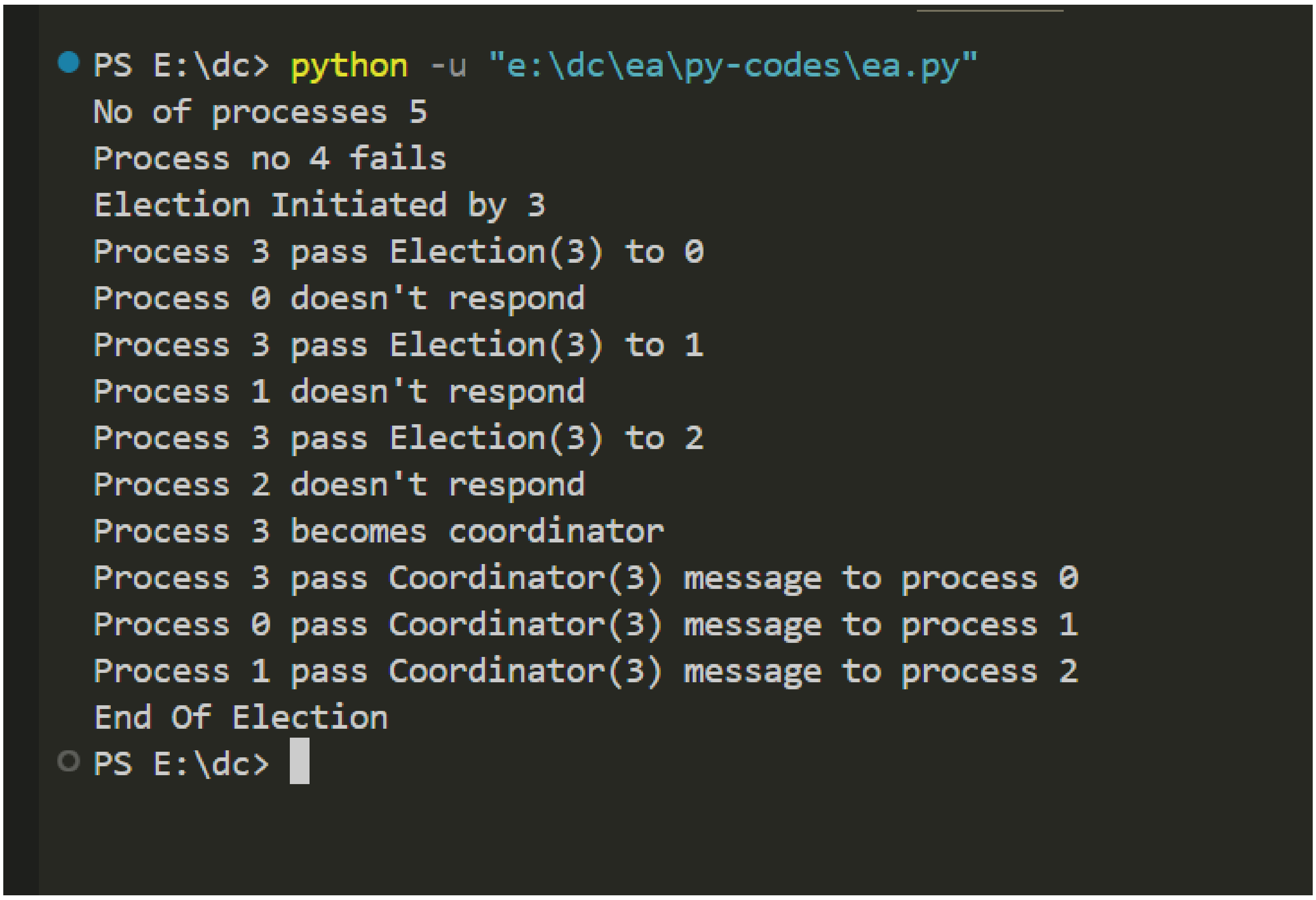
```
PS E:\dc\gc\py-code> python client.py
Enter message to send (type 'exit' to quit): aman
Enter message to send (type 'exit' to quit): jaiswal
Enter message to send (type 'exit' to quit): exit
PS E:\dc\gc\py-code> 
```

Election-Algo-ring-algo

Code

```python
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

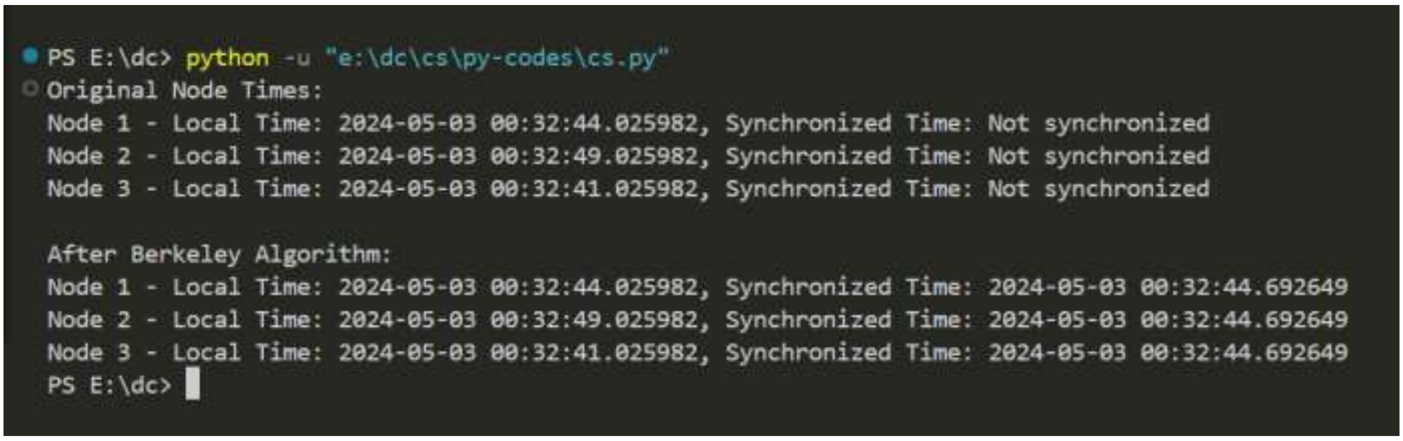

```
PS E:\dc> python -u "e:\dc\ea\py-codes\ea.py"
No of processes 5
Process no 4 fails
Election Initiated by 3
Process 3 pass Election(3) to 0
Process 0 doesn't respond
Process 3 pass Election(3) to 1
Process 1 doesn't respond
Process 3 pass Election(3) to 2
Process 2 doesn't respond
Process 3 becomes coordinator
Process 3 pass Coordinator(3) message to process 0
Process 0 pass Coordinator(3) message to process 1
Process 1 pass Coordinator(3) message to process 2
End Of Election
PS E:\dc>
```

Exp 6 – clock synchronization - berkeley_algorithm

Code

```python
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



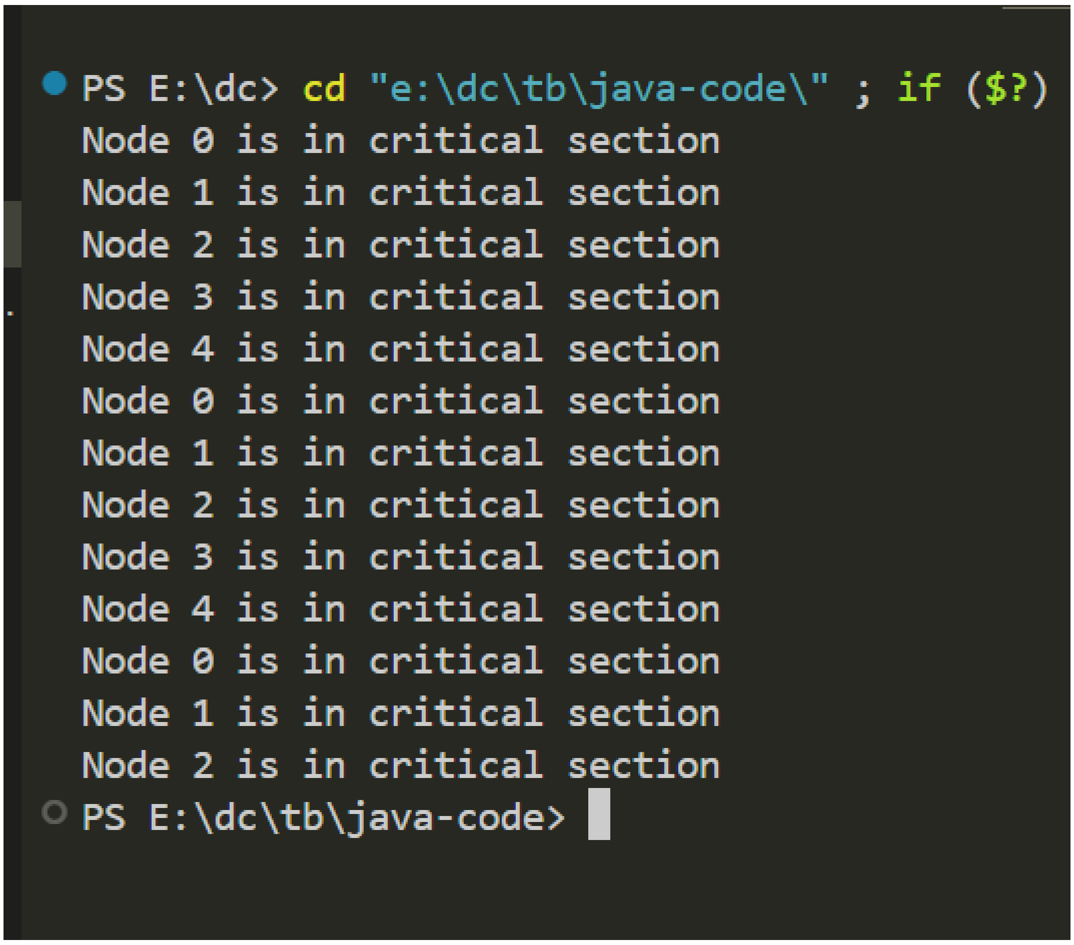
```
PS E:\dc> python -u "e:\dc\cs\py-codes\cs.py"
Original Node Times:
Node 1 - Local Time: 2024-05-03 00:32:44.025982, Synchronized Time: Not synchronized
Node 2 - Local Time: 2024-05-03 00:32:49.025982, Synchronized Time: Not synchronized
Node 3 - Local Time: 2024-05-03 00:32:41.025982, Synchronized Time: Not synchronized

After Berkeley Algorithm:
Node 1 - Local Time: 2024-05-03 00:32:44.025982, Synchronized Time: 2024-05-03 00:32:44.692649
Node 2 - Local Time: 2024-05-03 00:32:49.025982, Synchronized Time: 2024-05-03 00:32:44.692649
Node 3 - Local Time: 2024-05-03 00:32:41.025982, Synchronized Time: 2024-05-03 00:32:44.692649
PS E:\dc> 
```

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++) {
```

```java
            threads[i] = new Thread(new CriticalSection(i));
            threads[i].start();
        }
    }
}
```



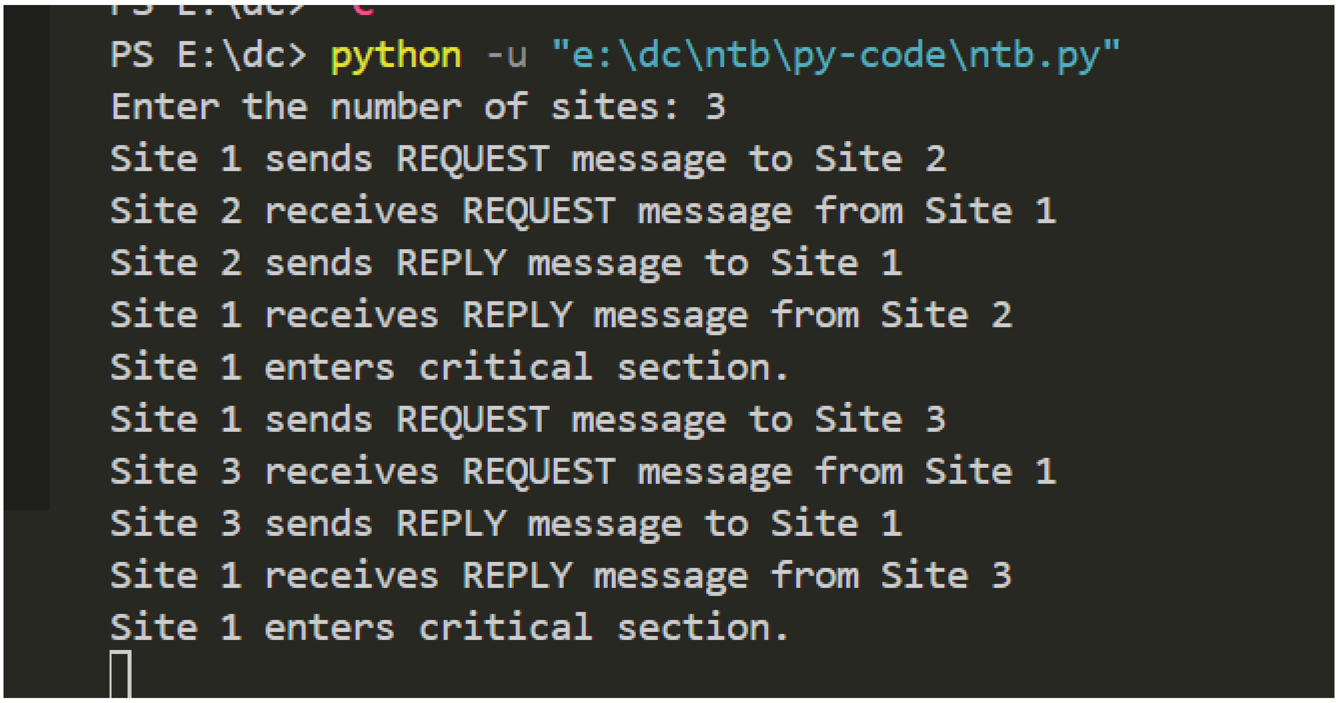
```
PS E:\dc> cd "e:\dc\tb\java-code\" ; if ($?)
Node 0 is in critical section
Node 1 is in critical section
Node 2 is in critical section
Node 3 is in critical section
Node 4 is in critical section
Node 0 is in critical section
Node 1 is in critical section
Node 2 is in critical section
Node 3 is in critical section
Node 4 is in critical section
Node 0 is in critical section
Node 1 is in critical section
Node 2 is in critical section
PS E:\dc\tb\java-code>
```

non token based – Ricart Agrawala Algorithm.

Code - python

```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



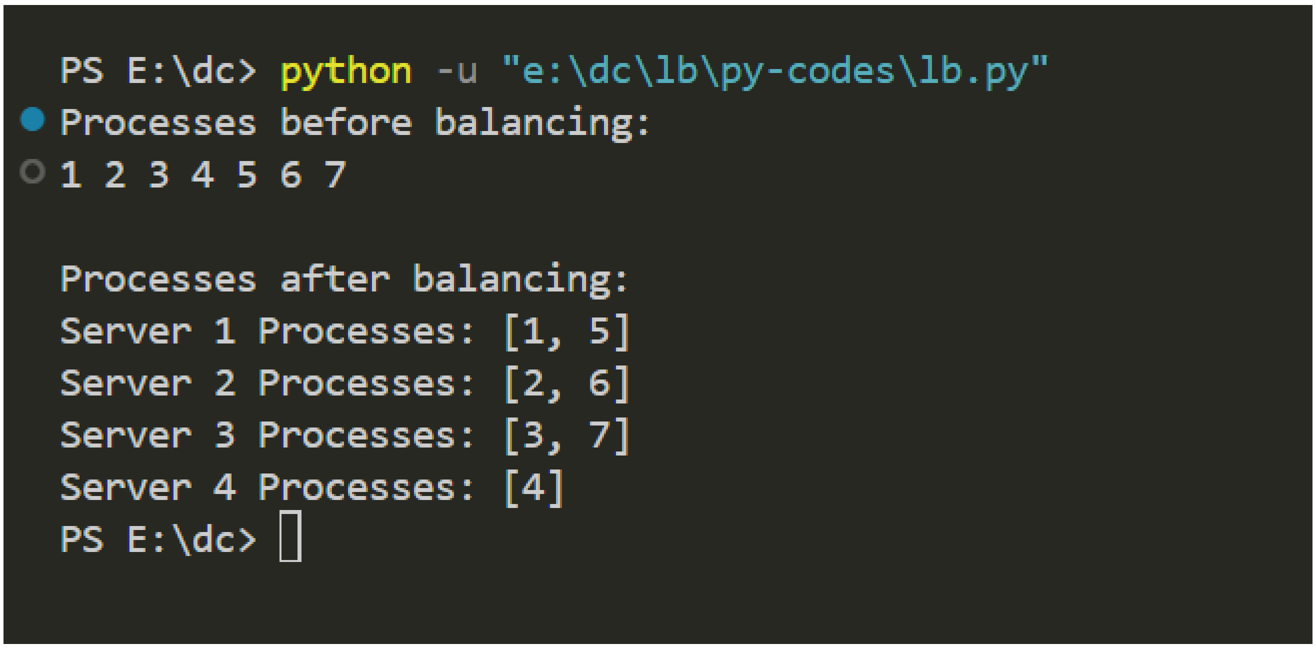
```
PS E:\dc> python -u "e:\dc\ntb\py-code\ntb.py"
Enter the number of sites: 3
Site 1 sends REQUEST message to Site 2
Site 2 receives REQUEST message from Site 1
Site 2 sends REPLY message to Site 1
Site 1 receives REPLY message from Site 2
Site 1 enters critical section.
Site 1 sends REQUEST message to Site 3
Site 3 receives REQUEST message from Site 1
Site 3 sends REPLY message to Site 1
Site 1 receives REPLY message from Site 3
Site 1 enters critical section.
```

Round robin - load balancer

Code - python

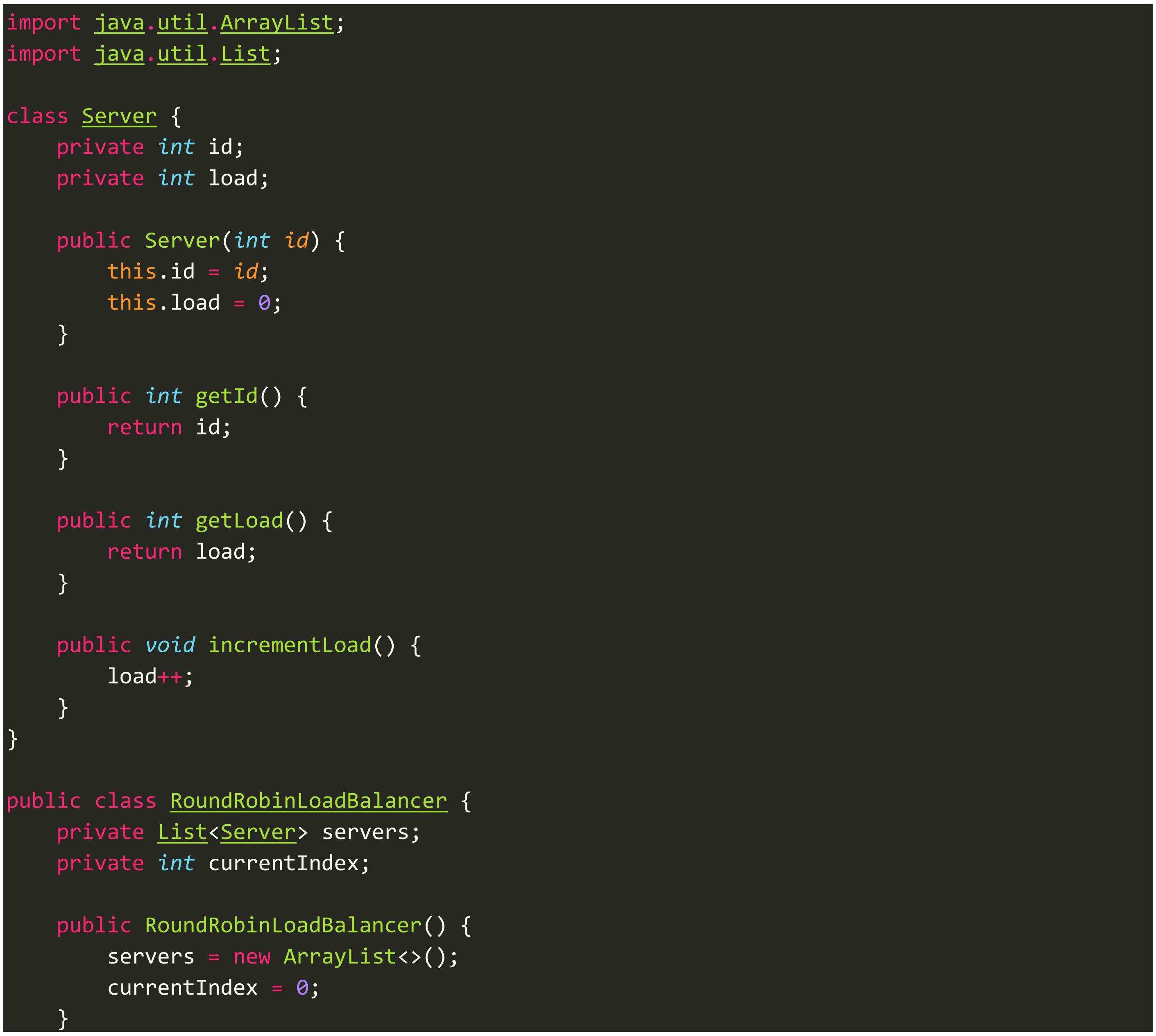
```python
class RoundRobinLoadBalancer:
    def __init__(self, numServers):
        self.numServers = numServers
        self.servers = [[] for _ in range(numServers)]

    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



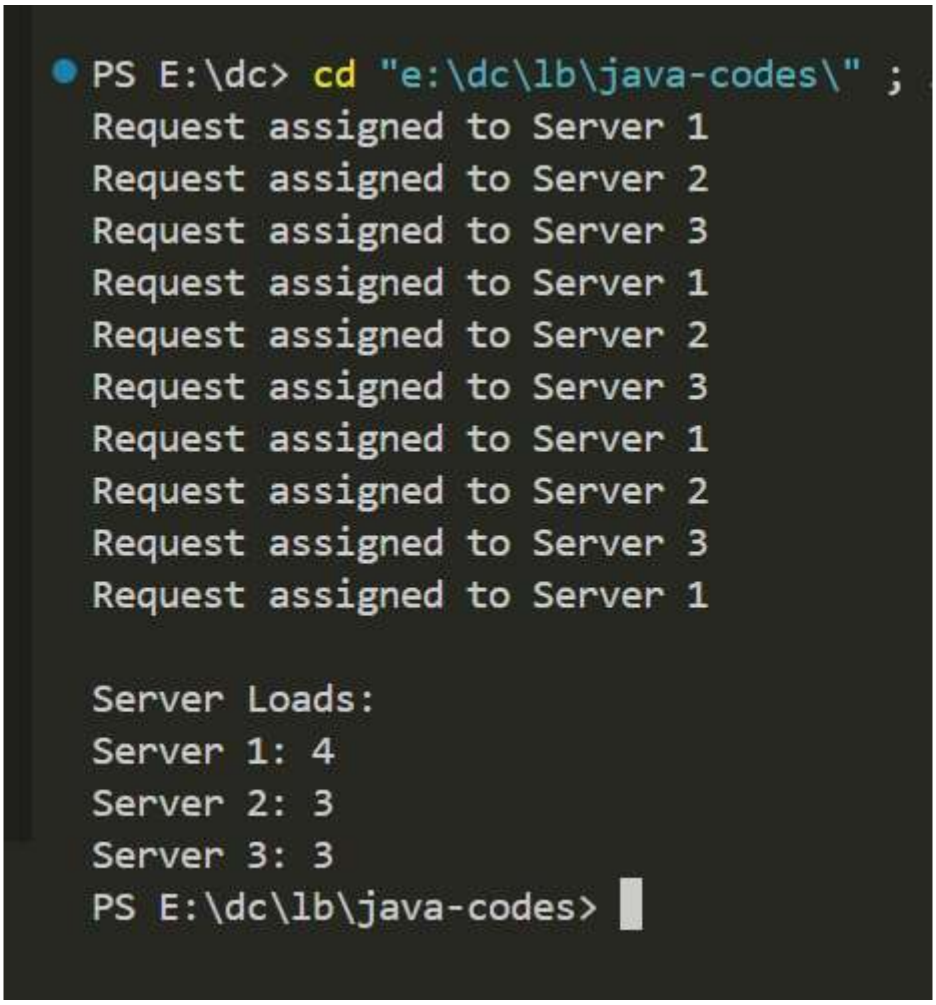
```
PS E:\dc> python -u "e:\dc\lb\py-codes\lb.py"
Processes before balancing:
1 2 3 4 5 6 7

Processes after balancing:
Server 1 Processes: [1, 5]
Server 2 Processes: [2, 6]
Server 3 Processes: [3, 7]
Server 4 Processes: [4]
PS E:\dc> 
```

Code - java



```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



```
PS E:\dc> cd "e:\dc\lb\java-codes\" ;
Request assigned to Server 1
Request assigned to Server 2
Request assigned to Server 3
Request assigned to Server 1
Request assigned to Server 2
Request assigned to Server 3
Request assigned to Server 1
Request assigned to Server 2
Request assigned to Server 3
Request assigned to Server 1

Server Loads:
Server 1: 4
Server 2: 3
Server 3: 3
PS E:\dc\lb\java-codes>
```