**Name: Shivam Tiwari**

**Roll No: 5117060**

**Aim:** To implement Group Communication

**Theory:**

A **group** is an operating system abstraction for a collective of related processes. A set of cooperative processes may, for example, form a group to provide an extendable, efficient, available and reliable service. The group abstraction allows member processes to perform computation on different hosts while providing support for communication and synchronisation between them.

The term **multicast** means the use of a single communication primitive to send a message to a specific set of processes rather than using a collection of individual point to point message primitives. Therefore common protocol for distributed system must support more flexible and efficient group communication in which a group address can be mapped on one or more data link addresses and the routing protocol can use a data link multicast address to send a message to all the receivers belonging to the group defined by the multicast address.

In both one to one communication and one to many communication the sender of a process has to specify two parameters:-

1. Destination address
2. Pointer to the message data

Therefore ideally the same send primitive can be used in one to one communication and one to many communication.

If destination address is specified in the send primitive is that of single process, the message is sent to that one process. On the other hand if the destination address is a group address the message is sent to all processes that belong to that group.

**Approach:**

1. **Many to One Communication**

Here sender sends message to a single receiver. The receiver may be selective or non-selective. A selective receiver specifies a unique sender, and a message. Exchange takes place only if that sender finds a message. A non-selective receiver specifies a set of senders and if senders in the set send a message to this receiver, a message exchange takes place.

1. **Many to Many Communication**

Here multiple senders send a message to multiple receiver. Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application.

**Code:**

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class GroupChat

{

private static final String TERMINATE = "Exit";

static String name;

static volatile boolean finished = false;

public static void main(String[] args)

{

if (args.length != 2)

System.out.println("Two arguments required: <multicast-host> <port-number>");

else

{

try

{

InetAddress group = InetAddress.getByName(args[0]);

int port = Integer.parseInt(args[1]);

Scanner sc = new Scanner(System.in);

System.out.print("Enter your name: ");

name = sc.nextLine();

MulticastSocket socket = new MulticastSocket(port);

// Since we are deploying

socket.setTimeToLive(0);

//this on localhost only (For a subnet set it as 1)

socket.joinGroup(group);

Thread t = new Thread(new

ReadThread(socket,group,port));

// Spawn a thread for reading messages

t.start();

// sent to the current group

System.out.println("Start typing messages...\n");

while(true)

{

String message;

message = sc.nextLine();

if(message.equalsIgnoreCase(GroupChat.TERMINATE))

{

finished = true;

socket.leaveGroup(group);

socket.close();

break;

}

message = name + ": " + message;

byte[] buffer = message.getBytes();

DatagramPacket datagram = new

DatagramPacket(buffer,buffer.length,group,port);

socket.send(datagram);

}

}

catch(SocketException se)

{

System.out.println("Error creating socket");

se.printStackTrace();

}

catch(IOException ie)

{

System.out.println("Error reading/writing from/to socket");

ie.printStackTrace();

}

}

}

}

class ReadThread implements Runnable

{

private MulticastSocket socket;

private InetAddress group;

private int port;

private static final int MAX\_LEN = 1000;

ReadThread(MulticastSocket socket,InetAddress group,int port)

{

this.socket = socket;

this.group = group;

this.port = port;

}

@Override

public void run()

{

while(!GroupChat.finished)

{

byte[] buffer = new byte[ReadThread.MAX\_LEN];

DatagramPacket datagram = new

DatagramPacket(buffer,buffer.length,group,port);

String message;

try

{

socket.receive(datagram);

message = new

String(buffer,0,datagram.getLength(),"UTF-8");

if(!message.startsWith(GroupChat.name))

System.out.println(message);

}

catch(IOException e)

{

System.out.println("Socket closed!");

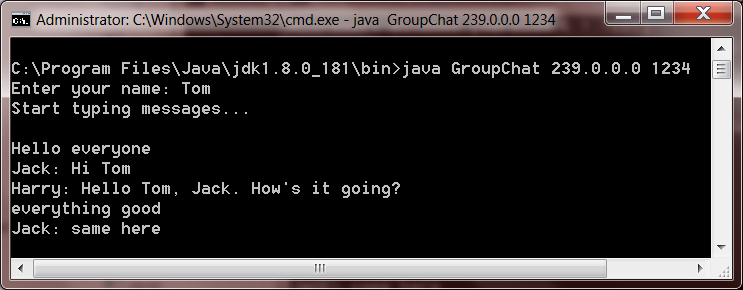
}

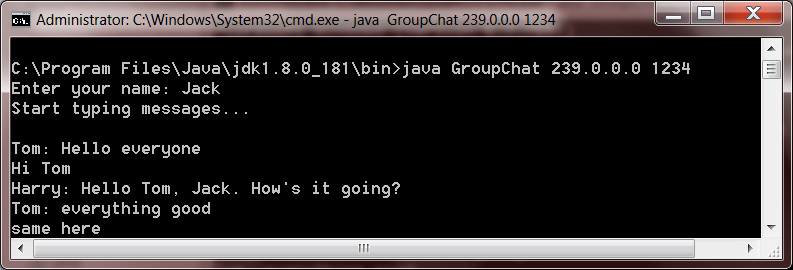
}

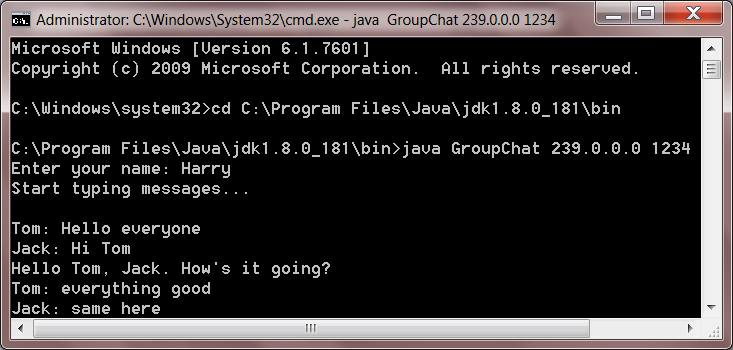
}

}

**Output:**







**Conclusion:**

Thus we have successfully studied and implemented group communication in a distributed system.