/\* Version 1.0 (watch for bugs), 2023-03-02:

Name: Rohan Dhoyda

Date: 2023-03-07

Java Version 19.0.1.1

Clark Elliott, GossipStarter.java copyright (c) 2023 with all rights reserved.

Here is some basic utility code to get you started on your Network Gossip Protocol system.

FEATURES:

1. Translation from Java Object to byte array, to datagram, over network, back to byte array, to Java Object.

2. UDP/Datagram listener.

3. UDP/Datagram sending.

4. Two threads: one listening for datagrams and one scanning for user console input.

5. UDP Listener loop fires off a worker to handle the incoming Java object.

Note that with three (or more) threads sharing one console file for output, console messages may

be in a scrambled order.

Java version 19.0.1 (change as appropriate!)

Compile: javac GossipStarter.java

\*/

import java.io.\*;

import java.net.\*;

class GossipData implements Serializable{

// Must be serializable to send 1 bit after another over the network.

int nodeNumber; // Here we have defined an integer variable that will represent node number of a Network

int average; // An integer variable average have been defined that will represent average number of a Network

int highValue; //An integer variable HighValue that will represent Highest Value in a Network

int lowValue; //An Integer Variable LowValue that will represent lowest value in a network

String userString; // A string data have defined for any additional information or comments that needs to be sent in a gossip network

}

class GossipWorker extends Thread {

//We have defined a class named Gossip Worker over here which extends thread class

GossipData gossipObj; //Gossip object is the object of the class gossip data

GossipWorker (GossipData c) {gossipObj = c;} // Constructor of the class will take an argument c and assign arg c to local object

public void run(){

System.out.println("\nGW: In Gossip worker: " + gossipObj.userString + "\n"); // The additional information will be sent along with the gossip data

}

}

public class GossipProgram {

public static int serverPort = 48100; // An integer variable is defined that will store the port number that the server will listen to

public static int NodeNumber = 0; // The integer variable Node Number is the value of the first argument that will be passed to the program

public static void main(String[] args) throws Exception {

System.out.println

("Rohan Dhoyda's Gossip Server 2.0 is now starting up and it is listening to the port " + GossipProgram.serverPort + ".\n");

ConsoleLooper CL = new ConsoleLooper(); // A console looper class is created and starts a different thread to run it

Thread t = new Thread(CL);

t.start();

boolean loopControl = true; // A loop Control boolean variable is created and is initialized to the value true and it keeps the datagram listener running.

try{

DatagramSocket DGSocket = new DatagramSocket(GossipProgram.serverPort);//A DGSocket object is created in the code which binds with GossipStarter.ServerPort using Datagram Socket Constructor

System.out.println("SERVER:Buffer size Received: " + DGSocket.getReceiveBufferSize() + "\n"); //The getReceiveBufferSize is called that will help in obtaining size of the receive buffer

byte[] incomingData = new byte[1024]; //Array named incoming data has a length of 1024

InetAddress IPAddress = InetAddress.getByName("localhost"); //Will generate IP address of the local host an assigns it to an object named IP Address

while (loopControl) {

//The loop will continue to execute while the loopcontrol boolean variable will be true

DatagramPacket incomingPacket = new DatagramPacket(incomingData, incomingData.length); //The incoming data will be received from the DG Socket. The constructor Datagram Packet will take the byte array and length as parameters from the incoming data

DGSocket.receive(incomingPacket); //Received Incoming data from the Socket. The data is stored in incoming Packet object

byte[] data = incomingPacket.getData(); //Data is retrieved from the packet and stored in a new byte array named data

ByteArrayInputStream in = new ByteArrayInputStream(data); //The object will read the data in the byte array

ObjectInputStream is = new ObjectInputStream(in); //The 'is' is used to deserialize the object which were previously serialized and the object will read object from byte array

try {

GossipData gossipObj = (GossipData) is.readObject(); // The ObjectInputStream object is read using readObject

if (gossipObj.userString.indexOf("stoptheserver") > -1){

System.out.println("SERVER: Stop the UDP listener now.\n");

loopControl = false; //The loop control is set to false it will cause the while loop to exit

}

System.out.println("\nSERVER: Gossip object received = " + gossipObj.userString + "\n"); //The code will print the message stating that a gossip object is received

new GossipWorker(gossipObj).start(); // A new Gossip worker object passing the gossipobj is created. The start method is called to start the execution of the worker thread

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

} catch (SocketException e) {

e.printStackTrace();

} catch (IOException i) {

i.printStackTrace();

}

}

}

class ConsoleLooper implements Runnable {

public void run(){

// The run method will be executed in a seperate thread. This is designed to listen for incoming datagrams and it will also allow user to enter command from the console

System.out.println("CL: In the Console Looper Thread");

BufferedReader in = new BufferedReader(new InputStreamReader(System.in)); // The BufferedReader object will be used to read the input which will be given from the console

try {

String someString; //A string variable named substring is created

do {

System.out.print

("CL: A String needs to be entered to be sent to the gossipServer, (or, quit): ");

System.out.flush ();

someString = in.readLine (); //Reads a line of text from the users input

if (someString.indexOf("quitting the process") > -1){

System.out.println("CL: As per the user request it is exiting now.\n");

System.exit(0); // Ugly way to stop. You can fix with a more elegant throw.

}

try{

System.out.println("CL: The preparation of Datagram Packet is initiated now...");

DatagramSocket DGSocket = new DatagramSocket(); //A new Datagram Socket DGSocket is created

InetAddress IPAddress = InetAddress.getByName("localhost"); //A Reference IP Address of local host is obtained

GossipData gossipObj = new GossipData(); //A new Gossip Data object GossipObj is created

gossipObj.userString = someString; //the userstring object is set to someString

ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

ObjectOutputStream os = new ObjectOutputStream(outputStream);

os.writeObject(gossipObj);

byte[] data = outputStream.toByteArray();

DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, GossipProgram.serverPort); // A sendPacket wich is a new DatagramPacket is created using data, data.lengthIPAddress, GossipStarter.serverPort

DGSocket.send(sendPacket); //Sends the datagram to the gossipserver

System.out.println("CL: Datagram has now been delivered.");

} catch (UnknownHostException UH){

System.out.println("\nCL: The Host is Unknown.\n");

UH.printStackTrace();

}

} while (true);

} catch (IOException x) {x.printStackTrace ();}

}

}

OUTPUT:

Microsoft Windows [Version 10.0.22621.1265]

(c) Microsoft Corporation. All rights reserved.

C:\Distributed System\GOSSIP CALCULATION>javac GossipProgram.java

C:\Distributed System\GOSSIP CALCULATION>java GossipProgram

Rohan Dhoyda's Gossip Server 2.0 is now starting up and it is listening to the port 48100.

CL: In the Console Looper Thread

CL: A String needs to be entered to be sent to the gossipServer, (or, quit): SERVER:Buffer size Received: 65536

Avengers Assemble

CL: The preparation of Datagram Packet is initiated now...

CL: Datagram has now been delivered.

CL: A String needs to be entered to be sent to the gossipServer, (or, quit):

SERVER: Gossip object received = Avengers Assemble

GW: In Gossip worker: Avengers Assemble