Name : Palak Keni

TEIT Batch B

Roll No : 26

**TITLE: Design distributed application which consists of a client and server using threads.**

**Scenario:**

The server listens to multiple clients on a network . Each client can append a string to the list.

Every time the user appends string the list gets updated and is shown to the user. When a particular client is modifying the string variable, no other client can access that string variable.

The user can type Exit to disconnect from the server.

**Problems:**

1) If two users try to save at the same time, there would be inconsistency.

2) If the data is too large , then it may be a problem while processing at server.

3) If the data is not saved properly or there are some connection, the system would be inconsistent.

**Solution:**

The application solves **Problem 1** by locking onto a particular client. It holds the lock when the user is saving data. Hence the consistency is maintained

**CODE:**

**Server.java**

import java.io.\*;

import java.text.\*;

import java.util.\*;

import java.net.\*;

public class Server {

static String text = "";

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

ServerSocket serverSocket = new ServerSocket(9898);

while (true){

Socket socket1 = null;

try{

socket1 = serverSocket.accept();

System.out.println("New client connected: " + socket1);

DataInputStream dataInput = new DataInputStream(socket1.getInputStream());

DataOutputStream dataOutput = new DataOutputStream(socket1.getOutputStream());

System.out.println("New thread is assiged to this client");

Thread t = new ClientHandler(socket1, dataInput, dataOutput);

t.start();

}

catch (Exception e){

socket1.close();

e.printStackTrace();

}

}

}

}

class ClientHandler extends Thread {

DataInputStream dataInput;

DataOutputStream dataOutput;

Socket socket1;

public ClientHandler(Socket s, DataInputStream dataInput, DataOutputStream dataOutput){

this.socket1 = s;

this.dataInput = dataInput;

this.dataOutput = dataOutput;

}

@Override

public synchronized void run() {

String receivedText;

while (true) {

try {

dataOutput.writeUTF("Enter words or type Exit to terminate connection");

receivedText = dataInput.readUTF();

if(receivedText.equals("Exit")){

System.out.println("Client " + this.socket1 + " has sent an exit statement");

System.out.println("Connection is being closed");

this.socket1.close();

System.out.println("Connection closed");

break;

}

Server.text += receivedText;

Server.text += "\n";

dataOutput.writeUTF(Server.text);

}

catch (IOException e) {

e.printStackTrace();

}

}

try{

this.dataInput.close();

this.dataOutput.close();

}

catch(IOException e){

e.printStackTrace();

}

}

}

**Client.java**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class Client{

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

try{

Scanner sc = new Scanner(System.in);

InetAddress ip = InetAddress.getByName("localhost");

Socket socket1 = new Socket(ip, 9898);

DataInputStream dataInput = new DataInputStream(socket1.getInputStream());

DataOutputStream dataOutput = new DataOutputStream(socket1.getOutputStream());

while (true){

System.out.println(dataInput.readUTF());

String text = sc.nextLine();

dataOutput.writeUTF(text);

if(text.equals("Exit")) {

System.out.println("Closing the connection: " + socket1);

socket1.close();

System.out.println("Connection closed");

break;

}

String receivedText = dataInput.readUTF();

System.out.println("\nCurrent version of edited doc:\n"+receivedText);

}

sc.close();

dataInput.close();

dataOutput.close();

}

catch(Exception e){

e.printStackTrace();

}

}

}

**OUTPUT:**

