**4a )** Write a multi-threaded program in Java/c for chatting (multiuser and multi-terminal) using threads

Subject:- Unix Operating System System Lab Class :- TYIT

**Name: - SHRENIK JADHAV**

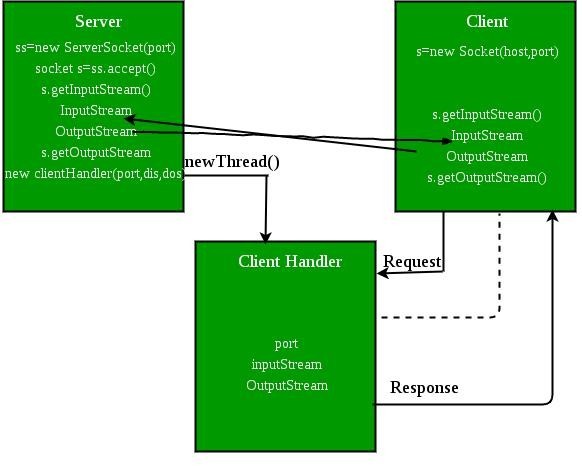
**OM GHARGE**

**PRN: -2020BTEIT00011,2020BTEIT00041**

**Objectives:**

1. To learn about threading in Linux/Unix and Java and difference between them
2. Use of system call/library to write effective programs

Theory:

As normal, we will create two Java files, **Server.java** and **Client.java**. Server file contains two classes namely **Server** (public class for creating server) and **ClientHandler** (for handling any client using multi-threading). Client file contain only one public class **Client** (for creating a client). Below is the flow diagram of how these three classes interact with each other.

**Server class :** The steps involved on server side are similar to the article Socket Programming In Java with a slight change to create the thread object after obtaining the streams and port number.

* 1. **Establishing the Connection:** Server socket object is initialized and inside a while loop a socket object continuously accepts incoming connection.
  2. **Obtaining the Streams:** The inputstream object and outputstream object is extracted from the current requests’ socket object.
  3. **Creating a handler object:** After obtaining the streams and port number, a new ClientHandler object (the above class) is created with these parameters.
  4. **Invoking the start() method :** The start() method is invoked on this newly created thread object.

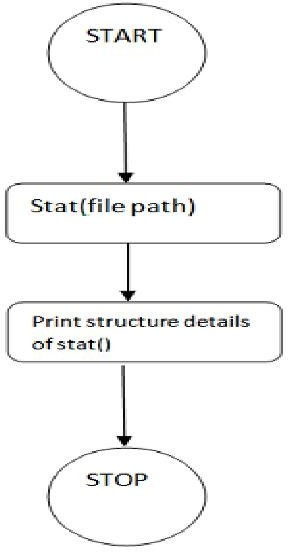
**ClientHandler class :** As we will be using separate threads for each request, lets understand the working and implementation of the ClientHandler class extending Threads. An object of this class will be instantiated each time a request comes.

* 1. First of all this class extends Thread so that its objects assumes all properties of Threads.
  2. Secondly, the constructor of this class takes three parameters, which can uniquely identify any incoming request, i.e. a **Socket**, a D**ataInput Stream** to read from and a DataOutputStream to write to. Whenever we receive any request of client, the server extracts its port number, the DataInputStream object and DataOutputStream object and creates a new thread object of this class and invokes start() method on it.

***Note : Every request will always have a triplet of socket, input stream and output stream. This ensures that each object of this class writes on one specific stream rather than on multiple streams.***

* 1. Inside the **run()** method of this class, it performs three operations: request the user to specify whether time or date needed, read the answer from input stream object and accordingly write the output on the output stream object.

Flowchart:

****

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
| Class Server |  |  |  |
| 1 | Ar | Vector | Store which clients are there. |
| 2 | i | int | Count of clients. |
| 3 | ss | ServerSocket | Create a socket for server side communication. |
| 4 | s | Socket | Socket is created. |
| 5 | dis | DataInputStream | Input data. |
| 6 | t | Thread | Used to create new thread for each client. |
| 7 | mtch | ClientHandler | Object of ClientHandler type. Used to handle client. |
| Class ClientHandler |  |  |  |
| 1 | scn | Scanner | Usedfor any input. |
| 2 | name | String | Store name of client. |
| 3 | dis | DataInputStream | Input a message. |
| 4 | dos | DataOutputStream | Output to standart ouput the message input. |
| 5 | received | String | Store message. |
| 6 | isloggedin | boolean | If the client is logged or not. |

**Program: import java.io.BufferedReader;**

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.List;

public class MultiUserChatServer {

private List<ClientHandler> clients = new ArrayList<>();

public void start(int port) {

try {

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Chat server started on port " + port);

while (true) {

Socket clientSocket = serverSocket.accept();

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

ClientHandler clientHandler = new ClientHandler(clientSocket);

clients.add(clientHandler);

clientHandler.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

public synchronized void broadcastMessage(String message, ClientHandler sender) {

for (ClientHandler client : clients) {

if (client != sender) {

client.sendMessage(sender.getUsername() + ": " + message);

}

}

}

public static void main(String[] args) {

MultiUserChatServer chatServer = new MultiUserChatServer();

chatServer.start(8080);

}

private class ClientHandler extends Thread {

private Socket clientSocket;

private PrintWriter writer;

private BufferedReader reader;

private String username;

public ClientHandler(Socket clientSocket) {

this.clientSocket = clientSocket;

}

public String getUsername() {

return username;

}

public void sendMessage(String message) {

writer.println(message);

}

@Override

public void run() {

try {

reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

writer = new PrintWriter(clientSocket.getOutputStream(), true);

writer.println("Enter your username:");

username = reader.readLine();

writer.println("Welcome to the chat, " + username + "!");

String message;

do {

message = reader.readLine();

if (message != null) {

broadcastMessage(message, this);

}

} while (message != null);

// Client has disconnected

System.out.println("Client disconnected: " + clientSocket);

clients.remove(this);

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

**run---**

**If Java is not installed, you can install OpenJDK using the following command:**

shell

* sudo apt update

sudo apt install openjdk-11-jdk

* Compile the Java code using the javac command:

shell

* javac MultiUserChatServer.java

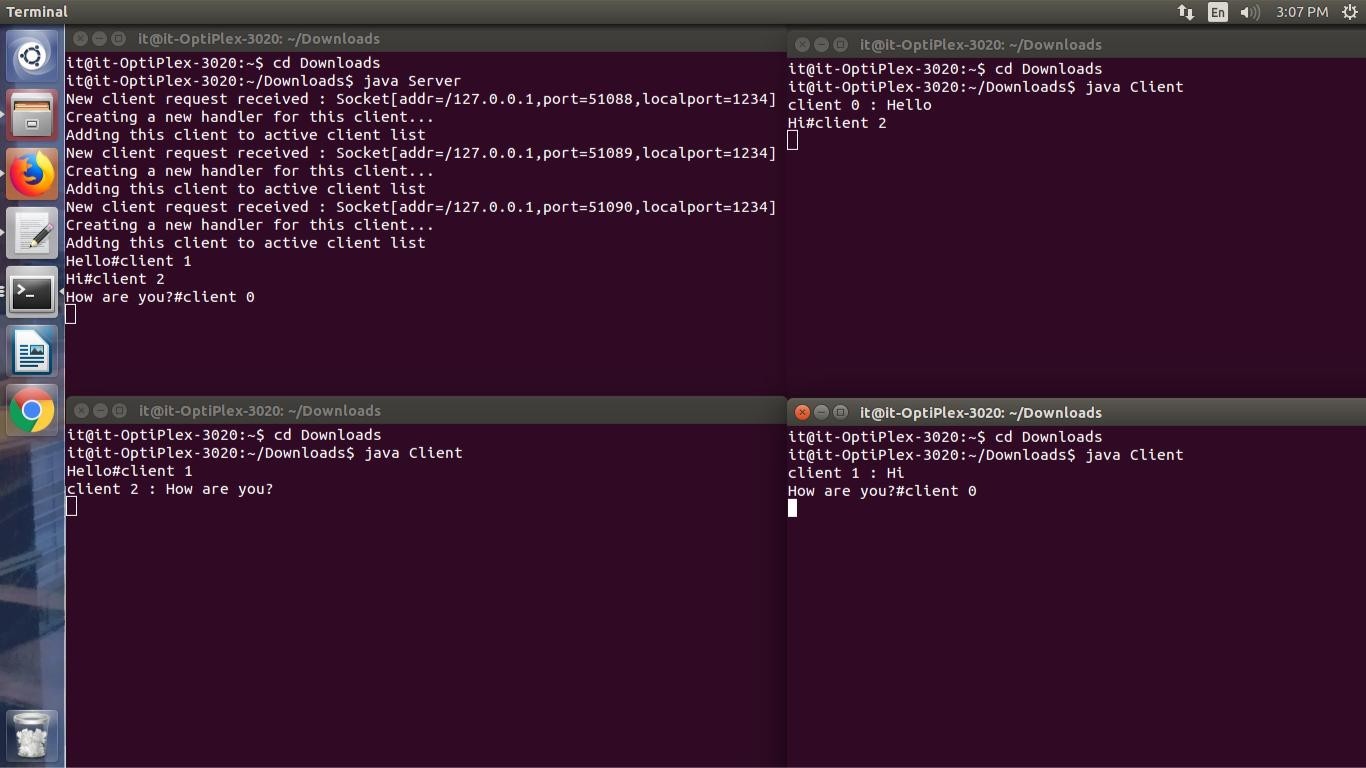
This will generate the MultiUserChatServer.class file.

* Run the program using the java command:

shell

java MultiUserChatServer

The chat server will start running and listen on port 8080.

Output:

**Conclusion:**

Various concepts and effective programming in Java using threads and sockets was studied. The concept of threading and multithreading understood.

References:

https://[www.geeksforgeeks.org/multithreading-in-java/](http://www.geeksforgeeks.org/multithreading-in-java/)