Distributed and Parallel Computing Lab

CS461 Lab3

Name: Dipean Dasgupta **ID:202151188**

Task: Multi-Client Multi-Server in Distributed Environment

Objective: To design and implement a distributed system in Java where multiple clients can communicate with multiple servers concurrently.

Code Language: **Java**

Steps:

Firstly, servers have to be created which will handle multiple requests from multiple clients connected to them.

Server side:

```
 import java.io.*;

import java.net.*;
import java.util.concurrent.atomic.AtomicInteger;

public class Server {
    private static final AtomicInteger client_ids = new AtomicInteger(1);

    public static void main(String[] args) throws IOException {
        int port = Integer.parseInt(args[0]); // Pass port as command line argument
        ServerSocket socket_server = new ServerSocket(port);
        System.out.println("Server starts at port " + port);

        while (true) {
            Socket socket = socket_server.accept();
            int clientId = client_ids.getAndIncrement();
            System.out.println("New client connected! to server on port " + port +
"  ID " + clientId);

            new ClientHandler(socket, clientId, port).start();
        }
    }
}

class ClientHandler extends Thread {
    private Socket socket;
    private int clientId;
```

```
    private int port;

    public ClientHandler(Socket socket, int clientId, int port) {
        this.socket = socket;
        this.clientId = clientId;
        this.port = port;
    }
    @Override
    public void run() {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            String line;
            while ((line = in.readLine()) != null) {
                System.out.println("Received from client  " + clientId + " on port "
+ port + ": " + line);
                out.println("Server on port " + port + " received: " + line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Similarly, multiple clients has to be created which will connect to the servers.

Client Side:

```
import java.io.*;
import java.net.*;
import java.util.Random;

public class Client {
    public static void main(String[] args) throws IOException {
        int[] ports = {6000, 6001, 6002}; // Use different ports
        Random random = new Random();
        int my_port = ports[random.nextInt(ports.length)];
```

```
        Socket socket = new Socket("localhost", my_port);
        System.out.println("Server Connected  on port " + my_port);

        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        BufferedReader stdIn = new BufferedReader(new
InputStreamReader(System.in));
        String input_client;
        String out_server;
        while ((input_client = stdIn.readLine()) != null) {
            if (input_client.equalsIgnoreCase("exit") ||
input_client.equalsIgnoreCase("quit")) {
                break;
            }
            out.println(input_client);
            out_server = in.readLine();
            System.out.println("Server on port " + my_port + " response: " +
out_server);
        }
        socket.close();
    }
}
```

Code Compilation

Compiling the Server and Client Code
Command: ***javac Server.java Client.java***

Running Servers

java Server 6000; java Server 6001; java Server 6002

```
D:\Java\CS461>java Server 6000
Server starts at port 6000
```

```
D:\Java\CS461>java Server 6001
Server starts at port 6001
```

```
D:\Java\CS461>java Server 6002
Server starts at port 6002
```

Running Multiple Clients:

Now the clients which are created will randomly connect to any of the 3 servers at 6000, 6001 and 6002.

Every client is given a unique ID by the server it is connected to, which also displays the messages that clients have sent along with their IDs and port numbers. The client can examine the server's answer and end the connection by sending a "quit" or "exit" command. The client chooses a server at random from a list of open ports when it connects.

Messages From Clients:

```
D:\Java\CS461>java Client
Server Connected  on port 6000
This is Lab 4 of CS461 DPC
Server on port 6000 response: Server on port 6000 received: This is Lab 4 of CS461 DPC
```

Client having ID 1 sends a message to server 6000

```
D:\Java\CS461>java Server 6000
Server starts at port 6000
New client connected! to server on port 6000  ID 1
Received from client  1 on port 6000: This is Lab 4 of CS461 DPC
```

Server 6000 confirms of receiving the message from Client ID 1

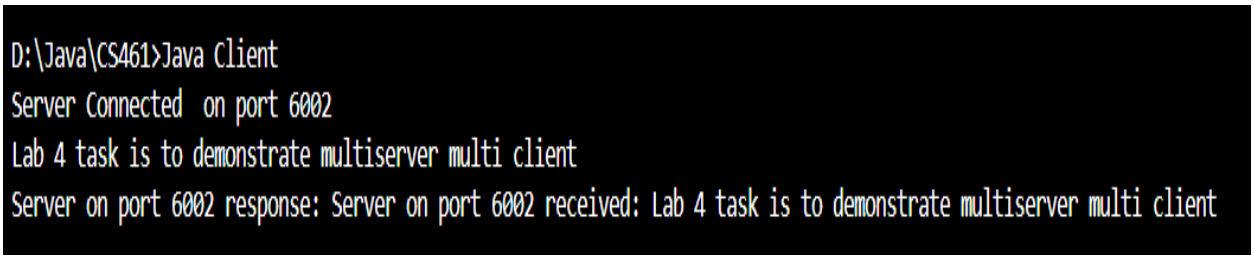
```
D:\Java\CS461>java Client
Server Connected  on port 6001
My ID is 202151188
Server on port 6001 response: Server on port 6001 received: My ID is 202151188

D:\Java\CS461>java Client
Server Connected  on port 6001
This is CS461 Lab
Server on port 6001 response: Server on port 6001 received: This is CS461 Lab
```

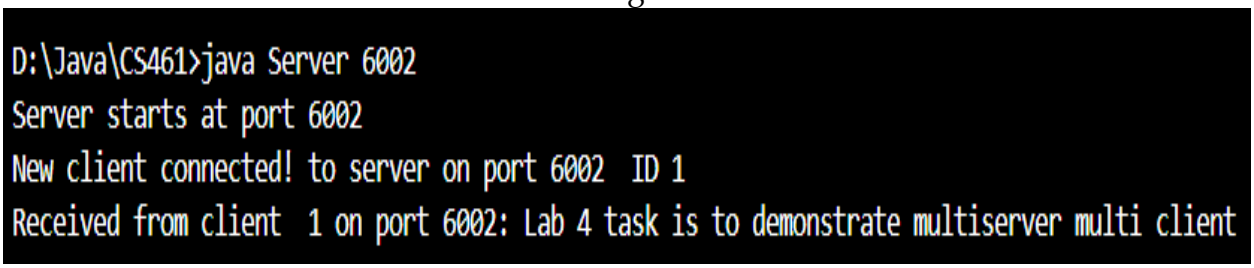
2 Clients connected to server 6001 sends message.

```
D:\Java\CS461>java Server 6001
Server starts at port 6001
New client connected! to server on port 6001  ID 1
New client connected! to server on port 6001  ID 2
Received from client  2 on port 6001: This is CS461 Lab
Received from client  1 on port 6001: My ID is 202151188
```

Server 6001 confirms receiving the messages from clients with ID 1 and 2



Client connected to server 6002 sends message



Server 6002 confirms the same from client with ID 1

So it is seen that multiple servers (6000, 6001, 6002) are connected randomly with multiple clients and successful transferring of messages is being carried out smoothly.