Programming Project 2 – Iterative Server vs Concurrent Server

Operating/Distributed Systems

**Objective**: In this project, the students will implement both the iterative server and the concurrent server and compare the performance.

**Task 1. Implementing the Client**

As the first step, implement the client. Use the code that you wrote for the project 1. When the client is started, it should print out your name and then create 100 threads The process that creates the 100 threads is the main client, and those 100 threads are called as sub-clients. Each sub-client, as soon as it is created, it should attempt to connect to the server. Each sub-client should keep trying to connect to the server if the server refuses to accept the connection request until it is connected.

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.time.\*;

public class Client{

    // initialize socket and input output streams

    private Socket socket = null;

    private DataInputStream input = null;

    private DataOutputStream out = null;

    Thread t;

    // constructor to put ip address and port

    public Client(String address, int port) {

        try {

            /\*\*

             \* Creates the socket and connects it to the IP address and port

             \*/

            long start0 = System.currentTimeMillis();

            socket = new Socket(address, port);

            System.out.println("connected");

            System.out.println("Hello, My name is Samuel!");

            for(int i = 0; i < 100; i++) {

                long start1 = System.currentTimeMillis();

                t = new Thread();

                t.start();

                long end1 = System.currentTimeMillis();

                System.out.println("Thread " + (i+1) + ":" + (end1-start1));

            }

            long end0 = System.currentTimeMillis();

            System.out.println("Elapsed time in milliseconds: :" + (end0-start0));

        } catch (UnknownHostException u) {

            System.out.println(u);

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public static void main(String args[]) {

        Client client = new Client("127.0.0.1", 5000);

    }

}

**Task 2. Implementing the Iterative Server**

When a client is connected, the server should perform the following computation for the client.

total = 0;

for (i = 0; i < 900000; i++)

total += total + i;

print total;

Note that the iterative server will handle the clients’ requests, one after another. Therefore, the iterative server will only accept a new connection request after finishing the computation for the currently connected client.

Assign a thread ID for each thread (e.g., thread1, thread2, …, thread100). For each thread, print out the thread ID and the server connection time. For example,

Thread 1 89392

Thread 2 89682

Thread 3 89772

import java.io.\*;

import java.net.\*;

import java.nio.charset.StandardCharsets;

public class Server {

    //initialize socket and input stream

    private Socket socket = null;

    private ServerSocket server = null;

    private DataInputStream in = null;

    public Server(int port) {

        try {

            server = new ServerSocket(port);

            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            while(true){

                socket = server.accept();

                long start00 = System.currentTimeMillis();

                System.out.println("Connected to Client: " );

                clientHandler(socket);

                long end00 = System.currentTimeMillis();

                System.out.println("Elapsed time in milliseconds: :" + (end00-start00));

            }

            } catch (IOException ioe) {

            ioe.printStackTrace();

        }

    }

    private static void clientHandler(final Socket socket) {

        int total = 0;

        for(int i = 0; i < 9000; i++){

            total += total + i;

        }

        System.out.println("Total: " + total);

        try {

            socket.close();

        } catch (IOException ioe) {

           ioe.printStackTrace();

        }

    }

    public static void main(String args[]) {

        long start00 = System.currentTimeMillis();

        Server server = new Server(5000);

        long end00 = System.currentTimeMillis();

        System.out.println("Elapsed time in milliseconds: :" + (end00-start00));

    }

}

**Task 3. Implementing the Concurrent Server**

The concurrent server, on the other hand, will pass the computation to a separate thread when a new connection request arrives. In other words, a separate thread will be created to perform the computation, and at the same time, the server will accept a new connection request. Therefore, the computation for multiple clients can happen simultaneously in the concurrent server.

Note that if your system has multiple cores, the concurrent server is expected to have a dramatic performance increase in terms of the computation time compared with the iterative server.

Print out the number of active server threads which should be updated dynamically while the server creates more threads, and some server threads finish handling the client’s request.

import java.io.\*;

import java.net.\*;

import java.nio.charset.StandardCharsets;

public class Server {

    //initialize socket and input stream

    private Socket socket = null;

    private ServerSocket server = null;

    private DataInputStream in = null;

    public Server(int port) {

        try {

            server = new ServerSocket(port);

            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            while(true){

                socket = server.accept();

                long start00 = System.currentTimeMillis();

                System.out.println("Connected to Client: " );

                // handle multiple client requests in different threads

                new Thread(new Runnable() {

                    public void run() {

                        clientHandler(socket);

                    }

                }).start();

                long end00 = System.currentTimeMillis();

                System.out.println("Elapsed time in milliseconds: :" + (end00-start00));

            }

            } catch (IOException ioe) {

            ioe.printStackTrace();

        }

    }

    private static void clientHandler(final Socket socket) {

        try (

            final PrintWriter writeToClient = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(), StandardCharsets.UTF\_8));

        ) {

            while (true) {

                //System.out.println(" ");

                System.out.println("Current Thread ID: "+ Thread.currentThread().getName());

                break;

            }

        } catch (IOException ioe) {

            ioe.printStackTrace();

        }

        try {

            socket.close();

        } catch (IOException ioe) {

           ioe.printStackTrace();

        }

    }

    public static void main(String args[]) {

        Server server = new Server(5000);

    }

}

**Task 4. Performance Evaluation**

The performance will be measured based on the total amount of time for processing all sub-clients’ computation requests. More precisely, start measuring the time, denoted by t\_{beginning}, when the first client sends a connection request to the server. Also measure the time denoted by t\_{finish} when the computations for all clients have been completed. Calculate the total amount of time required to finish the computation for all clients as follows:

delay = t\_{finish} – t\_{beginning}.

Repeat the measurements 10 times for each type of the server. Compute the average delay and the standard deviation for each type of the server.

**Task 5. Writing a one-page Report**

In your report, please include the following.

* Explanation on how to run you code.
* A paragraph describing the performance, in terms of the delay, of the concurrent server and iterative server. Compare the average delay and the standard deviation.
* A table or graph that summarizes the measurements. See an example below.

A picture containing text, white

Description automatically generated

**Files to submit**

* Source codes for the client and server (e.g., client.c and server.c)
* The one-page report.