

Programme Name: \_\_\_\_\_\_\_\_**BCS HONS**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Course Code: \_\_**CSC 2624**\_\_\_\_\_\_\_\_

Course Name: \_\_\_\_\_\_\_**Distributed And Parallel Computing**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Assignment** / Lab Sheet / Project / Case Study No. \_**2**\_\_\_

Date of Submission: \_\_\_\_\_\_**8/10/2021**\_\_\_\_\_\_\_\_\_\_\_\_\_

**Submitted By: Submitted To:**

Student Name**: Dipesh Tha Shrestha** Faculty Name**: Manoj Gautam**

IUKL ID: **041902900028** Department**: LMS**

Semester**: Fourth Semester**

Intake**: September 2019**

1. **Develop a java RMI Client and server program to compute the power of a number such that the client will call the RemoteCalcObject.computerPower(num) object method to compute the power of number and print the result in the screen.**

**Answer:** As we know, we need 2 folders (client and server) to run the above program. In given folder it will have its own files which will help them to connect with each other. Folder client will have 3 files named as Client.java, Number.java and RemoteCalcObject.java whereas Folder server will have Server.java, Number.java and NumberImpl.java. Therefore, Code for above program is given below:

**Folder Client**

**Client.java**

import java.rmi.\*;

import java.rmi.registry.\*;

public class Client {

    public static *void* main(*String*[] *args*) throws *RemoteException*, *NotBoundException* {

     try {

*Registry* remoteRegistry = LocateRegistry.getRegistry("127.0.0.1", 9300);

*Number* numm = (Number) remoteRegistry.lookup("number");

*RemoteCalcObject* remoteCalcObject = new RemoteCalcObject();

*double* finalnum = remoteCalcObject.computerPower(numm.getNum());

 System.out.println("The power of " + numm.getNum() + " by 2 is " + finalnum);

        } catch (*Exception* *e*) {

            System.out.println("Clinet error occoured " + e.toString());

        }

    }

}

**Number.java**

import java.rmi.\*;

public interface Number extends Remote {

    public *double* getNum() throws RemoteException;

}

**RemoteCalcObject.java**

import java.lang.Math;

class RemoteCalcObject {

    RemoteCalcObject() {

    }

    public *double* computerPower(*double* *num*) {

        return Math.pow(*num*, 2);

    }

}

**Folder Server**

**Server.java**

import java.rmi.server.UnicastRemoteObject;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

public class Server {

    public static *void* main(String[] *args*) {

        try {

            NumberImpl n1 = new NumberImpl(4);

            Number stub1 = (Number) UnicastRemoteObject.exportObject(n1, 0);

            Registry registry = LocateRegistry.getRegistry("127.0.0.1", 9300);

            registry.bind("number", stub1);

        } catch (Exception e) {

            System.out.println("Error :" + e);

        }

    }

}

**Number.java**

import java.rmi.\*;

public interface Number extends Remote {

    public *double* getNum() throws RemoteException;

}

**NumberImpl.java**

import java.rmi.\*;

import java.rmi.server.\*;

public class NumberImpl implements Number{

*double* numm;

    NumberImpl(*double* *newnumm*) throws RemoteException{

        this.numm = *newnumm*;

    }

    public *double* getNum() throws RemoteException{

        return this.numm;

    }

}

1. **Write an OpenMP C++ program to implement FOUR (4) parallel section clause by setting the number of threads to 4 and compute the sum of prime numbers up to (100 billion) using 4 threads.**

Answer: An OpenMP C++ program to implement FOUR (4) parallel section clause by setting the number of threads to 4 and compute the sum of prime numbers up to (100 billion) using 4 threads is given below:

#include<iostream>

#include<omp.h>

using *namespace* std;

*int* main(){

*long* *long* n = 100000000;

*long* *long* total = 0;

    #pragma omp parallel for num\_threads(4)

    for(*long* *long* i =1;i<=n;i++){

    #pragma omp critical

    {

*long* *long* it, num;

*bool* isPrime = true;

    num = i;

    if (num == 0 || num == 1) {

        isPrime = false;

    }

    else {

        for (it = 2; it <= num / 2; ++it) {

            if (num % it == 0) {

                isPrime = false;

                break;

            }

        }

    }

    if (isPrime){

        total = total+num;

        cout<<"prime : "<<num<<"\n";

        cout<<"total : "<<total<<"\n";

        }

}

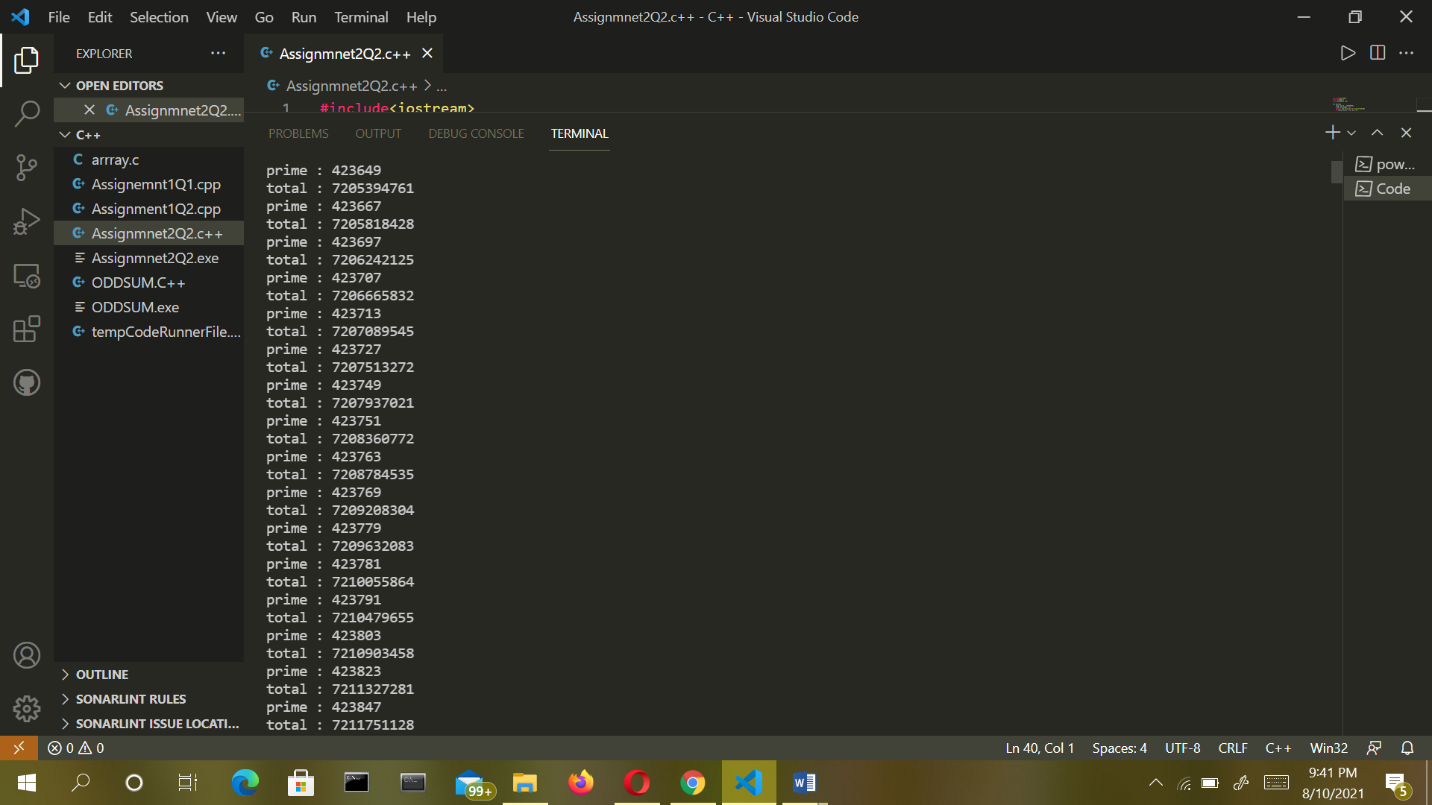
    }

cout<<"Total sum of prime number up to "<<n<<" is "<<total<<"\n";

    return EXIT\_SUCCESS;

}

It may take a lot of time to compute the sum of prime numbers up to (100 billion) using 4 threads, so here is the output of the process:



1. **Develop a multi-threaded web server that receive the file name from the client such that serverinfo.txt and return the file that resides in server to the client. Client will parse the file content and display it on the screen.**

Answer:

We are developing a multi-threaded web server that receive the file name from the client such that serverinfo.txt and return the file that resides in server to the client.

**Client-Side Program**: A client can communicate with a server using this code. This involves

Establish a Socket Connection

Communication

import java.io.\*;

import java.net.\*;

import java.util.\*;

// Client class

class Client {

    // driver code

    public static *void* main(String[] *args*) {

        // establish a connection by providing host and port

        // number

        try (Socket socket = new Socket("localhost", 1234)) {

            // writing to server

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            // reading from server

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            // object of scanner class

            Scanner sc = new Scanner(System.in);

            String line = null;

            while (!"exit".equalsIgnoreCase(line)) {

                // reading from user

                line = sc.nextLine();

                // sending the user input to server

                out.println(line);

                out.flush();

                // displaying server reply

                System.out.println("Server replied " + in.readLine());

            }

            // closing the scanner object

            sc.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

**Server-Side Program:** When a new client is connected, and he sends the message to the server.

**Server class:** The steps involved on the 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.

* Establishing the Connection: Server socket object is initialized and inside a while loop a socket object continuously accepts an incoming connection.
* Obtaining the Streams: The inputstream object and outputstream object is extracted from the current requests’ socket object.
* Creating a handler object: After obtaining the streams and port number, a new clientHandler object (the above class) is created with these parameters.
* Invoking the [start()](https://www.geeksforgeeks.org/start-function-multithreading-java/) method: The start() method is invoked on this newly created thread object.

Code:

import java.io.\*;

import java.net.\*;

// Server class

class Server {

    public static *void* main(String[] *args*) {

        ServerSocket server = null;

        try {

            // server is listening on port 1234

            server = new ServerSocket(1234);

            server.setReuseAddress(true);

            // running infinite loop for getting

            // client request

            while (true) {

                // socket object to receive incoming client

                // requests

                Socket client = server.accept();

                // Displaying that new client is connected

                // to server

                System.out.println("New client connected" + client.getInetAddress().getHostAddress());

                // create a new thread object

                ClientHandler clientSock = new ClientHandler(client);

                // This thread will handle the client

                // separately

                new Thread(clientSock).start();

            }

        } catch (IOException e) {

            e.printStackTrace();

        } finally {

            if (server != null) {

                try {

                    server.close();

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

    }

    // ClientHandler class

    private static class ClientHandler implements Runnable {

        private final Socket clientSocket;

        // Constructor

        public ClientHandler(Socket *socket*) {

            this.clientSocket = *socket*;

        }

        public *void* run() {

            PrintWriter out = null;

            BufferedReader in = null;

            try {

                // get the outputstream of client

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

                // get the inputstream of client

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

                String line;

                while ((line = in.readLine()) != null) {

                    // writing the received message from

                    // client

                    System.out.printf(" Sent from the client: %s\n", line);

                    out.println(line);

                }

            } catch (IOException e) {

                e.printStackTrace();

            } finally {

                try {

                    if (out != null) {

                        out.close();

                    }

                    if (in != null) {

                        in.close();

                        clientSocket.close();

                    }

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

    }

}