



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**

Desarrollo de Sistemas Distribuidos



Tarea 1

Cálculo de Pi

PROFESOR: Pineda Guerrero Carlos

ALUMNA: Ramírez Galindo Karina

GRUPO: 4CV11

Contenido

INTRODUCCION TEORICA 1

DESARROLLO..... 2

PRUEBAS 4

CONCLUSIONES..... 6

INTRODUCCION TEORICA

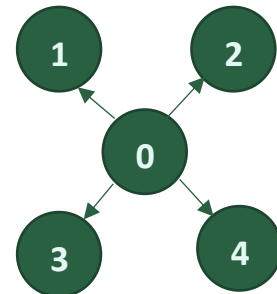
El número π es uno de los más famosos de la matemática. Mide la relación que existe entre la longitud de una circunferencia y su diámetro.

Un método bastante popular para el cálculo de π , es la utilización de las series infinitas de Gregory-Leibniz. Este método consiste en ir realizando operaciones matemáticas sobre series infinitas de números hasta que la serie converge en el número π . Aunque no es muy eficiente, se acerca cada vez más al valor de Pi en cada repetición, produciendo con precisión hasta cinco mil decimales de Pi con 500000 repeticiones.

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots\right)$$

Esta tarea consiste en desarrollar un programa distribuido, el cual calculará una aproximación de PI utilizando la serie de Gregory-Leibniz.

El programa va a ejecutar en forma distribuida sobre cinco nodos.



Se implementará la siguiente topología lógica en estrella:

El nodo 0 actuará como cliente y los nodos 1, 2, 3 y 4 actuarán como servidores.

El nodo 0 deberá implementar re-intentos de conexión a cada servidor, de manera que se pueda iniciar la ejecución de cada nodo en cualquier orden.

Se debe desarrollar un solo programa, por tanto, será necesario pasar como parámetro al programa el número de nodo actual, de manera que el programa pueda actuar como servidor o como cliente, según el número de nodo que pasa como parámetro.

Debido a que los nodos van a ejecutar en la misma computadora, cada nodo servidor deberá abrir un puerto diferente, por ejemplo, el nodo 1 podría abrir el puerto 50001, el nodo 2 podría abrir el puerto 50002, el nodo 3 podría abrir el puerto 50003 y el puerto 4 podría abrir el puerto 50004.

El objetivo del programa es distribuir el cálculo de PI en cuatro nodos (los nodos servidores). Cada nodo servidor deberá calcular la siguiente sumatoria:

Si "nodo" es impar:

$$\Sigma(4.0/(8 * i + 2 * (nodo - 2) + 3))$$

Si "nodo" es par:

$$-\Sigma(4.0/(8 * i + 2 * (nodo - 2) + 3))$$

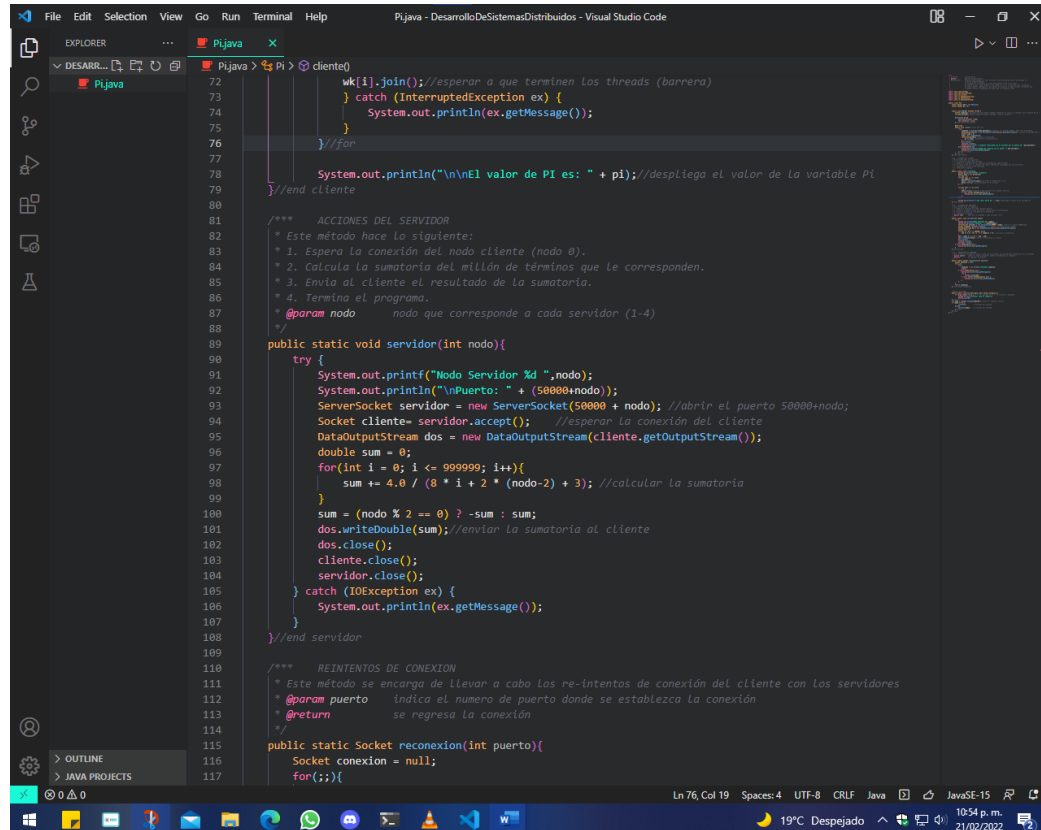
Donde:

$i = 0, 1, 2, \dots, 999999$ y "nodo" es el número de nodo (1, 2, 3 o 4).

DESARROLLO

De acuerdo con los requerimientos de la tarea, es necesario seccionar el código en las acciones que hará el servidor y las acciones que hará el cliente.

- Cada nodo servidor hace lo siguiente:
 1. Esperar la conexión del nodo cliente (nodo 0).
 2. Calcular la sumatoria del millón de términos que le corresponden.
 3. Enviar al cliente el resultado de la sumatoria.
 4. Terminar el programa.



```

72  wk[i].join(); //esperar a que terminen los threads (barrera)
73  } catch (InterruptedException ex) {
74      System.out.println(ex.getMessage());
75  }
76  } //for
77
78  System.out.println("\nEl valor de PI es: " + pi); //despliega el valor de la variable pi
79  } //end cliente
80
81  /**
82   * Este método hace lo siguiente:
83   * 1. Espera la conexión del nodo cliente (nodo 0).
84   * 2. Calcula la sumatoria del millón de términos que le corresponden.
85   * 3. Envía al cliente el resultado de la sumatoria.
86   * 4. Termina el programa.
87   * @param nodo    nodo que corresponde a cada servidor (1-4)
88   */
89  public static void servidor(int nodo){
90      try {
91          System.out.printf("Nodo Servidor %d ",nodo);
92          System.out.println("\nPuerto: " + (50000+nodo));
93          ServerSocket servidor = new ServerSocket(50000 + nodo); //abrir el puerto 50000+nodo;
94          Socket cliente= servidor.accept(); //esperar la conexión del cliente
95          DataOutputStream dos = new DataOutputStream(cliente.getOutputStream());
96          double sum = 0;
97          for(int i = 0; i <= 999999; i++){
98              sum += 4.0 / (8 * i + 2 * (nodo-2) + 3); //calcular la sumatoria
99          }
100         sum = (nodo % 2 == 0) ? -sum : sum;
101         dos.writeDouble(sum); //enviar la sumatoria al cliente
102         dos.close();
103         cliente.close();
104         servidor.close();
105     } catch (IOException ex) {
106         System.out.println(ex.getMessage());
107     }
108 } //end servidor
109
110 /**
111 * Este método se encarga de llevar a cabo los re-intentos de conexión del cliente con los servidores
112 * @param puerto    indica el número de puerto donde se establezca la conexión
113 * @return          se regresa la conexión
114 */
115 public static Socket reconexion(int puerto){
116     Socket conexion = null;
117     for(;;){

```

Ilustración 1

Sección de código de las funciones del servidor

- El nodo 0 (nodo cliente) hace lo siguiente:
 1. Conectarse a cada nodo servidor.
 2. Esperar el resultado de la sumatoria calculada por cada servidor.
 3. El valor de PI será la suma de las cuatro sumatorias calculadas por los servidores.
 4. Desplegar el valor de PI calculado.
 5. Terminar el programa.

```

File Edit Selection View Go Run Terminal Help
Pijava - DesarrolloDeSistemasDistribuidos - Visual Studio Code

EXPLORER
DESARROLLODESISTEMAS...
Pijava

Pijava
38 suma = dis.readDouble();
39 synchronized(obj){ //Bloque sincronizado
40 pi += suma; //sumatoria y sincronizacion
41 } //synchronized
42 dis.close();
43 conexion.close();
44 System.out.printf("\n Conexion finalizada con el servidor por el puerto %d ",pto_servidor);
45 }catch(Exception e){
46 System.out.println("Intento de conexion con el puerto: "+ pto_servidor);
47 System.out.println(e.getMessage());
48 } //end try-catch
49 } //end run
50 } //end class Worker
51
52 /** ACCIONES DEL CLIENTE
53 * Este método hace lo siguiente:
54 * 1. Se conecta a cada nodo servidor.
55 * 2. Espera el resultado de la sumatoria calculada por cada servidor.
56 * 3. El valor de PI será la suma de las cuatro sumatorias calculadas por los servidores.
57 * 4. Despliega el valor de PI calculado.
58 * 5. Termina el programa.
59 */
60 public static void cliente(){
61 System.out.println("Modo cliente");
62 Worker wk[] = new Worker[4];
63 int pto = 0;
64 for(int i=0; i < 4; i++){
65 pto=50001+i;
66 wk[i] = new Worker(pto); //se crean 4 threads del 1 al 4
67 wk[i].start(); //Iniciación de los threads
68 } //for
69
70 for(int i=0; i < 4; i++){
71 try {
72 wk[i].join(); //espera a que terminen los threads (barrera)
73 } catch (InterruptedException ex) {
74 System.out.println(ex.getMessage());
75 }
76 } //for
77
78 System.out.println("\nEl valor de PI es: " + pi); //despliega el valor de la variable PI
79 } //end cliente
80
81 /** ACCIONES DEL SERVIDOR
82 * Este método hace lo siguiente:
83 * 1. Espera la conexión del modo cliente (modo 0).

```

Ilustración 2

Sección de código de las funciones del servidor

Es necesario hacer utilizar hilos y sincronizarlos para el cálculo de la sumatoria de pi

```

File Edit Selection View Go Run Terminal Help
Pijava - DesarrolloDeSistemasDistribuidos - Visual Studio Code

EXPLORER
DESARROLLODESISTEMAS...
Pijava

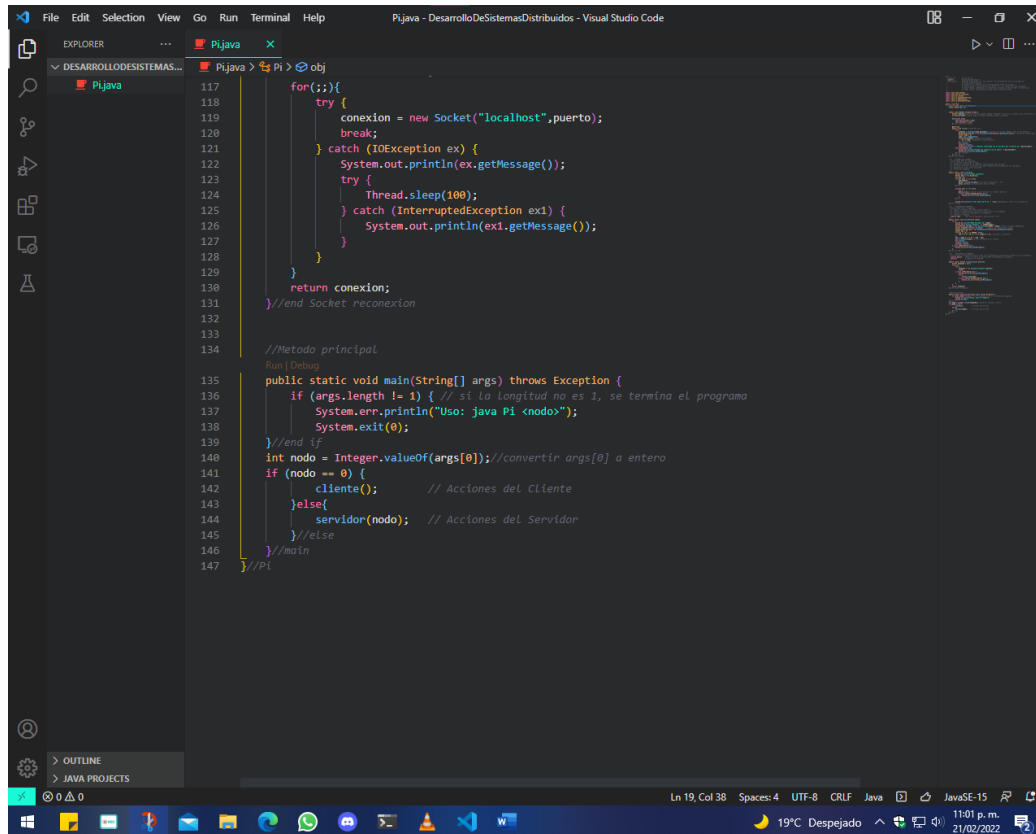
Pijava
11 import java.lang.*;
12 import java.net.ServerSocket;
13 import java.net.Socket;
14 import java.io.DataOutputStream;
15 import java.io.IOException;
16 import java.io.DataInputStream;
17
18 public class Pi{
19 static Object obj = new Object();
20 static double pi = 0;
21
22
23 static class Worker extends Thread {
24 Socket conexion; //variable de tipo Socket llamada "conexion" la cual va a contener una instancia de la clase Socket
25 int pto_servidor; //puerto donde se estará llevando a cabo la conexión
26
27 Worker(int pto){
28 this.pto_servidor = pto;
29 this.conexion = null;
30 } //end constructor Worker
31
32 @Override
33 public void run(){ //cuerpo del hilo
34 try{
35 conexion = reconexion(pto_servidor); //conexión al servidor 50000 + nodo con re-intentos
36 DataInputStream dis = new DataInputStream(conexion.getInputStream()); //stream de entrada para la conexión
37 double suma = 0;
38 suma = dis.readDouble();
39 synchronized(obj){ //Bloque sincronizado
40 pi += suma; //sumatoria y sincronizacion
41 } //synchronized
42 dis.close();
43 conexion.close();
44 System.out.printf("\n Conexion finalizada con el servidor por el puerto %d ",pto_servidor);
45 }catch(Exception e){
46 System.out.println("Intento de conexion con el puerto: "+ pto_servidor);
47 System.out.println(e.getMessage());
48 } //end try-catch
49 } //end run
50 } //end class Worker
51
52 /** ACCIONES DEL CLIENTE
53 * Este método hace lo siguiente:
54 * 1. Se conecta a cada nodo servidor.
55 * 2. Espera el resultado de la sumatoria calculada por cada servidor.
56 * 3. El valor de PI será la suma de las cuatro sumatorias calculadas por los servidores.
57 * 4. Despliega el valor de PI calculado.

```

Ilustración 3

sincronización de hilos para la sumatoria de pi

La verificación del número de nodo dependiendo si se trata de un cliente o un servidor se encuentra en el método main:



```

117     for(;;){
118         try {
119             conexion = new Socket("localhost",puerto);
120             break;
121         } catch (IOException ex) {
122             System.out.println(ex.getMessage());
123             try {
124                 Thread.sleep(100);
125             } catch (InterruptedException ex1) {
126                 System.out.println(ex1.getMessage());
127             }
128         }
129     }
130     return conexion;
131 } //end Socket reconexion
132
133
134 //Metodo principal
135 public static void main(String[] args) throws Exception {
136     if (args.length != 1) { // si la longitud no es 1, se termina el programa
137         System.err.println("Uso: java Pi <nodo>");
138         System.exit(0);
139     } //end if
140     int nodo = Integer.valueOf(args[0]); //convertir args[0] a entero
141     if (nodo == 0) {
142         cliente(); // Acciones del Cliente
143     } else {
144         servidor(nodo); // Acciones del Servidor
145     } //else
146 } //main
147 } //Pi
  
```

Ilustración 4

Método Principal

PRUEBAS

Se deberá probar el programa en una sola computadora utilizando cinco ventanas de comandos de Windows o cinco terminales de Linux o MacOS, cada ventana representará un nodo (una instancia del programa).

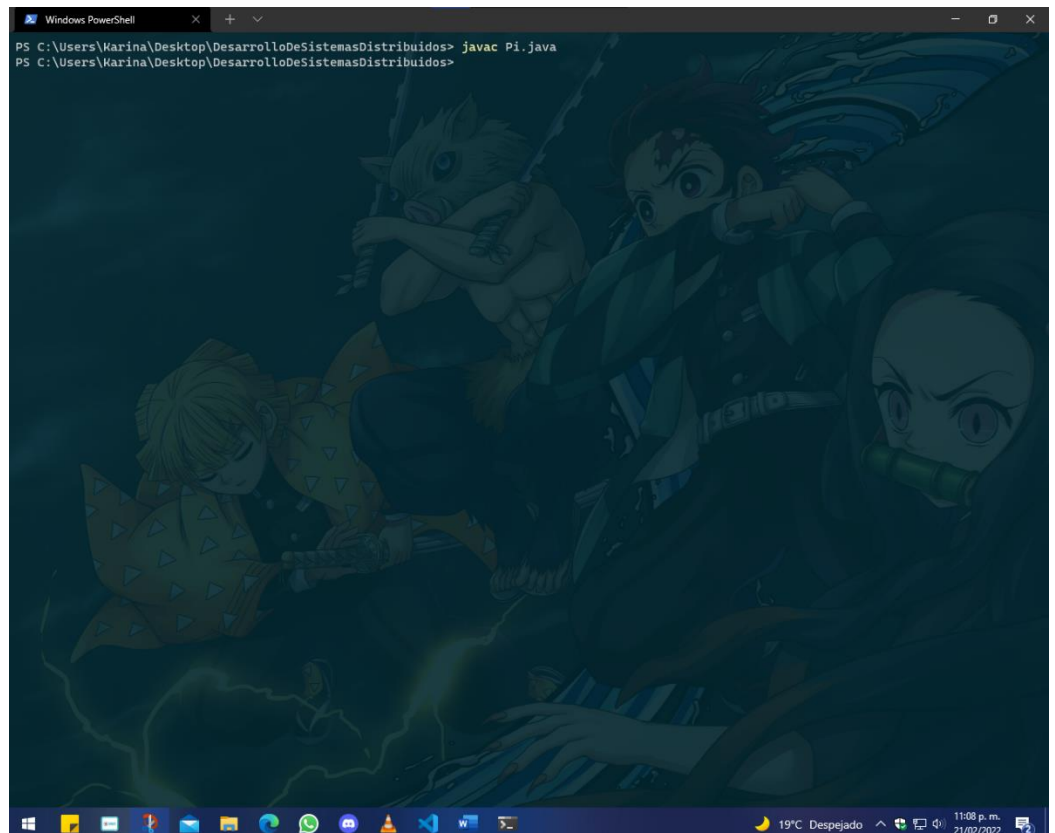


Ilustración 5

Compilación del programa Pi.java

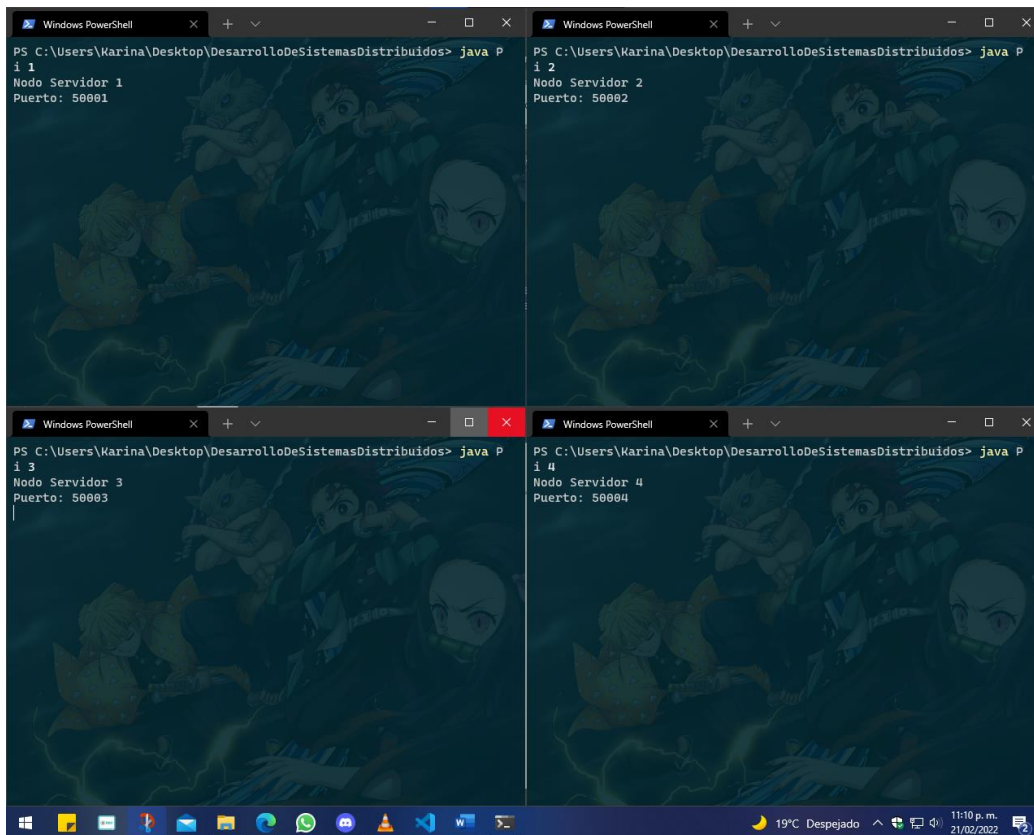


Ilustración 6

Ejecución de todos los servidores (1-4)

Como se observa en la ilustración 6, en cada terminal de servidor se muestra su número de nodo, así como el puerto donde esta conectado.

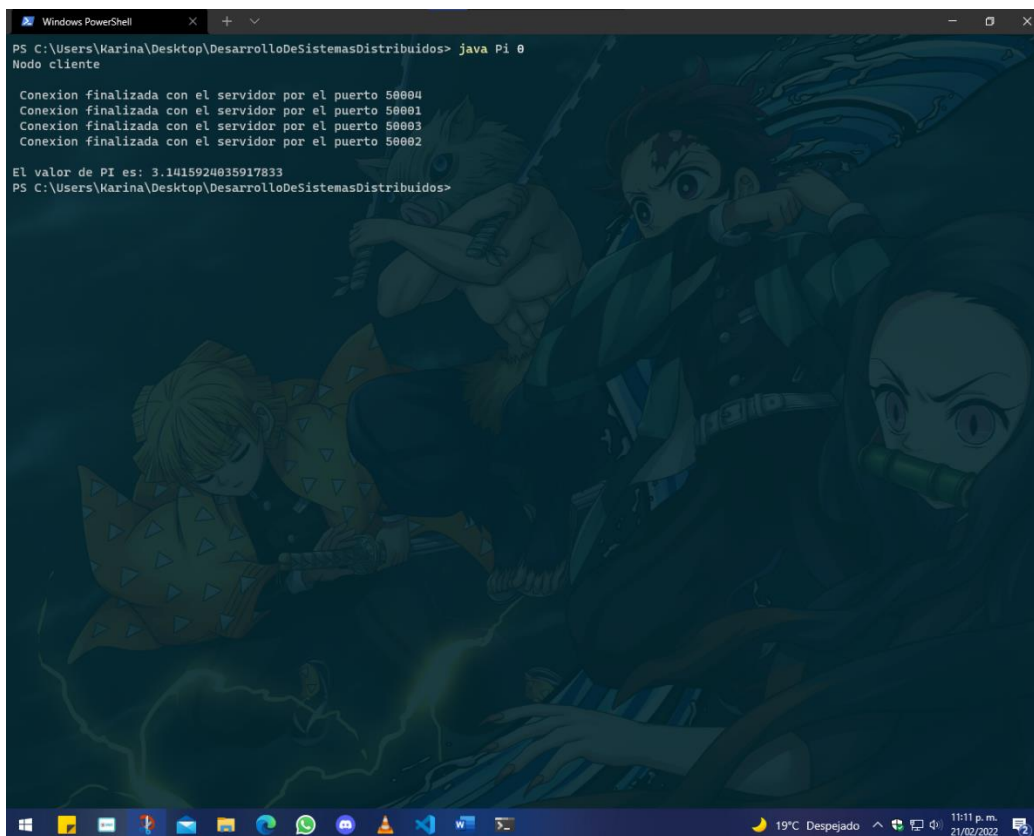


Ilustración 7

Vista del Cliente

En la ilustración 7 se muestra tanto las conexiones con cada servidor y en que puerto se hicieron, así como el resultado final del cálculo de Pi.

```

PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos> java P
i 1
Nodo Servidor 1
Puerto: 50001
PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos>

PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos> java P
i 2
Nodo Servidor 2
Puerto: 50002
PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos>

PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos> java Pi 0
Nodo cliente
Conexion finalizada con el servidor por el puerto 50004
Conexion finalizada con el servidor por el puerto 50001
Conexion finalizada con el servidor por el puerto 50003
Conexion finalizada con el servidor por el puerto 50002
El valor de PI es: 3.1415924035917833
PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos>

PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos> java P
i 3
Nodo Servidor 3
Puerto: 50003
PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos>

PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos> java P
i 4
Nodo Servidor 4
Puerto: 50004
PS C:\Users\Karina\Desktop\DesarrolloDeSistemasDistribuidos>
  
```

Ilustración 8

Vista de todas las pantallas en ejecución

En la ilustración 8 se muestra la topología inicial en ejecución, con una aproximación de

$$\pi \approx 3.1415924035917833$$

CONCLUSIONES

Esta práctica ayudo a reforzar el conocimiento adquirido en previas unidades de aprendizaje tanto de la parte de programación en los temas de sockets con hilos como la parte matemática, resulto interesante hacer el calculo de un numero que usamos muchas veces en ingeniería desde otro enfoque mediante la utilización de las series infinitas de Gregory-Leibniz.