



INSTITUTO POLITÉCNICO NACIONAL.
ESCUELA SUPERIOR DE CÓMPUTO.



Sistemas Distribuidos

Tarea 01: Reporte de “Cálculo de PI”

Alumno: Oaxaca Pérez David Arturo

Grupo:

4CV12

A cargo del profesor:

PINEDA GUERRERO CARLOS

Contenido

Introducción	3
Desarrollo	4
Partes desarrolladas de los algoritmos requeridos.....	7
Captura de pantalla mostrando como se compilo el programa	8
Captura de pantalla corriendo los cinco programas pasando el número de nodo como parámetro	9
Resultado del programa una vez que terminan los cinco threads.....	12
Conclusiones	13

Introducción

La serie de Gregory-Leibniz permite calcular el valor de PI mediante una serie de sumas:

$$\pi = 4 \left(\sum_{k=1}^{\infty} \frac{(-1)^{(k+1)}}{(2k-1)} \right)$$
$$= 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right)$$

Imagen 3. Serie de Gregory-Leibniz.

Esta tarea consiste en implementar un programa con topología de estrella donde cada nodo tendrá un identificador, siendo el nodo 0 el que actuara como servidor y los nodos 1, 2, 3 y 4 actuarán como clientes.

El identificador del nodo se asignará mediante los parámetros al ejecutar el programa.

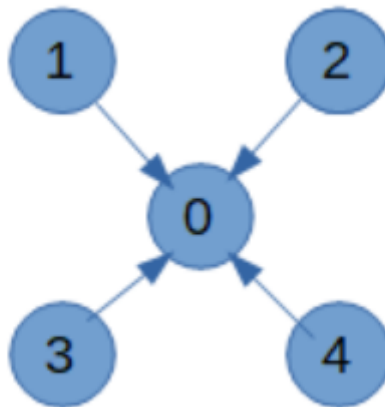


Imagen 2. Topología de estrella.

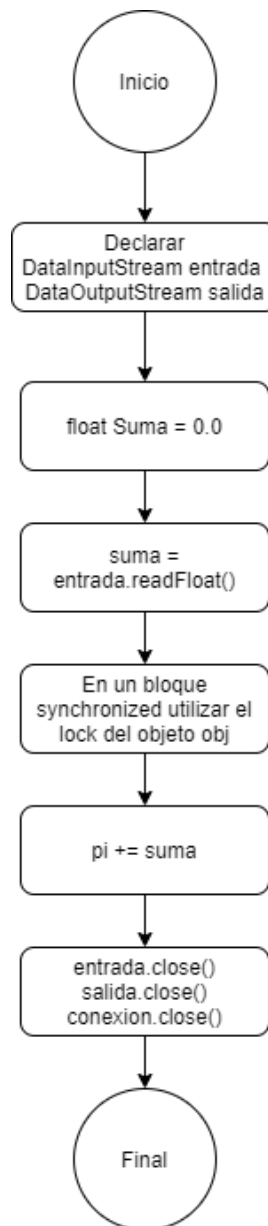
La forma en la que esto se hará es que cada nodo cliente será aportar a esta sumatoria, los nodos pares aportaran las fracciones de valor negativo que podemos observar en la imagen de la serie, mientras que los impares aportaran la parte positiva. Se usará un bloque synchrnized para qué la sumatoria se realice completa como se vio en la actividad anterior donde se sumaban las iteraciones de dos ciclos a una variable estática hasta llegar a 200 usando un lock en el bloque synchronize.

Desarrollo

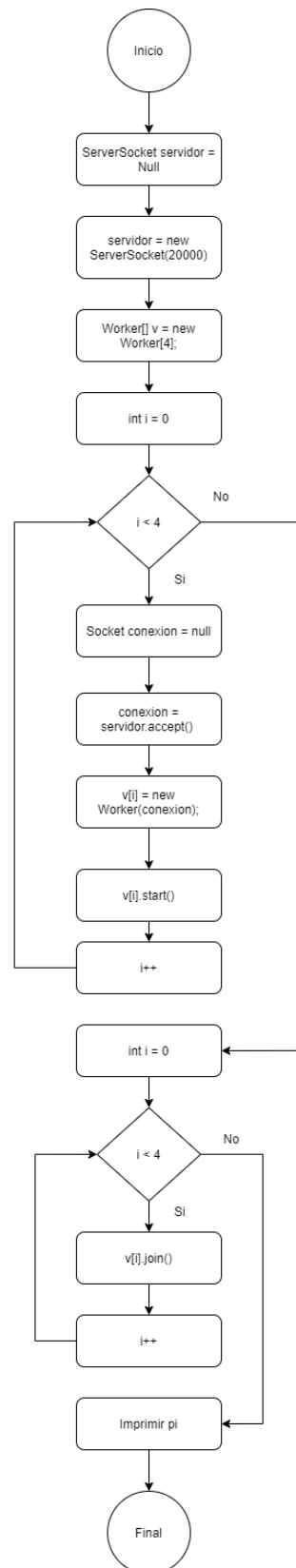
Para el desarrollo usamos el esqueleto proporcionado por la asignación de la tarea, este ya consiste en los métodos `main()` y `run()` que serán los más importantes para empezar a construir este proyecto y hace falta la construcción de los algoritmos que darán funcionalidad al programa, tanto el que es de cliente, como el de servidor más sus respectivas tareas, para esto solo será necesario desarrollar los algoritmos detallados en los pasos también proporcionados por la asignación los cuales se verán en la siguiente parte.

Los algoritmos mencionados se pueden ver en los siguientes diagramas:

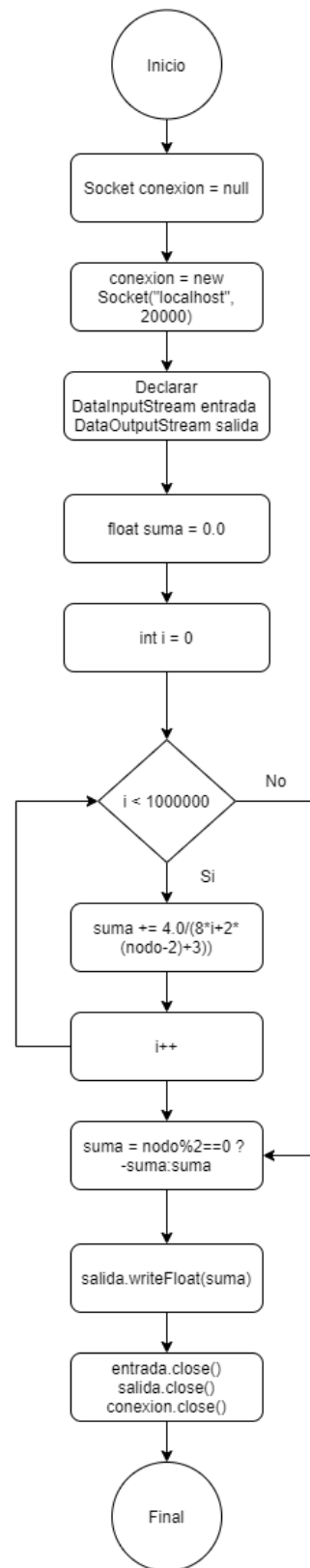
Algoritmo 1:



Algoritmo 2:

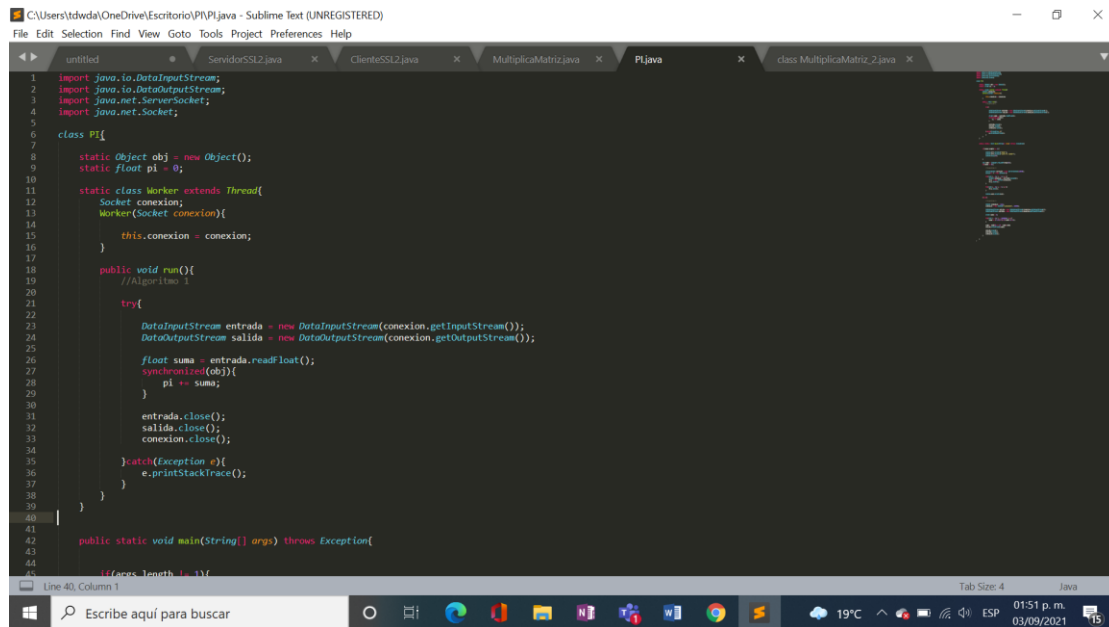


Algoritmo 3:



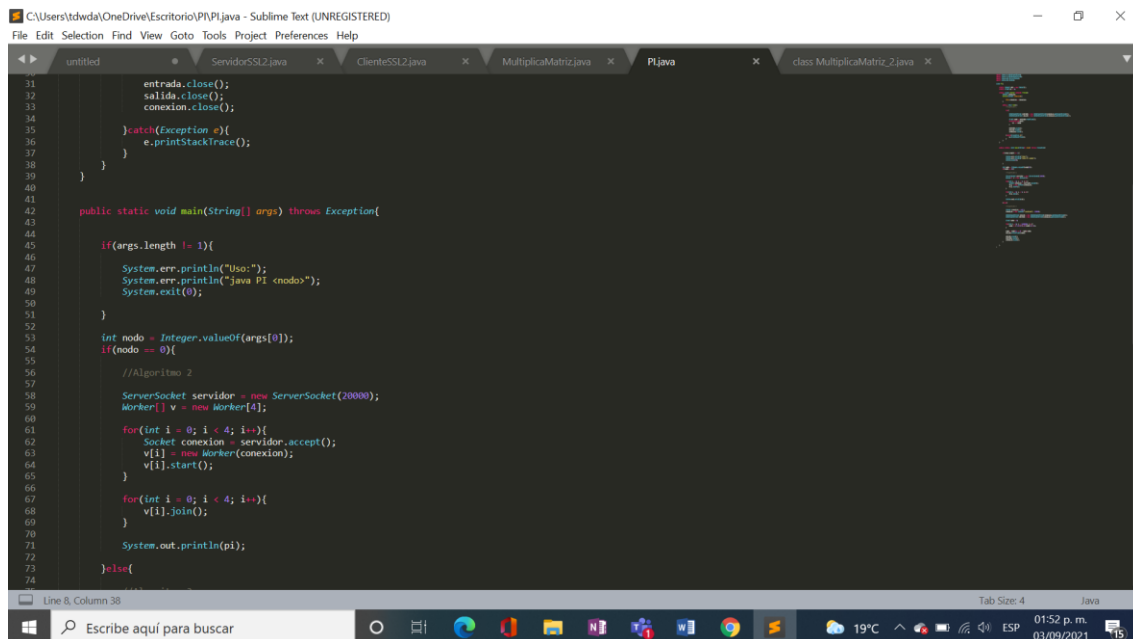
Partes desarrolladas de los algoritmos requeridos

Captura de pantalla mostrando el desarrollo del algoritmo 1 en código:



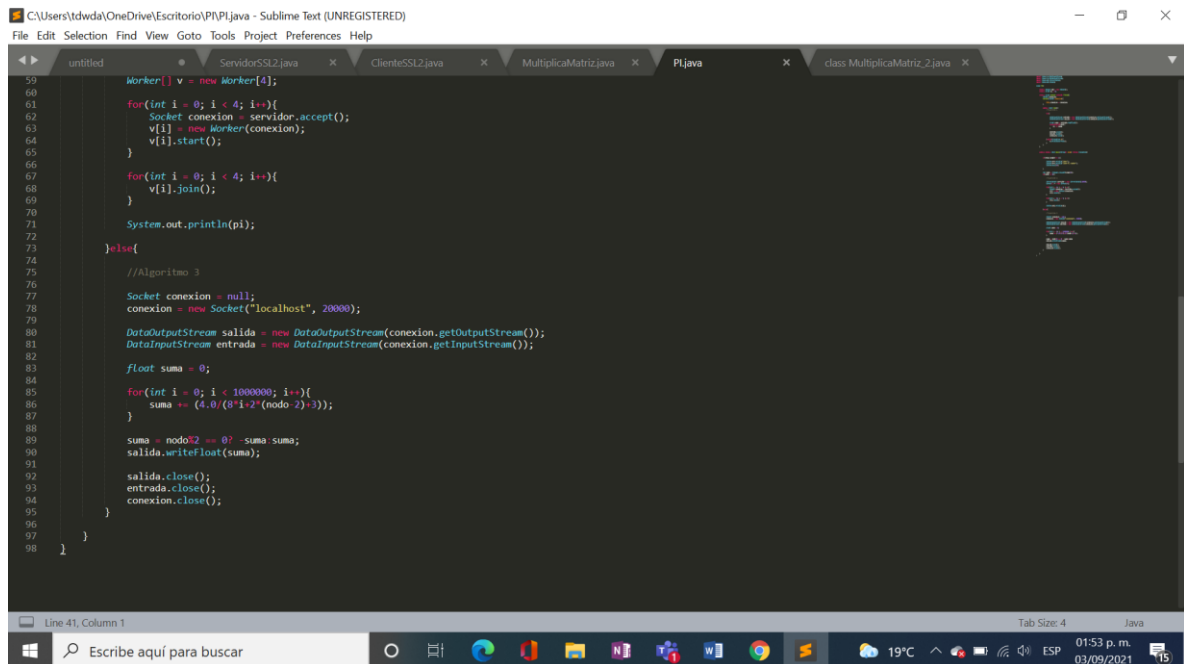
```
1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5
6 class PI{
7
8     static Object obj = new Object();
9     static float pi = 0;
10
11     static class Worker extends Thread{
12         Socket conexion;
13         Worker(Socket conexion){
14             this.conexion = conexion;
15         }
16
17         public void run(){
18             //Algoritmo 1
19
20             try{
21
22                 DataInputStream entrada = new DataInputStream(conexion.getInputStream());
23                 DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
24
25                 float suma = entrada.readFloat();
26                 synchronized(obj){
27                     pi += suma;
28                 }
29
30                 entrada.close();
31                 salida.close();
32                 conexion.close();
33             }catch (Exception e){
34                 e.printStackTrace();
35             }
36         }
37     }
38
39     public static void main(String[] args) throws Exception{
40
41         if(args.length != 1){
42
43             System.err.println("Uso:");
44             System.err.println("java PI <nodo>");
45             System.exit(0);
46         }
47
48         int nodo = Integer.valueOf(args[0]);
49         if(nodo == 0){
50             //Algoritmo 2
51
52             ServerSocket servidor = new ServerSocket(20000);
53             Worker[] w = new Worker[4];
54
55             for(int i = 0; i < 4; i++){
56                 Socket conexion = servidor.accept();
57                 w[i] = new Worker(conexion);
58                 w[i].start();
59             }
60
61             for(int i = 0; i < 4; i++){
62                 w[i].join();
63             }
64
65             System.out.println(pi);
66         }else{
67
68             //Algoritmo 3
69
70             Socket conexion = new Socket("localhost", 20000);
71             DataInputStream entrada = new DataInputStream(conexion.getInputStream());
72             DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
73
74             float suma = entrada.readFloat();
75             synchronized(obj){
76                 pi += suma;
77             }
78
79             entrada.close();
80             salida.close();
81             conexion.close();
82         }
83     }
84 }
```

Captura de pantalla mostrando el desarrollo del algoritmo 2 en código:



```
31         entrada.close();
32         salida.close();
33         conexion.close();
34     }
35     }catch (Exception e){
36         e.printStackTrace();
37     }
38 }
39
40 public static void main(String[] args) throws Exception{
41
42     if(args.length != 1){
43
44         System.err.println("Uso:");
45         System.err.println("java PI <nodo>");
46         System.exit(0);
47     }
48
49     int nodo = Integer.valueOf(args[0]);
50     if(nodo == 0){
51         //Algoritmo 2
52
53         ServerSocket servidor = new ServerSocket(20000);
54         Worker[] w = new Worker[4];
55
56         for(int i = 0; i < 4; i++){
57             Socket conexion = servidor.accept();
58             w[i] = new Worker(conexion);
59             w[i].start();
60         }
61
62         for(int i = 0; i < 4; i++){
63             w[i].join();
64         }
65
66         System.out.println(pi);
67     }else{
68         //Algoritmo 3
69
70         Socket conexion = new Socket("localhost", 20000);
71         DataInputStream entrada = new DataInputStream(conexion.getInputStream());
72         DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
73
74         float suma = entrada.readFloat();
75         synchronized(obj){
76             pi += suma;
77         }
78
79         entrada.close();
80         salida.close();
81         conexion.close();
82     }
83 }
```

Captura de pantalla mostrando el desarrollo del algoritmo 3 en código:



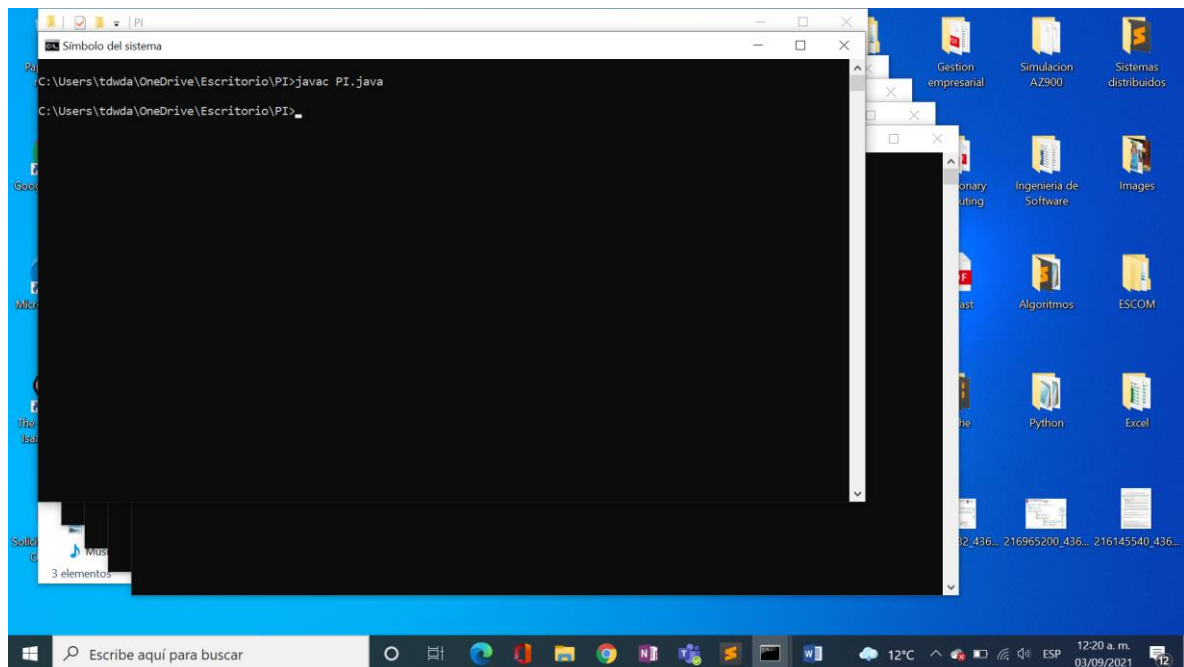
The screenshot shows a Sublime Text editor window with the title bar "C:\Users\tdwda\OneDrive\Escritorio\PI\PI.java - Sublime Text (UNREGISTERED)". The editor has several tabs open: "untitled", "ServidorSSL2.java", "ClienteSSL2.java", "MultiplicaMatriz.java", "PI.java", and "class MultiplicaMatriz_2.java". The active tab is "PI.java", which contains Java code for a server and client application. The code includes a `Worker` class, a `main` method, and a `run` method. The `run` method implements a client that connects to a server on `localhost:20000`, sends a request, and receives a response. The code is as follows:

```
50 Worker[] v = new Worker[4];
51
52 for(int i = 0; i < 4; i++){
53     Socket conexion = servidor.accept();
54     v[i] = new Worker(conexion);
55     v[i].start();
56 }
57
58 for(int i = 0; i < 4; i++){
59     v[i].join();
60 }
61
62 System.out.println(pi);
63
64 }else{
65
66     //Algoritmo 3
67
68     Socket conexion = null;
69     conexion = new Socket("localhost", 20000);
70
71     DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
72     DataInputStream entrada = new DataInputStream(conexion.getInputStream());
73
74     float suma = 0;
75
76     for(int i = 0; i < 1000000; i++){
77         suma += (4.0/(8*i+2*(nodo-2)+3));
78     }
79
80     suma = nodo%2 == 0? -suma : suma;
81     salida.writeFloat(suma);
82
83     salida.close();
84     entrada.close();
85     conexion.close();
86 }
87
88 }
```

Estas capturas de pantalla fueron tomadas posteriormente para la elaboración del reporte.

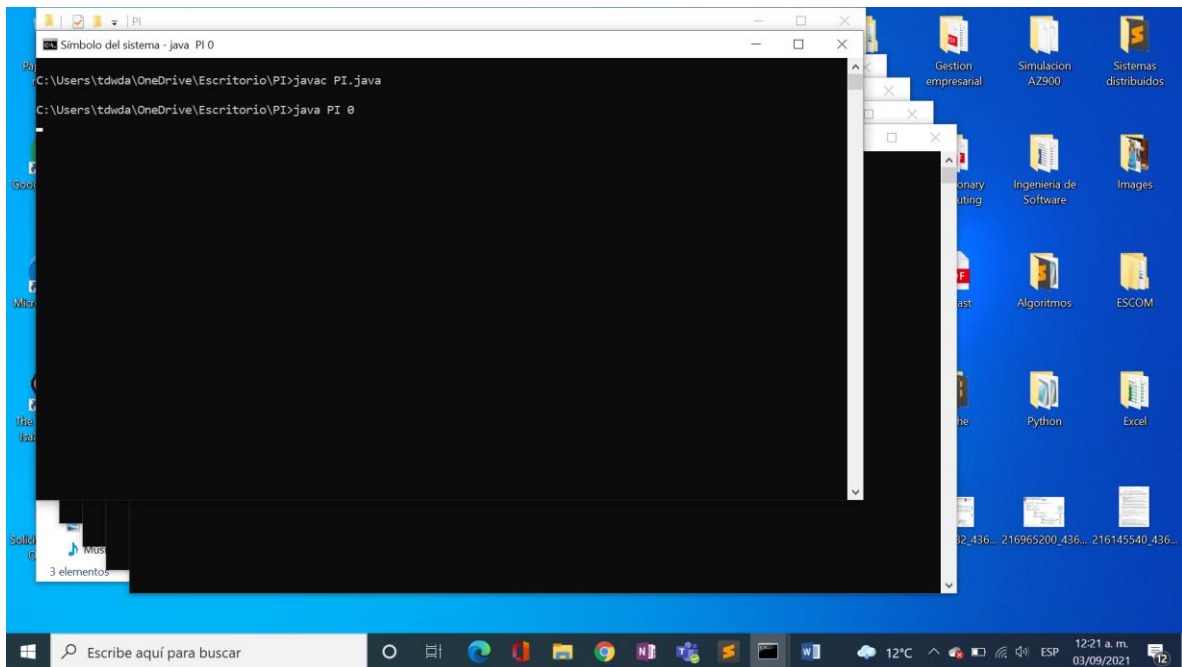
Posteriormente al terminar con la construcción del programa compilamos el programa y abrimos 5 ventanas de la línea de comando para poder ejecutarlo posteriormente.

Captura de pantalla mostrando como se compilo el programa

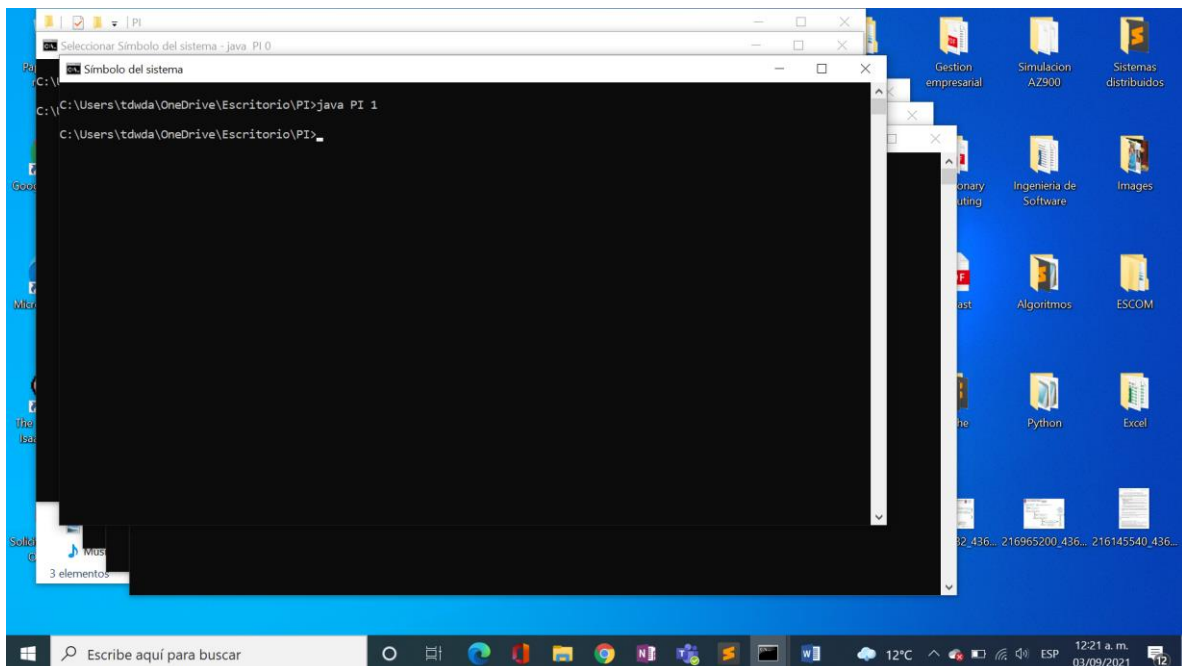


Captura de pantalla corriendo los cinco programas pasando el número de nodo como parámetro

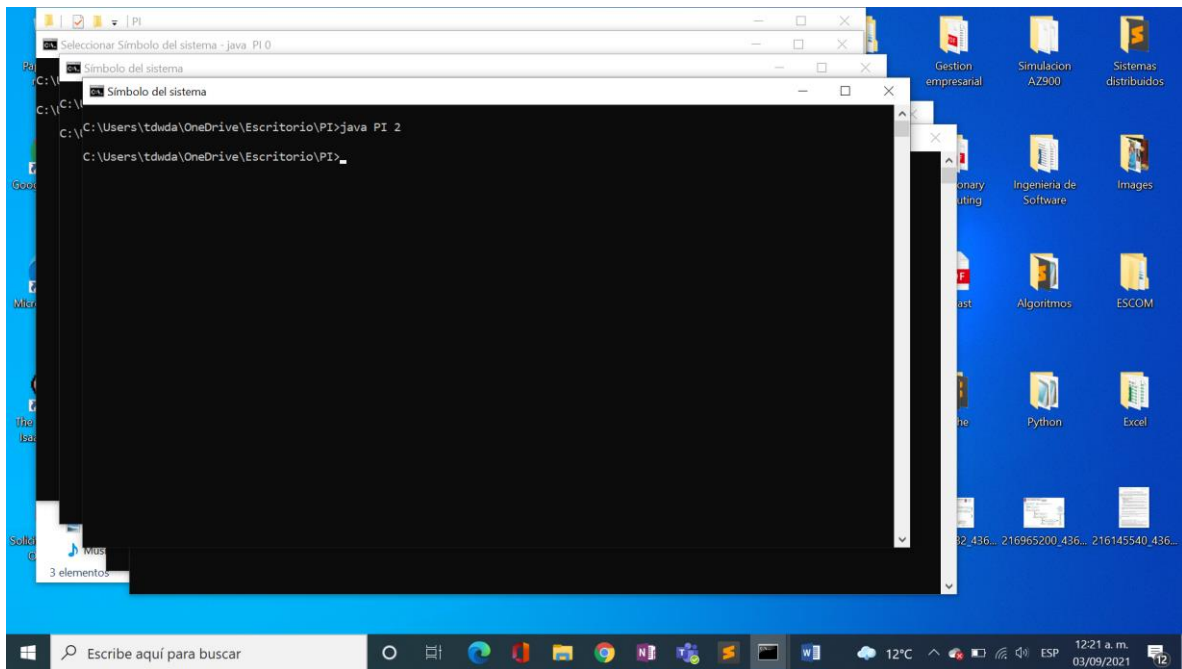
Nodo 0 (Servidor):



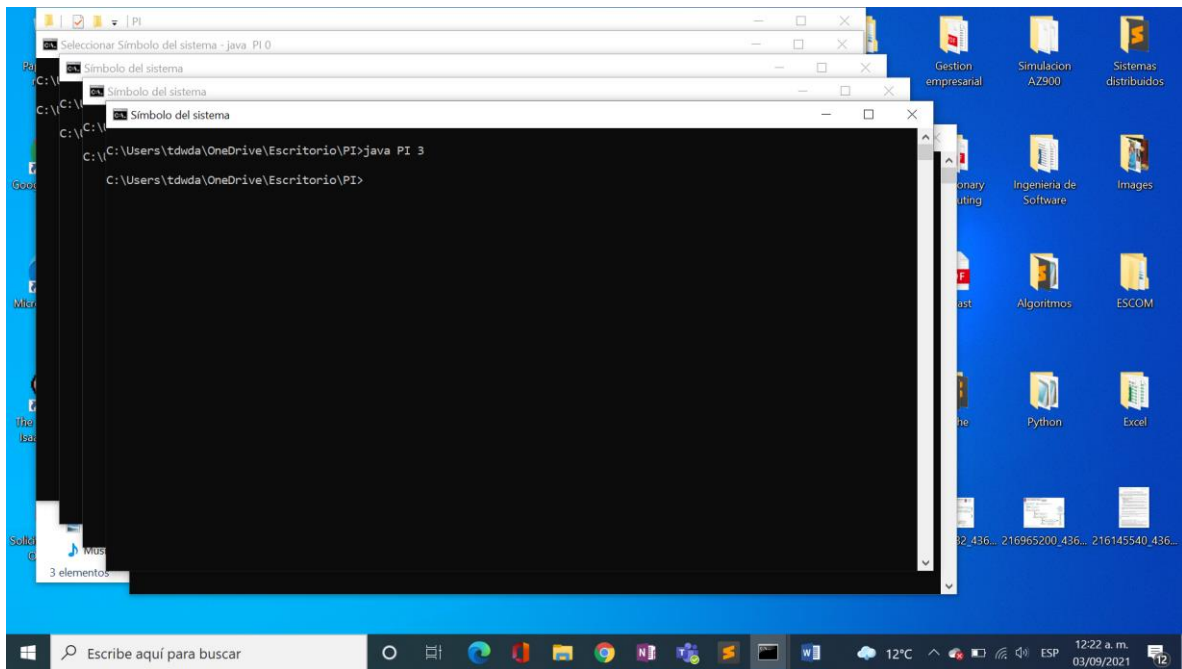
Nodo 1 (Cliente):



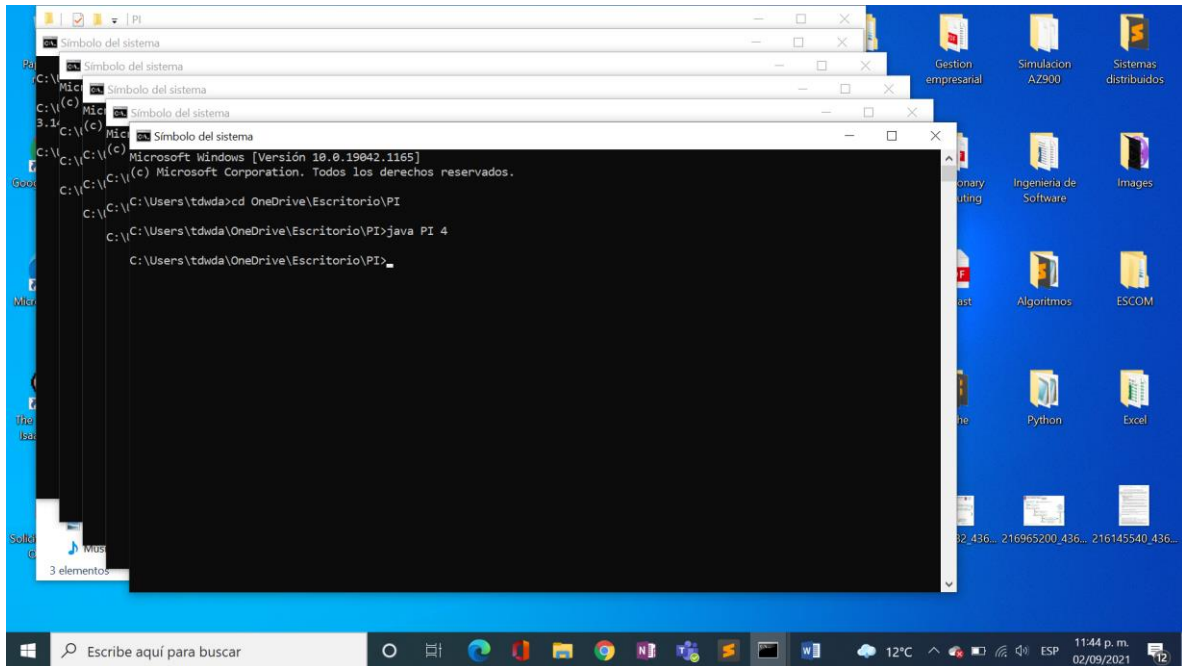
Nodo 2 (Cliente):



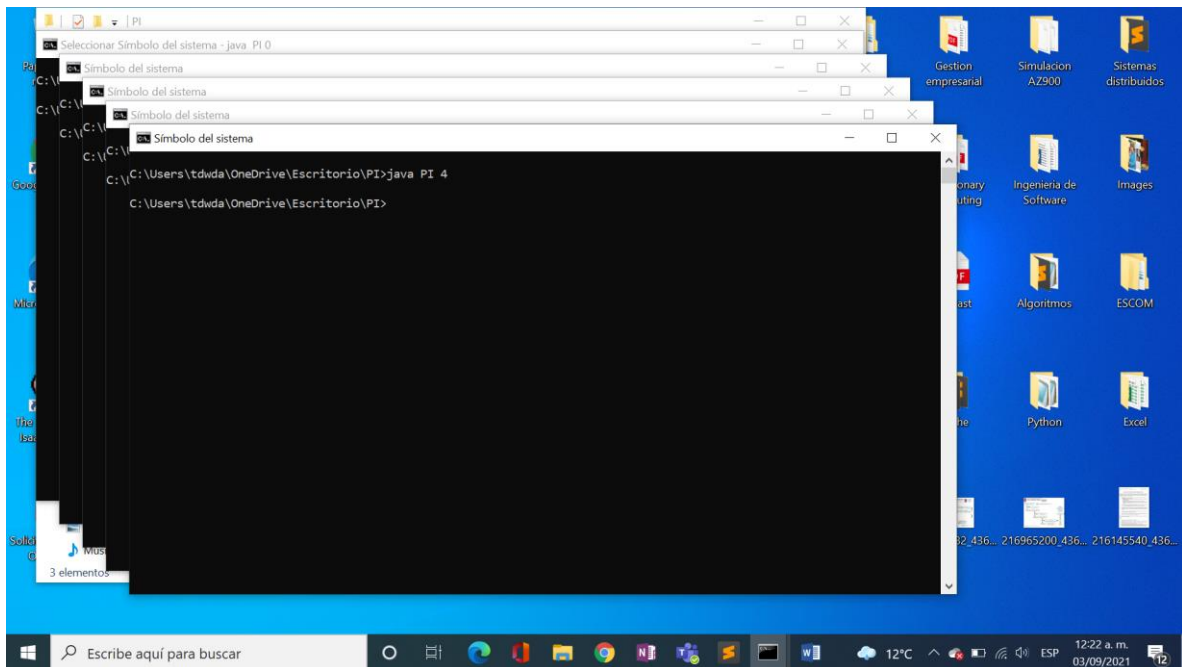
Nodo 3 (Cliente):



Nodo 4 (Cliente):

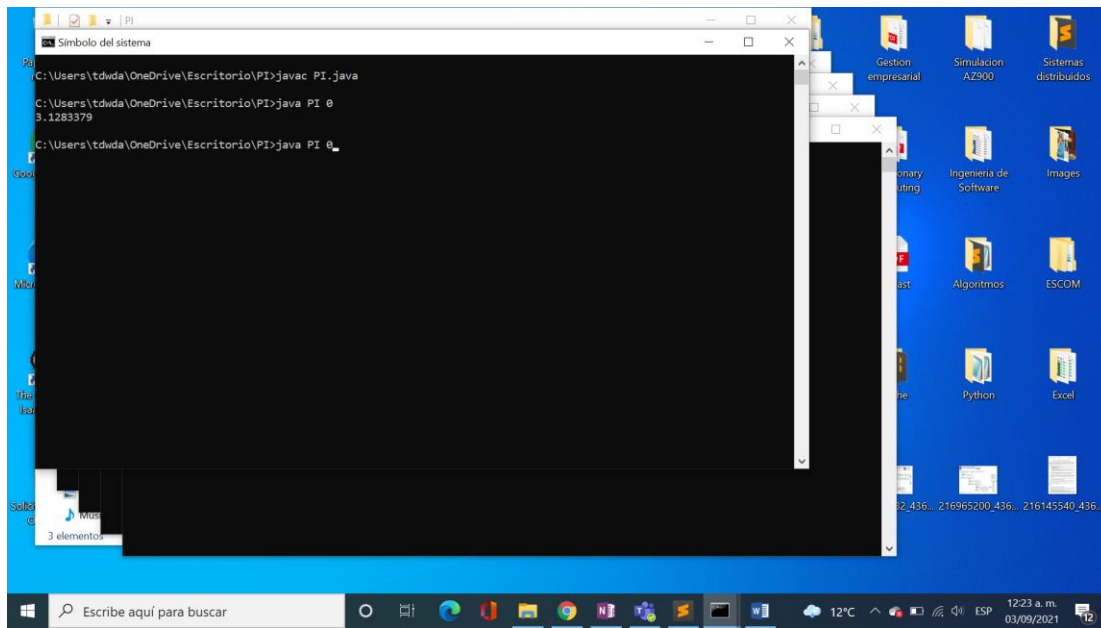


Nodo 4 (Cliente):



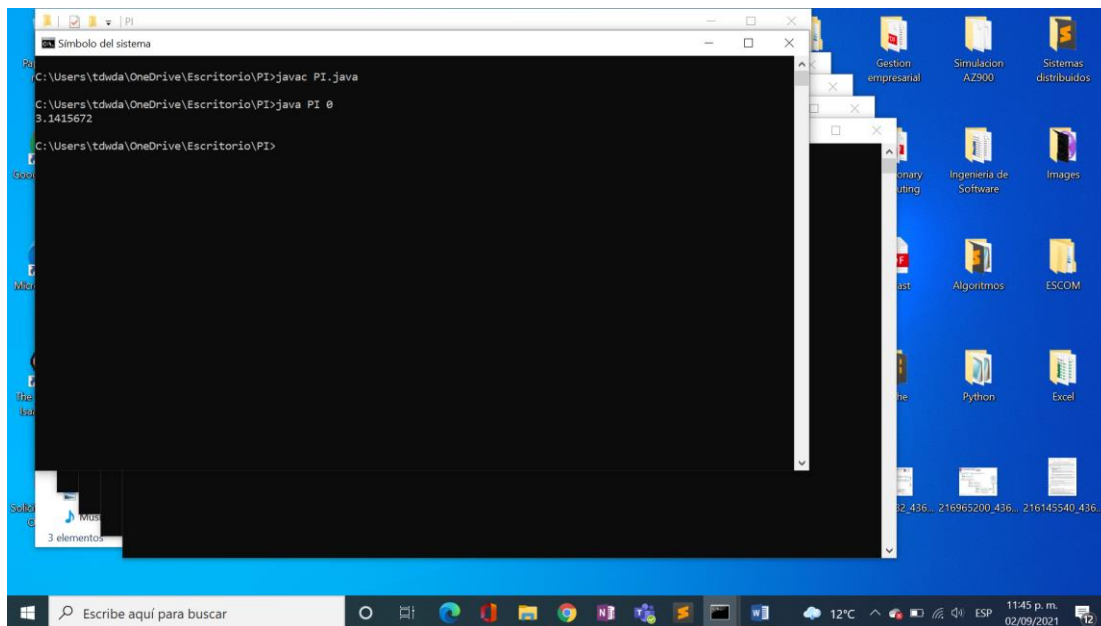
Resultado del programa una vez que terminan los cinco threads

Probando el programa con 1,000,000 millones de términos para la serie de Gregory-Leibniz como indica el algoritmo tres obtenemos este resultado:



```
Símbolo del sistema
C:\Users\tdwda\OneDrive\Escritorio\PI>javac PI.java
C:\Users\tdwda\OneDrive\Escritorio\PI>java PI 0
3.1283379
C:\Users\tdwda\OneDrive\Escritorio\PI>java PI 0_
2,436... 216965200,436... 216145540,436...
```

Haciendo algunas pruebas previamente, con 10,000 iteraciones se obtiene un resultado más parecido a pi.



```
Símbolo del sistema
C:\Users\tdwda\OneDrive\Escritorio\PI>javac PI.java
C:\Users\tdwda\OneDrive\Escritorio\PI>java PI 0
3.1415672
C:\Users\tdwda\OneDrive\Escritorio\PI>java PI 0_
2,436... 216965200,436... 216145540,436...
```

Conclusiones

Fue una práctica interesante para realizar, pues explica bastante bien como utilizar la topología de estrella usando hilos, además de que profundiza en el tema de sincronizar hilos que previamente habíamos visto en clase, mostrando un uso que ilustra mejor como funciona, siendo este el cálculo de PI, si acaso, surgieron dudas con los resultados obtenidos, ya que al usar 1,000,000 de iteraciones el resultado se empezaba a alejar un poco del resultado deseado, mientras que probando un poco más bajos se acerca un poco más, fuera de eso, es una práctica que ayudo a comprender mejor como es el uso de sockets con hilos.