Instituto Politécnico Nacional

Escuela Superior de Cómputo

CÁLCULO DE PI

TAREA 1

Materia:

Desarrollo De Sistemas Distribuidos

Grupo:

4CV14

Profesor:

Pineda Guerrero Carlos

Alumno:

Castro Cruces Jorge Eduardo

Boleta:

2015080213

Fecha:

Viernes, 3 de septiembre de 2021

1. Desarrollo del programa

En esta tarea vamos a desarrollar un programa distribuido, el cual calculará una aproximación de PI utilizando la serie de Gregory-Leibniz.

La serie tiene la siguiente forma: 4/1-4/3+4/5-4/7+4/9-4/11+4/13-4/15 ...

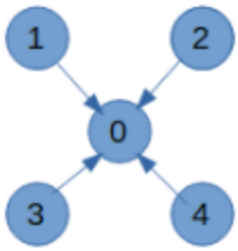
Notar que los denominadores son los números impares: 1, 3, 5, 7, 9, 11, 13 ...

El programa va a ejecutar en forma distribuida sobre cinco nodos, cada nodo sería una computadora diferente.

Vamos a 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).

Cada nodo (excepto el nodo 0) deberá calcular un millón de términos de la serie.

Implementaremos la siguiente topología lógica de tipo estrella, cada nodo se ha identificado con un número entero:



El nodo 0 actuará como servidor y los nodos 1, 2, 3 y 4 actuarán como clientes.

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 actual como servidor o como cliente, según el número de nodo que pasa como parámetro.

Se deberá implementar el siguiente programa:

```
class PI
{
  static Object obj= new Object();
  static float pi = 0;
  static class Worker extends Thread
  {
    Socket conexion;
    Worker(Socket conexion)
    {
      this.conexion = conexion;
```

```
    }
    public void run()
    {
      // Algoritmo 1
    }
  }
  public static void main(String[] args) throws Exception
  {
    if (args.length != 1)
    {
      System.err.println("Uso:");
      System.err.println("java PI <nodo>");
      System.exit(0);
    }
    int nodo = Integer.valueOf(args[0]);
    if (nodo == 0)
    {
      // Algoritmo 2
    }
    else
    {
      // Algoritmo 3
    }
  }
}
```

Se deberá implementar los siguientes algoritmos:

Algoritmo 1

1. Crear los streams de entrada y salida.
2. Declarar la variable **suma** de tipo float.
3. Recibir en la variable **suma** la suma calculada por el cliente.
4. En un bloque synchronized utilizar el lock del objeto **obj**:
4.1 Asignar a la variable **pi** la expresión: suma+pi
5. Cerrar los streams de entrada y salida.
6. Cerrar la conexión **conexion**.

Algoritmo 2

1. Declarar una variable **servidor** de tipo ServerSocket.
2. Crear un socket servidor utilizando el puerto 40000 y asignarlo a la variable **servidor**.
3. Declarar un vector **v** de tipo Worker con 4 elementos.
4. Declarar una variable entera **i** y asignarle cero.
5. En un ciclo:

    - 5.1 Si la variable **i** es igual a 4, entonces salir del ciclo.
    - 5.2 Declarar una variable **conexion** de tipo Socket.

- 5.3 Invocar el método servidor.accept() y asignar el resultado a la variable **conexion**.
- 5.4 Crear una instancia de la clase Worker, pasando como parámetro la variable **conexion**. Asignar la instancia al elemento v[i].
- 5.5 Invocar el método v[i].start()
- 5.6 Incrementar la variable **i**.
- 5.7 Ir al paso 5.1

6. Declarar una variable **i** entera y asignarle cero.
7. En un ciclo:

- 7.1 Si la variable **i** es igual a 4, entonces salir del ciclo.
- 7.2 Invocar el método v[i].join()
- 7.3 Incrementar la variable **i**.
- 7.4 Ir al paso 7.1

8. Desplegar el valor de la variable **pi**.

Algoritmo 3

1. Declarar la variable **conexion** de tipo Socket y asignarle null.
2. Realizar la conexión con el servidor (localhost) a través del puerto 40000 implementando re-intento. Asignar el socket a la variable **conexion**.
3. Crear los streams de entrada y salida.
4. Declarar la variable **suma** de tipo float y asignarle cero.
5. Declarar una variable **i** de tipo entero y asignarle cero.
6. En un ciclo:

- 6.1 Si la variable **i** es igual a 1000000, entonces salir del ciclo.
- 6.2 Asignar a la variable **suma** la expresión: 4.0/(8*i+2*(nodo-2)+3)+suma
- 6.3 Incrementar la variable **i**.
- 6.4 Ir al paso 6.1

7. Asignar a la variable **suma** la expresión: nodo%2==0?-suma:suma
8. Enviar al servidor el valor de la variable **suma**.
9. Cerrar los streams de entrada y salida.
10. Cerrar la conexión **conexion**.

Notar que el algoritmo 1 se deberá ejecutar dentro de un bloque try.

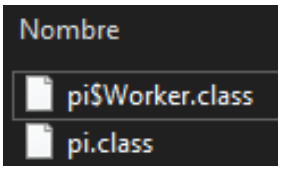
También se debe notar que la variable **obj** debe ser estática para que todos los threads accedan al mismo lock.

2. Pruebas de escritorio

Primero compilamos el programa:

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>javac pi.java
```

Se van a crear dos archivos:



Ahora, ejecutamos la clase Pi y le enviamos como parámetro el numero del nodo a ejecutar:

Nodo 0: Servidor

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi 0
```

Nodo 1: Cliente 1

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi 1
```

Nodo 2: Cliente 2

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi 2
```

Nodo 3: Cliente 3

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi 3
```

Nodo 4: Cliente 4

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi 4
```

En caso de no enviarle como parametro algúnos de los numeros validos, muestra el siguiente error:

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi
Uso:
java PI <nodo>
```

Primero debe estár ejecutandose el servidor y va a estar en espera de que los clientes le envíen la aproximación correspondiente:

Para que finalmente, el servidor sume cada una des las 4 aproximaciones:

```
C:\Users\georg\Desktop\ESCOM\8vo Semestre\Distribuidos\Tareas\Tarea1>java pi 0
Recibi la suma calculada por el cliente: 11.10200142313125
Recibi la suma calculada por el cliente: -8.284754741054094
Recibi la suma calculada por el cliente: 7.634109723770362
Recibi la suma calculada por el cliente: -7.3097640022557355
El valor de la variable pi es: 3.141592403591784
```

3. Conclusiones

Puesto que es un sistema distribuido, no importa el orden en que se ejecuten los nodos, siempre y cuando se ejecuten los 3 y no se repita ninguno para que funcione correctamente.

El problema llega cuando ejecutamos más de una vez algún cliente, porque la suma de las aproximaciones no va a ser correcta.

También, pude notar que si se ejecuta el mismo cliente, siempre hará la misma suma, sin importar el número de veces que se ejecute.