3-9-2021

Desarrollo de Sistemas Distribuidos

Pineda Guerrero Carlos

Aproximación de Pi

Martínez Coronel Brayan Yosafat

De acuerdo con la aproximación Gregory-Leibniz, pi tiene un valor dado por la siguiente expresión:

4/1 + 4/3 + 4/5 + 4/7 + …

En esta tarea el objetivo es obtener la aproximación de pi mediante el uso de Sockets, siguiendo la siguiente estructura:

Un dibujo de una persona

Descripción generada automáticamente con confianza baja

Donde el nodo 0 funciona como servidor y el resto calcula un millón de términos de la serie.

# Código

import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
public class Pi {  
 static final Object *obj* = new Object();  
 static float *pi* = 0;  
  
 static class Worker extends Thread {  
 Socket conexion;  
  
 Worker(Socket conexion) {  
 this.conexion = conexion;  
 }  
  
 @Override  
 public void run() {  
 try {  
 DataInputStream entrada = new DataInputStream(conexion.getInputStream());  
 DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());  
 float suma = entrada.readFloat();  
  
 synchronized (*obj*) {  
 *pi* = suma + *pi*;  
 }  
  
 entrada.close();  
 salida.close();  
 conexion.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 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.*parseInt*(args[0]);  
  
 if (nodo == 0) {  
 ServerSocket servidor = new ServerSocket(10000);  
 Worker[] v = new Worker[4];  
  
 for (int i = 0; i < 4; i++) {  
 Socket conexion = servidor.accept();  
 v[i] = new Worker(conexion);  
 v[i].start();  
 }  
  
 for (int i = 0; i < 4; i++) {  
 v[i].join();  
 }  
  
 System.*out*.println(*pi*);  
 } else {  
 Socket conexion;  
  
 while (true) {  
 try {  
 conexion = new Socket("localhost", 10000);  
 break;  
 } catch (IOException e) {  
 Thread.*sleep*(100);  
 }  
 }  
  
 DataInputStream entrada = new DataInputStream(conexion.getInputStream());  
 DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());  
 float suma = 0;  
  
 for (int i = 0; i < 1000000; i++) {  
 suma = (float) (4.0 / (8 \* i + 2 \* (nodo - 2) + 3) + suma);  
 }  
  
 suma = nodo % 2 == 0 ? -suma : suma;  
 salida.writeFloat(suma);  
 entrada.close();  
 salida.close();  
 conexion.close();  
 }  
 }  
}

# Configuraciones

Como se puede observar, el programa necesita que se le pase el número de nodo que va a correr. Se muestra la configuración para un nodo, el resto básicamente son iguales, solamente cambia el número.

Captura de pantalla de computadora

Descripción generada automáticamente

# Resultado

A continuación, se muestra lo que nos retornó después de correr todos los nodos. Primero que los nodos del 1 al 4 están corriendo ya, y al final se muestra cuando se llama al nodo 0.

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

# Conclusión

La aproximación no es realmente la mejor, pero si modificamos el tipo de dato de flotante a doble precisión, entonces obtenemos lo siguiente:

Captura de pantalla de computadora

Descripción generada automáticamente

Con este tenemos la certeza de que el programa funciona.