Sanchez Rico Raúl

4CV2  
Desarrollo de sistemas distribuidos  
ESCOM

Calculo distribuido de pi

Tarea 1

**Desarrollo**

Compilar al programa PI.java de la siguiente manera: *javac PI.java,* como se muestra en la ilustración 1.

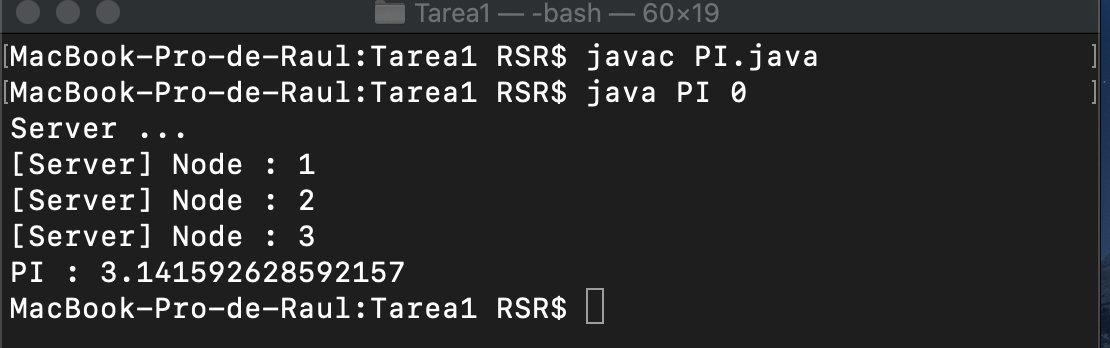


Ilustración 1 Compilación del programa PI.java

Para poder correr el programa se necesitan de los 4 nodos, en este caso son 4 terminales, cada una, con un numero de nodo diferente, siendo el Nodo 0, el servidor y los Nodos 1, 2 y 3, los clientes, se usa el comando: *java PI #Nodo.* Las ilustraciones 2, 3 y 4 muestran el funcionamiento de estos nodos.

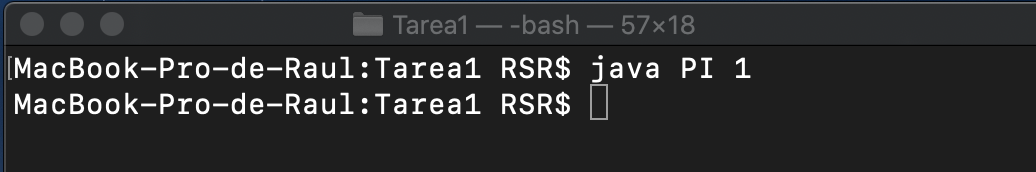


Ilustración 2 Nodo 1

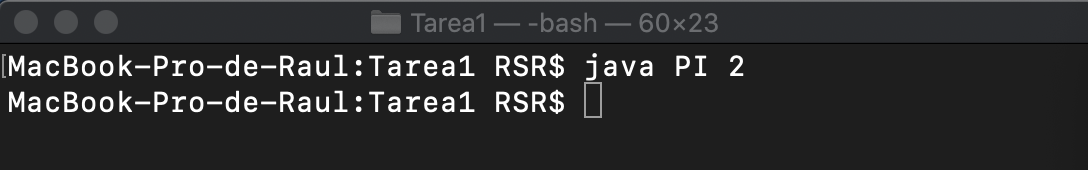


Ilustración 3 Nodo 2

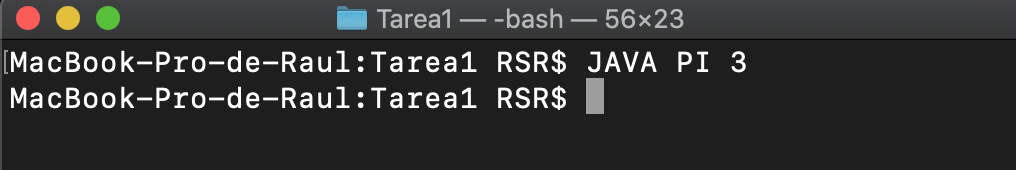


Ilustración 4 Nodo 3

La ilustración 5, muestra el nodo 0, siendo este el servidor, el cual va a esperar a que los 3 nodos se conecten a el para poder terminar el proceso. Tal y como se ve ene esta figura. Para poder dar como resultado, la aproximación de PI mediante la la serie Gregory-Leibniz.

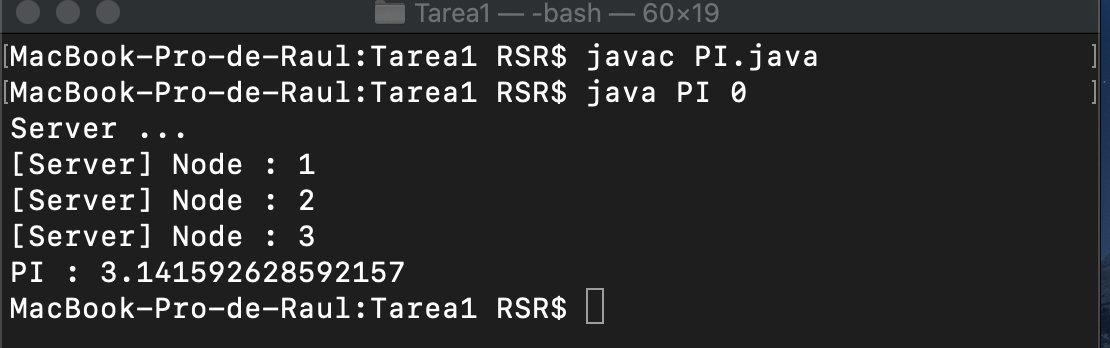


Ilustración 5 Nodo 0 - Servidor

**Código**

1. **import** java.net.Socket;
2. **import** java.net.ServerSocket;
3. **import** java.nio.ByteBuffer;
4. **import** java.lang.Thread;
5. **import** java.io.DataOutputStream;
6. **import** java.io.DataInputStream;
8. **class** PI {
9. **static** Object lock = **new** Object();
10. **static** **double** pi = 0;
11. **static** **class** Worker **extends** Thread{
12. Socket conexion;
13. Worker(Socket conexion) {
14. **this**.conexion = conexion;
15. }
16. **public** **void** run(){
17. **try**{
18. DataOutputStream salida = **new** DataOutputStream(conexion.getOutputStream());
19. DataInputStream  entrada = **new** DataInputStream(conexion.getInputStream());
20. **double** x = entrada.readDouble();
21. **synchronized**(lock){
22. pi += x;
23. }
25. salida.close();
26. entrada.close();
27. conexion.close();
29. }**catch**(Exception e){
30. System.err.println(e.getMessage());
31. }
32. }
33. }
35. **public** **static** **void** main(String[] args) **throws** Exception{
36. **if** (args.length != 1){
37. System.err.println("Uso:");
38. System.err.println("java PI <nodo>");
39. System.exit(0);
40. }
41. **int** nodo = Integer.valueOf(args[0]);
42. **if** (nodo == 0){
43. System.out.println("Server ...");
44. ServerSocket servidor = **new** ServerSocket(50000);
45. Worker w[] = **new** Worker[3];
46. **int** i = 0;
47. **while**(i < 3){
48. Socket conexion = servidor.accept();
49. w[i]= **new** Worker(conexion);
50. w[i].start();
51. i++;
52. System.out.println("[Server] Node : "+ i);
53. }
54. **double** suma = 0;
55. i = 0;
56. **while**(i <10000000){
57. suma += 4.0/(8\*i+1);
58. i++;
59. }
60. **synchronized**(lock){
61. pi += suma;
62. }
63. i = 0;
64. **while**(i<3){
65. w[i].join();
66. i++;
67. }
68. System.out.println("PI : "+pi);
69. }
70. **else**{
71. Socket conexion = **null**;
72. **for**(;;)
73. **try**{
74. conexion = **new** Socket("localhost",50000);
75. **break**;
76. }
77. **catch** (Exception e){ Thread.sleep(100);
78. }
79. DataOutputStream salida = **new** DataOutputStream(conexion.getOutputStream());
80. DataInputStream entrada = **new** DataInputStream(conexion.getInputStream());
81. **double** suma = 0;
82. **int** i = 0;
83. **while**(i < 10000000){
84. suma += 4.0/(8\*i+(2\*(nodo-1)+3));
85. i++;
86. }
87. suma = (nodo%2 == 0) ? suma : (-suma);
88. salida.writeDouble(suma);
89. salida.close();
90. entrada.close();
91. conexion.close();
92. }
93. }
94. }