

ALEJANDRO DE JESÚS ZEPEDA FLORES

DESARROLLO DE SISTEMAS DISTRIBUIDOS - 4CM3

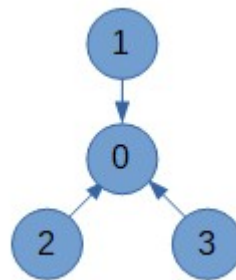
Tarea 1. Cálculo distribuido de PI.

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

Por lo pronto, vamos a probar el programa en una sola computadora utilizando cuatro ventanas de comandos de Windows o cuatro terminales de Linux, en cada ventana se ejecutará una instancia del programa.

Cada nodo (incluso el nodo 0) deberá calcular 10 millones de términos de la serie.

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



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

Desarrollo.

```
adejesuszf@alejandrozf: ~/Escritorio/Distribuidos/Actividad02
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ javac PI.java
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ java PI 0
Valor de pi: 3.141592628592157
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$
```

La imagen muestra una ventana de terminal Linux. La barra superior indica el usuario 'adejesuszf' y la ruta de trabajo '~/Escritorio/Distribuidos/Actividad02'. El historial de comandos muestra: 1. 'javac PI.java' para compilar el programa. 2. 'java PI 0' para ejecutarlo con el argumento 0. La salida del programa es 'Valor de pi: 3.141592628592157'. La línea de comando actual está vacía.

Imagen 1. Compilación y ejecución (Modo servidor).

En la imagen 1, se observa la compilación del programa, así como su ejecución donde se le pasa como argumento el valor de 0 para que esta ejecución se realice como servidor.

```

if(nodo == 0){
    //Algoritmo 2
    ServerSocket servidor = new ServerSocket(50000);
    Worker w[] = new Worker[3];
    int i = 0;
    while(i<3){
        Socket conexion = servidor.accept();
        w[i] = new Worker(conexion);
        w[i].start();
        i = i + 1;
    }

    double suma = 0.0;
    i = 0;
    while(i<10000000){
        suma = 4.0/(8*i+1)+suma;
        i = i + 1;
    }

    synchronized(lock){
        pi = suma + pi;
    }

    i = 0;
    while(i<3){
        w[i].join();
        i = i + 1;
    }

    System.out.println("Valor de pi: "+pi);
}

```

La ejecución como servidor, consiste en crear un ServerSocket utilizando el puerto 5000; además de un arreglo de tipo Worker de dimensión 3.

Posteriormente, dentro de un ciclo creamos un socket e invocamos el método accept(). El resultado de esta operación, se lo pasamos como parámetro a cada instancia del arreglo Worker e iniciamos la ejecución del hilo.

El siguiente paso es hacer el cálculo de 10 millones de términos de la serie y sumarlos a la variable suma.

Después, con el método synchronized, realizamos la suma del valor de la variable suma con el valor pi.

Mostramos el valor de pi.

Imagen 2. Código del servidor.

Después de haber ejecutado nuestro programa en modo servidor, esa terminal está a la espera de las peticiones de los clientes; por lo que, demos abrir 3 terminales más y ejecutar el programa con el parámetro 1-3 para simular otros clientes.

```

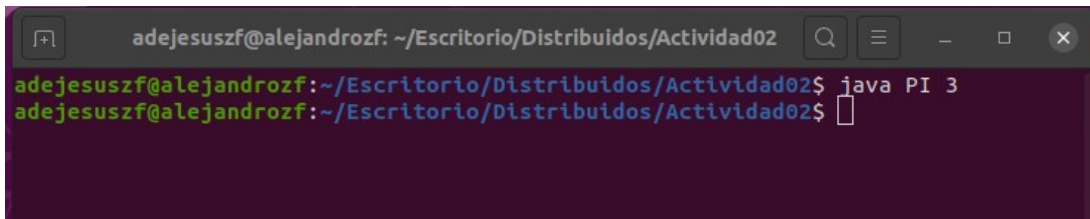
adejesuszf@alejandrozf: ~/Escritorio/Distribuidos/Actividad02
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ java PI 1
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$

```

```

adejesuszf@alejandrozf: ~/Escritorio/Distribuidos/Actividad02
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ java PI 2
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$

```

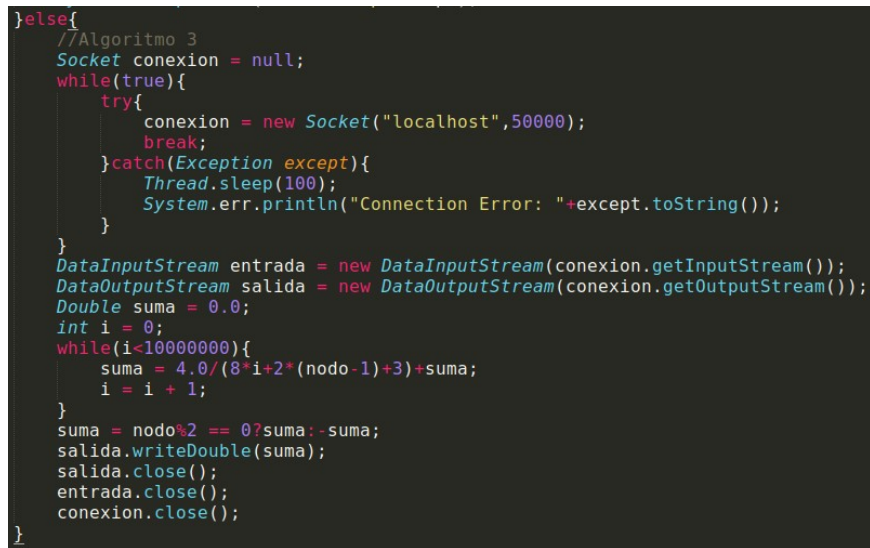
A terminal window with a dark background. The title bar shows the user 'adejesuszf@alejandrozf' and the path '~/Escritorio/Distribuidos/Actividad02'. The prompt is 'adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02\$'. The user has entered 'java PI 3' and the prompt is now 'adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02\$' with a cursor.

```
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ java PI 3
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$
```

Imagen 3. Ejecución de los clientes.

Una vez que el servidor está a la espera de las peticiones de los clientes, ejecutamos los siguientes 3 clientes, lo único que va a cambiar, es el parámetro que le vamos a pasar, será del 1 al 3.

El programa, al detectar esas opciones, entrará en la parte del código que se muestra en la imagen 4. En resumen, se crea una conexión con el servidor, así como los buffers de entrada y salida. Se calculan 10000000 de términos de la serie, se realiza la suma y se envían al servidor el valor de la misma.

A code snippet in a dark-themed editor. It shows a Java code block starting with 'else{' and ending with '}'. The code creates a Socket connection to localhost:50000, sets up DataInputStream and DataOutputStream, calculates a sum of 10,000,000 terms of a series, and sends the result to the server.

```
}else{
    //Algoritmo 3
    Socket conexion = null;
    while(true){
        try{
            conexion = new Socket("localhost",50000);
            break;
        }catch(Exception except){
            Thread.sleep(100);
            System.err.println("Connection Error: "+except.toString());
        }
    }
    DataInputStream entrada = new DataInputStream(conexion.getInputStream());
    DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
    Double suma = 0.0;
    int i = 0;
    while(i<10000000){
        suma = 4.0/(8*i+2*(nodo-1)+3)+suma;
        i = i + 1;
    }
    suma = nodo%2 == 0?suma:-suma;
    salida.writeDouble(suma);
    salida.close();
    entrada.close();
    conexion.close();
}
```

Imagen 4. Código del cliente.

La clase Worker es importante, ya que contiene un constructor dónde se inicializa la conexión. Como la clase hereda de Thread, se puede implementar el método run(), que es el se va a encargar de recibir la suma enviada por los clientes y lo va agregando al valor de pi, dentro del método synchronized. Este código se puede observar en la imagen 5.

```

static class Worker extends Thread{
    Socket conexion;


    Worker(Socket conexion){
        this.conexion = conexion;
    }

    public void run(){
        //Algoritmo 1
        try{
            DataInputStream entrada = new DataInputStream(conexion.getInputStream());
            DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
            double x = entrada.readDouble();
            synchronized(lock){
                pi = x + pi;
            }
            salida.close();
            entrada.close();
            conexion.close();
        }catch(Exception except){
            System.err.println("Thread Error: "+except.toString());
        }
    }
}

```

Imagen 5. Clase Worker.

Para finalizar, en la terminal del servidor, una vez que los 3 clientes hayan completado su ejecución, se mostrará en consola el valor calculado de PI.



```

adejesuszf@alejandrozf: ~/Escritorio/Distribuidos/Actividad02
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ javac PI.java
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ java PI 0
Valor de pi: 3.141592628592157
adejesuszf@alejandrozf:~/Escritorio/Distribuidos/Actividad02$ 

```