Instituto Politécnico Nacional

Escuela Superior de Cómputo

Asignatura:

Desarrollo de sistemas distribuidos

Grupo:4CM3

Tarea 1. Cálculo distribuido de PI

Alumno: Garcia Garcia Rafael

Profesor: Carlos Pineda Guerrero

Capturas de pantalla de la tarea:

Una vez terminado nuestro programa **PI.java** el cuál recibe como entrada un número en donde 0 indica que es el servidor y un número del 1 al 3 indica que es un cliente vamos a pasar a compilar nuestro código en la terminal.

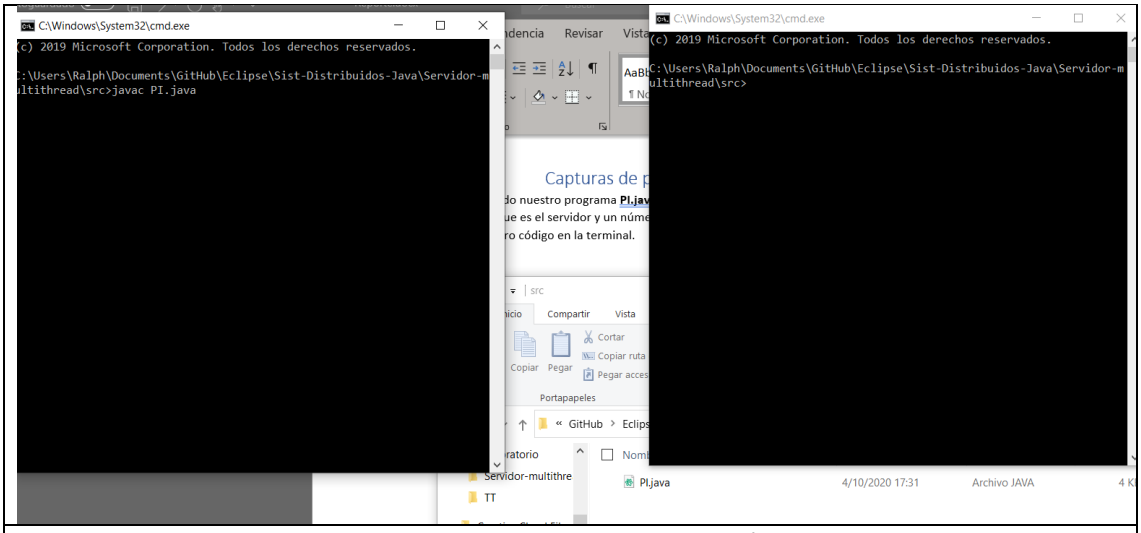

Imagen 1. Antes de la compilación

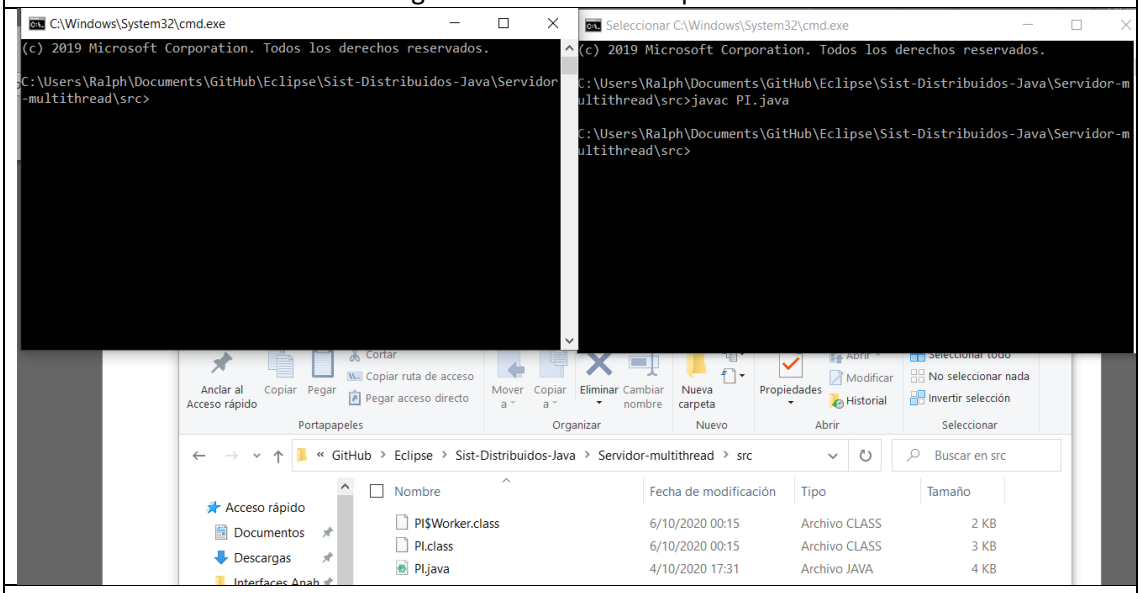

Imagen 2. Después de la compilación

Como vemos en las imágenes 1 y 2 se generán 2 archivos .class uno del Worker y uno del programa principal, ahora procedemos a ejecutar nuestro programa principal indicando primero la numeración del 1 al 3 para inicializar los clientes y esperen a nuestro servidor

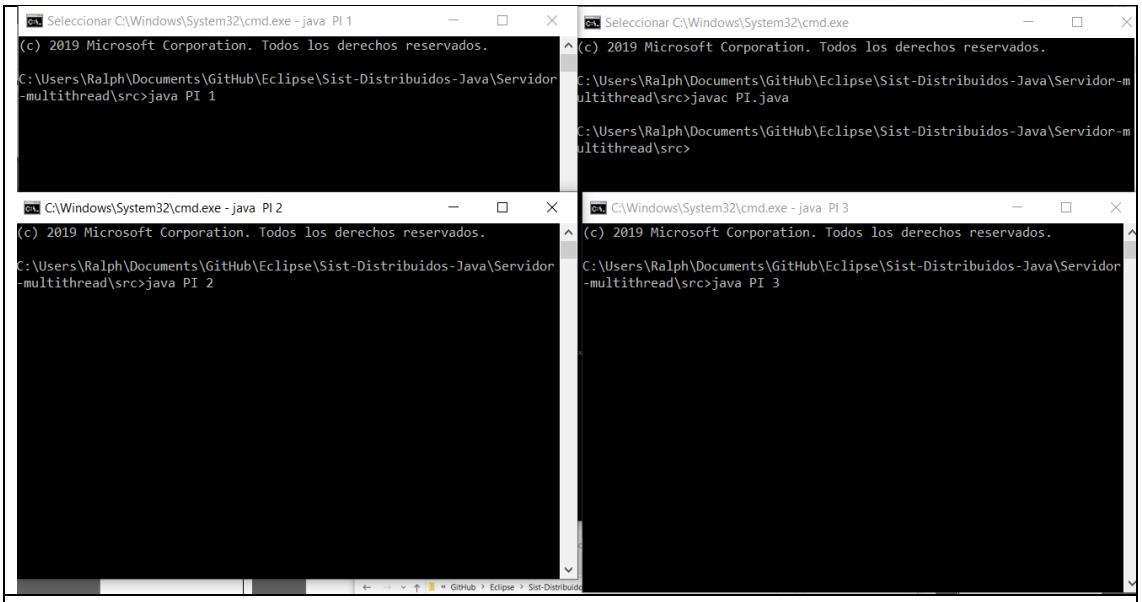

Imagen 3. Ejecución de los clientes

Una vez inicializados nuestros clientes procedemos a ejecutar nuestro servidor para que por medio de hilos estos clientes se ejecuten de manera simultanea y realicen sus operaciones, al final nuestro servidor (terminal arriba a la derecha) recibirá la respuesta y la mostrará en la salida

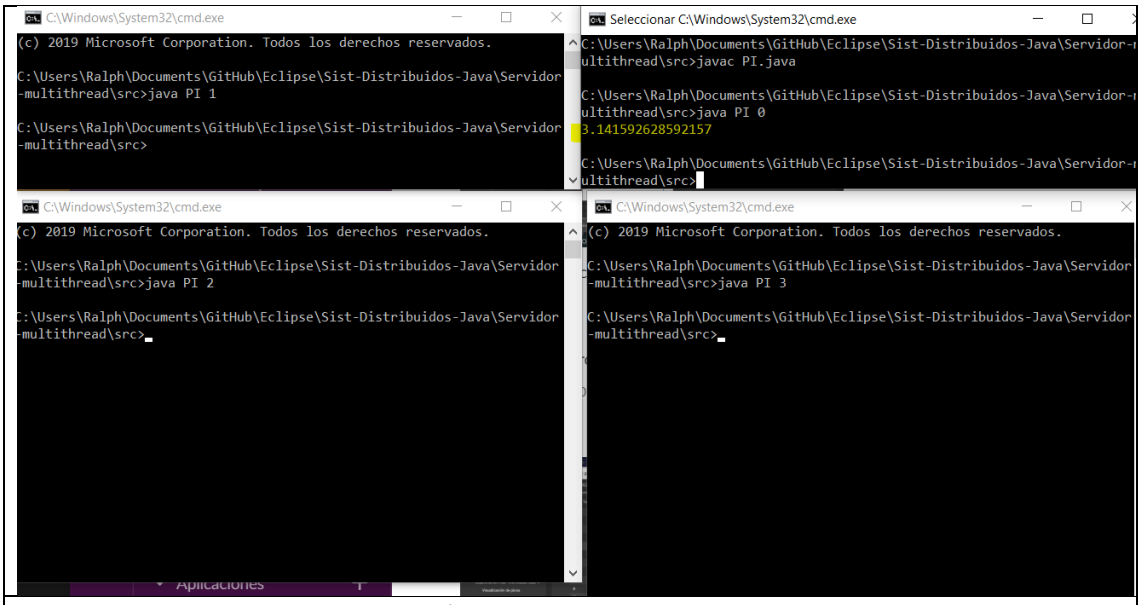

Imagen 4. Ejecución del servidor y muestra del resultado

Como se observa en la imagen 4 el servidor tiene una salida aproximada al valor de PI y nuestros clientes y servidor terminaron su ejecución. Por ende hemos concluido la práctica de manera correcta obteniendo los resultados esperados.

En el código fuente se pueden encontrar comentarios paso a paso de los 3 algoritmos desarrollados donde se explica a detalle que se hace en cada uno de ellos. Se anexan capturas de cada uno de los algoritmos desarrollados

```java
public void run(){
    // Algoritmo 1
    //Creamos try y catch

    try{
        //Creamos los streams de entrada y salida
        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
        DataInputStream entrada = new DataInputStream(conexion.getInputStream());
        //Recibimos la suma del cliente
        double x = entrada.readDouble();
        //bloque sincronizado de la suma
        synchronized(lock){
            pi = x+pi;
        }
        //cerramos nuestros sockets de entrada y salida
        salida.close();
        entrada.close();
        //Cerramos la conexión
        conexion.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}
}
```

Imagen 5. Algoritmo 1

```java
if (nodo == 0){
    // Algoritmo 2
    //Declaramos servidor asignando puerto 50000
    ServerSocket servidor = new ServerSocket(50000);
    //Arreglo de Objeto Worker tamaño 3
    Worker[] w = new Worker[3];
    //Loop que limita el número de conexiones que aceptaremos
    for (int i =0;i!=3;i++){
        //Aceptamos la conexión
        Socket conexion = servidor.accept();
        //Instanciamos nuestra conexion
        w[i] = new Worker(conexion);
        //Inicializamos nuestro hilo
        w[i].start();
    }
    double suma =0;
    //Realizamos la suma para que converja al valor
    for (int i=0;i!=10000000;i++)
        suma =  4.0/(8*i+1)+ suma;
    //asignar valor de la convergencia a PI
    synchronized(lock){
        pi = suma+pi;
    }
    for(int i =0;i!=3;i++)
        //ejecutamos nuestros hilos
        w[i].join();
    System.out.println(pi);
```

Imagen 6. Algoritmo 2

```
        System.out.println(pi);
    }else{
        // Algoritmo 3
        //En esta parte del código inicializamos nuestros clientes
        Socket conexion =null;
        //Realizamos el intento de conexión al servidor
        for(;;){
            try{
                conexion = new Socket("localhost",50000);
                break;
            }catch (Exception e){
                Thread.sleep(100);
            }//fin catch
        }//fin for
        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
        DataInputStream entrada = new DataInputStream(conexion.getInputStream());
        double suma = 0;
        for(int i=0;i!=10000000;i++)
            suma =4.0/(8*i+2*(nodo-1)+3)+suma;
        suma = nodo%2==0?suma:-suma;
        salida.writeDouble(suma);
        entrada.close();
        salida.close();
        conexion.close();
    }
```

Imagen 7. Algoritmo 3

El algoritmo 1 fue la codificación de nuestro Worker, el cual se encargará de gestionar el funcionamiento de nuestro servidor en este caso cuando ya tengamos 3 clientes va a sumar los valores devueltos por ellos para generar la aproximación

El algoritmo 2 es la parte en donde le indicamos a nuestro programa que inicialice un servidor y haga unas operaciones síncronas y asíncronas.

Por último el algoritmo 3 es la parte de nuestro código en donde generamos a nuestros clientes realizamos unas operaciones y se envía el valor por medio de un socket a nuestro servidor