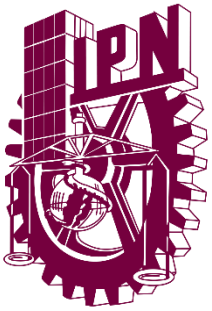


Instituto Politécnico Nacional

Escuela Superior de Cómputo

Desarrollo de sistemas distribuidos

Reporte Tarea 1.
"Cálculo de PI"

Grupo: 4CV13

Integrantes:
Erick Eduardo Ramírez Arellano
Elisa Ramos Gomez
Omar Ramos Herrera

Febrero, 2022.

Desarrollo.

Para esta tarea se calculó la aproximación de PI en 4 servidores. Se creo una clase llamada Practica1 donde iremos describiendo todo el proceso.

Dentro de la función main, que es donde inicia toda la magia, lo que hicimos fue dividir en casos las posibles entradas que recibiría el programa donde tenemos:

1. Cuando ingresan valor de 0 se ejecutará la porción de código designado al cliente.

2. Cuando se ingresa un valor de 1 a 4, se ejecutará la porción de código designado al servidor.

3. Cualquier otra cosa ingresada mandará una excepción de RunTime, con el mensaje "valor de nodo incorrecto".

Ahora iremos a detallar lo que sucede en el Cliente, que es el punto número uno.

1. Cliente

Dentro de esta clase está inmerso los reintentos de conexión esperando a que los servidores sean "levantados" por llamarlo de alguna forma y una vez que sucede esto pasamos a utilizar un socket para hacer la conexión con el respectivo servidor; quedando en espera del resultado calculado por el servidor.

Una vez que el servidor termina el cálculo, el Cliente lo lee y hacemos uso de synchronized para justamente controlar con sincronización la escritura de la variable PI.

El código se ve de esta manera:

```
static class Cliente extends Thread{

        int numServidor;
        static double PI = 0.0;
        static Object obj = new Object();


        Cliente(int numServidor){
            this.numServidor = numServidor;
        }

        public void run(){
            Socket conexion = null;
            for(;;){
                try{

                  conexion = new Socket("localhost", 50000+
                  numServidor);
```

```
                    System.out.println("Cliente conectato con el servidor
                    #: " + numServidor);

                    DataInputStream entrada = new
                    DataInputStream(conexion.getInputStream());
                      // System.out.println(PI + " ANTES DE SUMA ");

                    synchronized(obj){
                       PI += entrada.readDouble();
                          // System.out.println(PI + " en DE SUMA ");
                    }

                      // System.out.println(PI + " despues DE SUMA ");

                      Thread.sleep(1000);
                      conexion.close();
                      break;

                  }catch(Exception e){

                      try {
                          Thread.sleep(100);
                      } catch (InterruptedException e2) {
                          // e2.printStackTrace();
                          System.out.println(e2.getMessage());
                      }
                  }
              }

          }
      }
```

Dentro de la función main utilizamos barreras para asegurar el correcto funcionamiento de los hilos y el cálculo de PI.

2. Servidor

Es una clase estática que hereda de la clase Thread (hilo) donde lo primero que realizamos es el cálculo de PI donde creamos una función llamada calculaTerminos donde tenemos acceso al número de servidor (en nuestro código numNodo) y se calcula usando la fórmula descrita en el desarrollo de la tarea, posteriormente utilizamos la instancia del flujo de datos para escribir y enviar el resultado de la sumatoria, donde escrito en código se ve de la siguiente manera.

```
static class Servidor extends Thread{

        Socket conexion;
        int numNodo;
```

```
    Servidor(Socket conexion, int numNodo){
        this.conexion = conexion;
        this.numNodo = numNodo;
    }

    public double calculaTerminos(){

        double sumaTotal = 0.0;

        for(int i = 0; i < 1000000; i++){
            sumaTotal += (4.0/(8*i+2*(numNodo-2)+3));
        }

        if(numNodo%2 == 0){
            return -sumaTotal;
        }

        return sumaTotal;
    }


    //Metodo run que se ejecuta al iniciar un hilo
    public void run(){
        try {

            System.out.println("Cliente conectado, calculando Serie
            Gregory-Leibniz...");
            DataOutputStream salida = new
            DataOutputStream(conexion.getOutputStream());

            // System.out.println("El valor es: ");
            double valor = calculaTerminos();
            // System.out.println(valor);

            salida.writeDouble(valor);
            Thread.sleep(1000);


        } catch (Exception e) {
            //System.out.println("Se murio");
            System.out.println(e.getMessage());
        }

    }

}
```

Una vez finalizado el cálculo se le envía al Cliente y cerramos la conexión.

3. Main

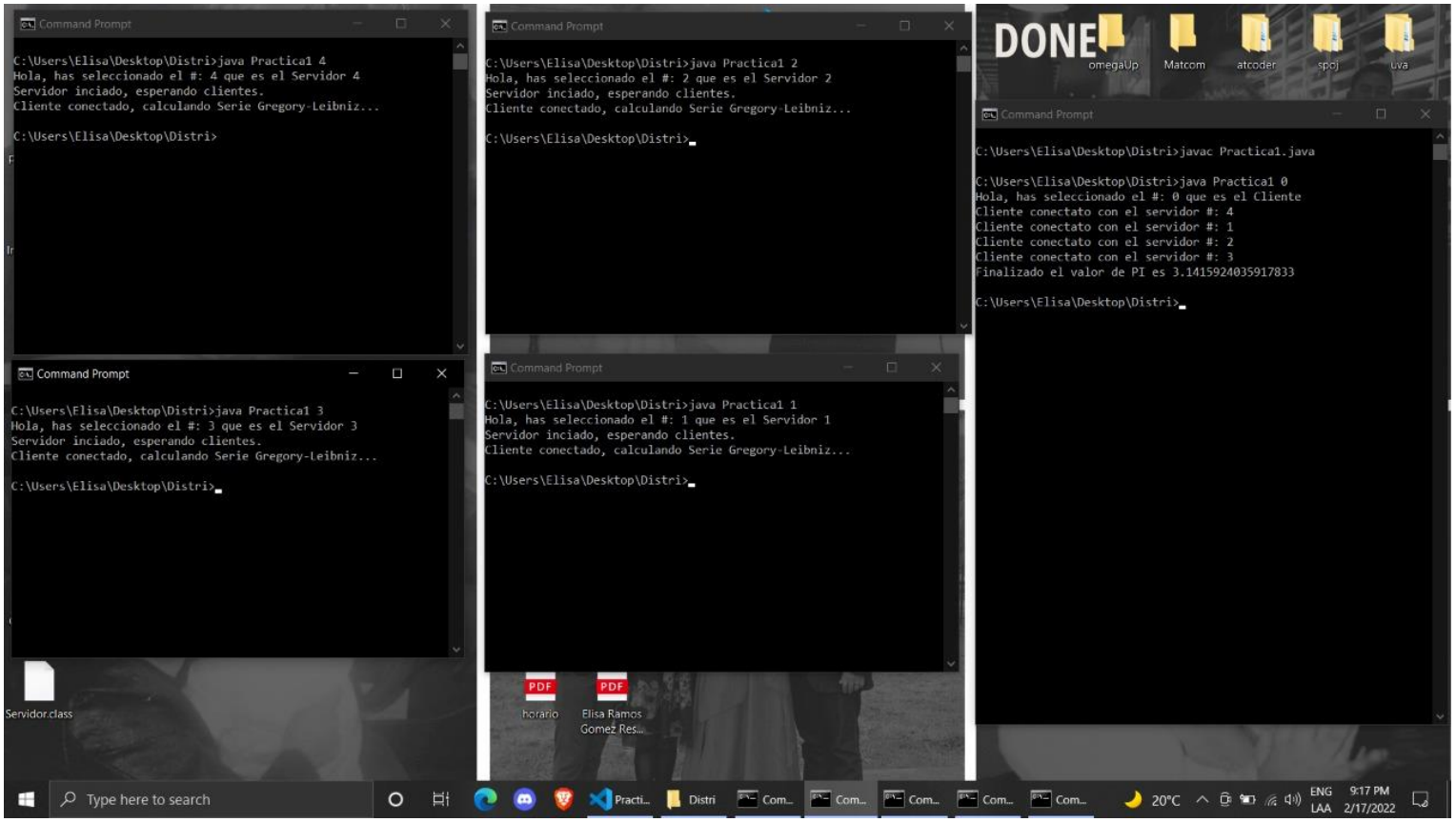
Cuando el usuario ingresa el número 0, lo primero que hacemos es enviarle un mensaje de que ha seleccionado al Cliente y posteriormente crean 4 instancias de cliente donde se le envía el número de servidor. Cabe mencionar que Cliente es una clase estática que hereda de la clase Thread (hilo) la cual nos ayudará a este cómputo paralelo del cálculo de PI. Implementamos una barrea con los joins, así haremos que el hilo principal espera a que terminen los demás hilos. Después accedes a la variable PI para mostrar el valor calculado.

Cuando el usuario ingresa el servidor, lo primero que hacemos es enviarle un mensaje de que ha seleccionado al Servidor y posteriormente creamos una instancia de ServerSocket para abrir un puerto de conexión, pasando a crear otra nueva instancia de tipo Servidor enviándole como parámetro el número del servidor.

```
public static void main(String[] args) {


        System.out.print("Hola, has seleccionado el #: ");
        System.out.print(args[0]);
        System.out.print(" que es ");

        int numNodo = Integer.parseInt(args[0]);

        if(numNodo == 0){
            System.out.println("el Cliente");

            try{

                Cliente c1 = new Cliente(1);
                Cliente c2 = new Cliente(2);
                Cliente c3 = new Cliente(3);
                Cliente c4 = new Cliente(4);

                c1.start();
                c2.start();
                c3.start();
                c4.start();

                c1.join();
                c2.join();
                c3.join();
                c4.join();

                System.out.println("Finalizado el valor de PI es " +
                Cliente.PI);

            }catch(Exception e){
                System.out.println(e.getMessage());
            }


        }else if (numNodo>= 1 && numNodo <= 4){
            System.out.println("el Servidor " + numNodo);
```

```
                try{

                    ServerSocket servidor = new
                    ServerSocket(50000+numNodo);
                    System.out.println("Servidor inciado, esperando
                    clientes.");

                    Socket conecta = servidor.accept();

                    Servidor calcula = new Servidor(conecta, numNodo);
                    calcula.start();

                    // conecta.close();
                    servidor.close();

                }catch(Exception e){
                    System.out.println(e.getMessage());
                }

            }else{
                throw new RuntimeException("un valor de nodo incorrecto");
            }


        }

    }
```

Como paso final adjuntamos la captura de pantalla de los 4 servidores y el cliente funcionando de manera correcta; también es importante resaltar que esta serie tarda en converger por lo que el resultado final es idéntico hasta el dígito 6 después del punto.



Conclusiones.

Erick Eduardo Ramírez Arellano: Gracias a la elaboración de esta práctica pude identificar la función de un nodo servidor y un nodo cliente, de igual forma entendí más a fondo la comunicación que se realiza entre ellos, así como, entender que se debe de hacer para establecer un nodo servidor, ya que, al leer el desarrollo y los ejemplos del profesor, entendí que cada nodo servidor debe de ser declarado en diferentes puertos siempre y cuando los nodos servidores sean establecidos en una misma computadora, de igual forma comprendí un poco más la diferencia de la función de un nodo cliente y un nodo servidor, así como la forma de comunicarse de cada uno.

Omar Ramos Herrera: La elaboración del programa distribuido que calcule una aproximación del número pi nos ayudó como repaso al tema antes visto, Sockets e hilos, aprendimos a utilizar las barreras de join y así esperar que terminaran los hilos. Vimos como ejecutar varios clientes con hilos, ocupar el arg del método main el cual nos permitió elegir que ejecutar, el cliente y servidor conectados con un socket orientado a conexión.

Elisa Ramos Gomez: Me gustó mucho realizar esta tarea, ya que fue mucha prueba y error, forzándome a comprender los mensajes recibidos de la máquina virtual, de igual forma de cuidar las excepciones que arrojan los hilos, el runtime, etc; para poder realizarla tuve que regresarme a realizar las actividades propuestas en la plataforma para poder integrar de manera correcta los temas vistos en clase y poder hacer el trabajo en equipo dejando ideas de solución concretas y específicas, igual tuve que desempolvar los conocimientos en Java ya que tenía un rato que no hacía uso del lenguaje pero no me imagino lo tedioso de hacer esta misma práctica en C entonces estoy agradecida con la decisión de utilizar lenguaje Java en el curso, definitivamente la tarea me ayudó a darme una idea un poco más tangible de cómo es que se lleva a cabo el cómputo paralelo, que me tiene muy sorprendida.