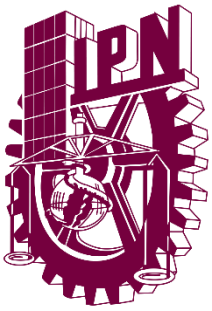
Instituto Politécnico Nacional

Escuela Superior de Cómputo



Desarrollo de sistemas distribuidos

Reporte Tarea 2.

"Implementación de un token-ring"

Grupo: 4CV13

Alumnos:

Erick Eduardo Ramírez Arellano

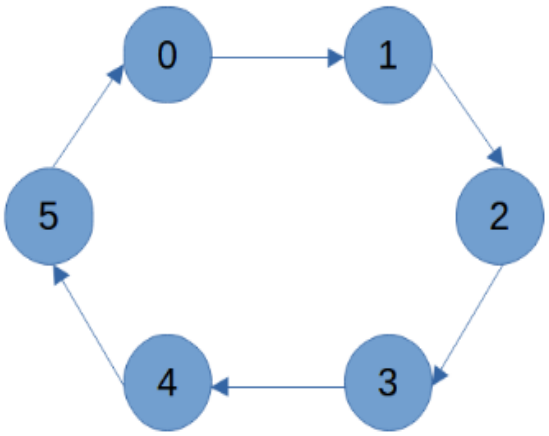
Elisa Ramos Gómez

Omar Ramos Herrera

Marzo, 2022.

Desarrollo

Se desarrollo una topología de anillo donde la idea es que cada nodo funja como cliente y servidor. Para poder comprender esto un poco mejor partimos de una abstracción donde cada círculo es un servidor y las flechas significan ser cliente.



Una vez partiendo de eso, decidimos pelearnos una poco con el manejo de hilos y crear seis hilos para poder hacer la comunicación, donde la función por default es el run metimos la lógica del Servidor y creamos una función void dentro de nuestra clase Proceso para escribir la lógica del Cliente, al hacer esto es de vital importancia tener el control del número de nodo que estamos manejando.

1. Cliente

Como ya mencionamos todo esta encapsulado dentro de la clase Proceso donde tenemos una función especial de tipo void donde metemos la lógica del cliente, decidimos hacerlo con los hilos para adquirir práctica en esto de utilización de hilos, entonces regresando a esto lo que hicimos fue primero implementar un cliente con reintentos de conexión como ya habíamos visto y utilizado en las actividades de las clases, luego utilizamos sockets especiales para enviar – recibir mensajes de manera segura utilizando sockets seguros abriendo conexión y modulando el número de nodo que recibíamos para abrir solo los seis puertos diferentes que necesitamos para realizar esta tarea, que de igual forma vimos en la asesoría.

Una vez que abrimos conexión solo enviamos el token para que sea alterado su valor por otro servidor, en código se ve de la siguiente manera:

```
public void clienteSSL(short token){

  Socket conexion = null;
  for(;;){
      try{
            SSLSocketFactory cliente = (SSLSocketFactory)
            SSLSocketFactory.getDefault();

            conexion = cliente.createSocket("localhost", 50000  +
            ((numNodo + 1) %6));

            DataOutputStream salida = new
            DataOutputStream(conexion.getOutputStream());
```

```
            System.out.println("Cliente conectato con el servidor #: " +
            ((numNodo + 1) %6));

            salida.writeShort(token);

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

      }catch(Exception e){

            try{
                  Thread.sleep(1000);
            }catch(InterruptedException e2){
                  System.out.println(e2.getMessage());
            }
       }

   }

}
```

2. Servidor

Para esta parte lo bueno es que se copia y pega el servidor de la tarea pasada lo que cambiamos fue a utilizar los sockets seguros para abrir el puerto correspondiente, donde cada uno de los servidores estará ejecutándose, estando de alguna forma “preparado” para aceptar conexiones de los clientes. Una vez aceptada la conexión recibimos el token, lo incrementamos e imprimimos un mensaje que diga su valor y enviamos al cliente.

Como ya mencionamos el servidor será el run del hilo porque quisimos practicar el manejo de estos, de igual manera aquí es donde agregamos la condición que cubre el requerimiento de que si el token es mayor o igual a un valor de 500 y nos encontramos en nodo 0, entonces finalizamos el programa, entonces el run del hilo quedaría de la siguiente forma:

```
public void run(){  //Servidor
      try{

            SSLServerSocketFactory socket_factory =
            (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();

            ServerSocket servidor =
            socket_factory.createServerSocket(50000 + numNodo);

            for(;;){

                  Socket conectado = servidor.accept();

                  DataInputStream entrada = new
                  DataInputStream(conectado.getInputStream());
```

```
                System.out.print("El valor es: ");
                short token = entrada.readShort();
                token++;
                 System.out.println(token);

                 if(numNodo == 0 && token >= 500){
                     System.out.println("Hemos finalizado");
                     break;
                 }

                 clienteSSL(token);

                 conectado.close();
             }

             servidor.close();

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

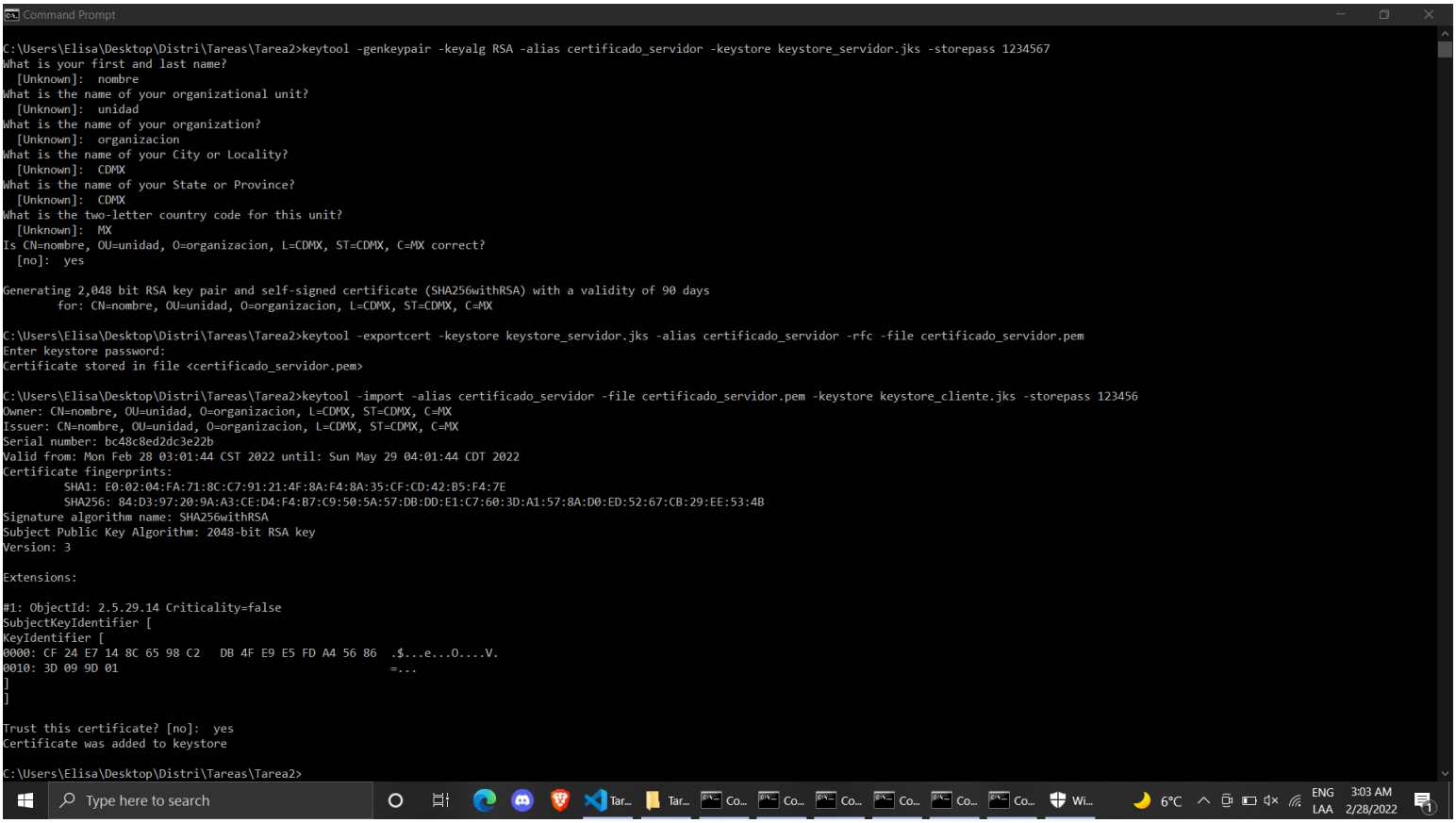
3. Main

Esta parte pues estuvo sencilla ya que se quedó como la plantilla de la práctica anterior donde si es el nodo 0 la cuestión interesante es que iniciamos primero el cliente y después iniciamos el hilo, para todos los demás iniciamos igual su propio hilo y sucede todo lo que hemos ya descrito y que en la parte inferior agregamos las capturas de pantalla de este.

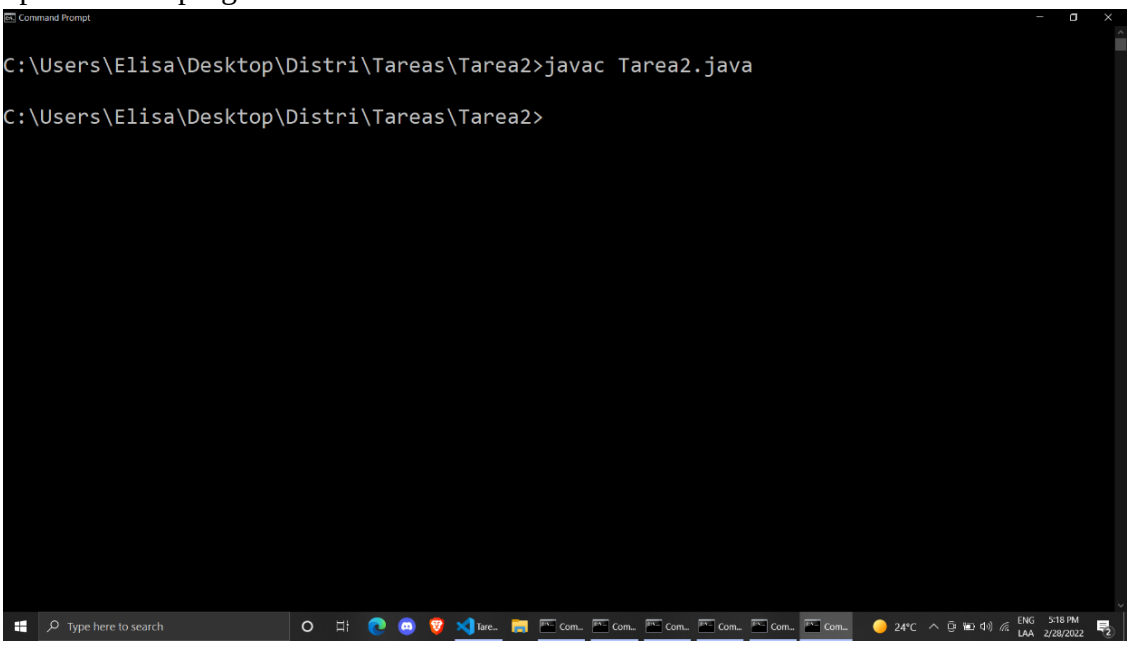
```
public static void main(String[] args) {

        System.out.print("Hola, has seleccionado el #: ");
        System.out.print(args[0]);
        int numNodo = Integer.parseInt(args[0]);

        if(numNodo == 0){

            Proceso inicia = new Proceso(numNodo);
            short valorToken = 0;
            inicia.clienteSSL(valorToken);
            inicia.start();

        }else if (numNodo>= 1 && numNodo <= 5){
            Proceso inicia = new Proceso(numNodo);
            inicia.start();
        }else{
            throw new RuntimeException("un valor de nodo incorrecto");
        }
    }
```

Adjuntamos las capturas de pantalla, donde creamos los certificados están a las 3:00am debido a que primero lo realizamos de manera recomendada en la asesoría sin sockets seguros y eso nos quedo en la madrugada y ese mismo día creamos los certificados y ya al día siguiente una vez descansados pues ya lo realizamos con los sockets seguros; igual en el código fuente dejamos la parte de los sockets normales comentados para tener como una guía de donde habría que cambiarles a los sockets seguros.

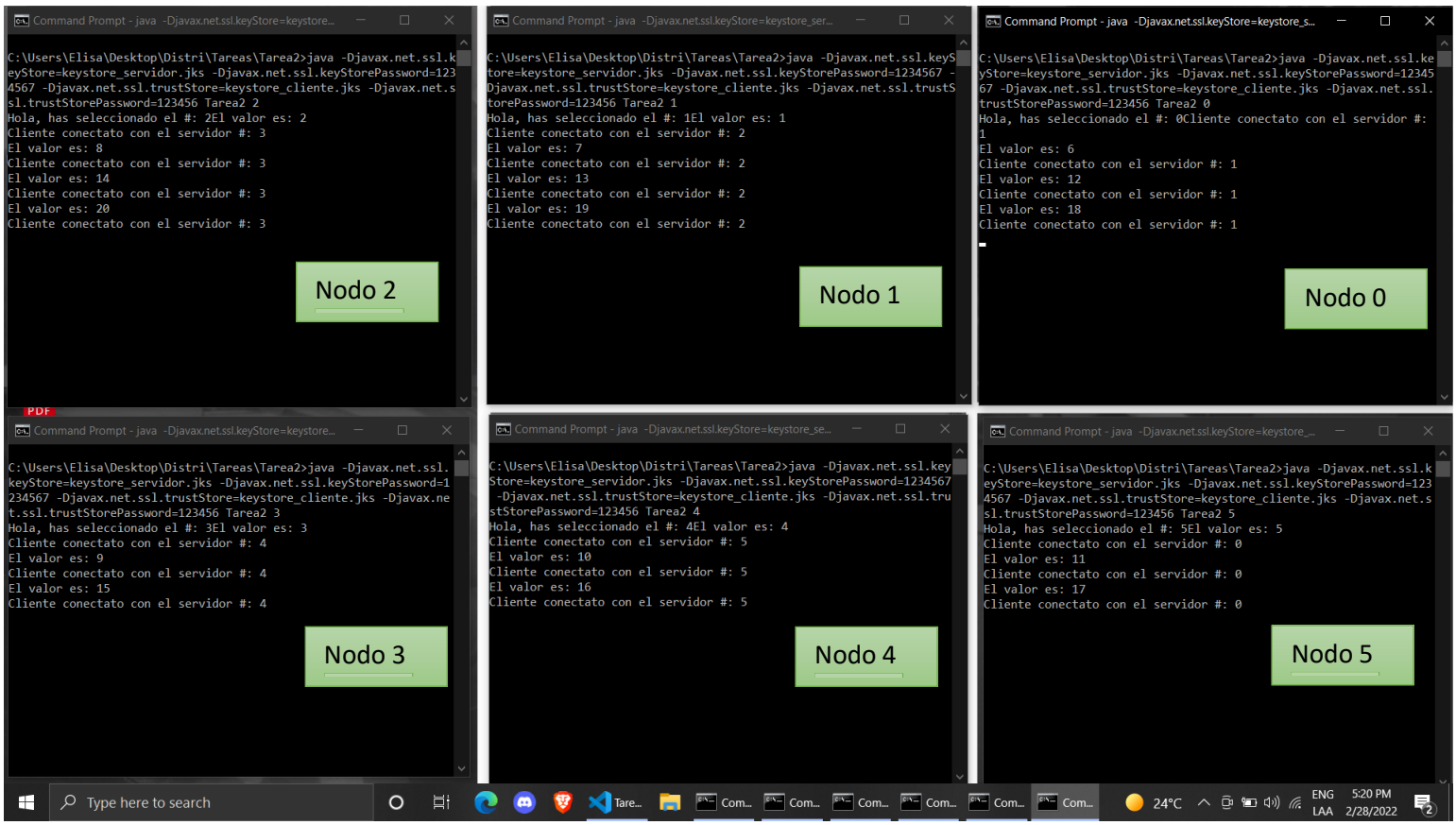
Creación del repositorio (keyStore) para el servidor y el cliente.



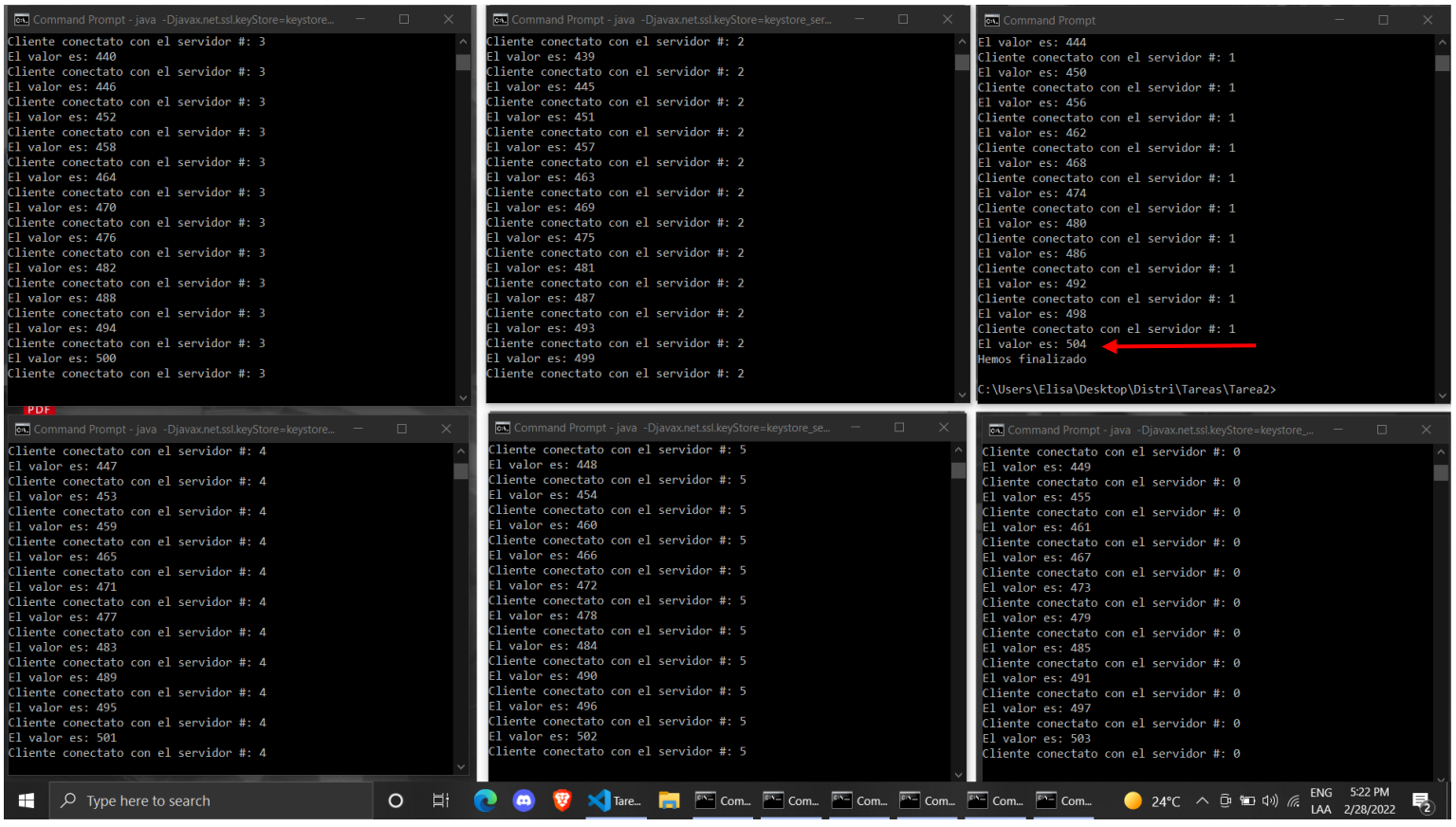
Compilación del programa.



Ejecución de los clientes y servidores, se puede apreciar como el token va incrementando.



Finalización del programa cundo el toque vale más de 500.



Erick Eduardo Ramírez Arellano: Con base en la elaboración de esta tarea, comprendí a fondo la función de los servidores threads así como las ventajas que este implica, ya que, al momento de ser estos implementados, no es necesario hacer uso de un método bloqueante permitiendo así la comunicación entre diferentes nodos, de igual forma, aprendí el cómo se debe de implementar los re-intentos entre un cliente y servidor, forzando así al nodo cliente una sucesión de intentos de conexión al nodo servidor estableciendo yo mis el tiempo de demora, también me ayudo a comprender más a fondo la implementación de los sockets seguros mediante el uso de certificados auto firmados utilizando el programa keytool incluido en JDK.

Elisa Ramos Gomez: Al inicio me sentía un poco espantada del dibujo, pero después razonándolo opte por hacer la lógica en papel, con dibujos entonces ahí fue donde se iluminó una idea de como poder realizarlo, que fueron los hilos y encapsular en una clase el proceso de ambas partes cumpliendo con la dualidad de que cada nodo sea cliente y servidor. Una vez hecho eso, ya tenía realizado la actividad de sockets seguros aunque tuve que darle otro repaso porque hay algunas partes que no me quedaban muy esclarecidas, de igual manera tengo que admitir que StackOverflow una vez más me salvo la vida ya que me daba errores de ejecución con los sockets seguros, intente moverle al setProperty pero no funcionó, aun no se la razón exacta del porque pero fue cuando al buscar el error y vi que lo quitaban y funcionaba entonces hice lo mismo que recomendaban; por otro lado fue más relajado realizar esta tarea porque bueno ya con la anterior teníamos como una plantilla sólida de que hacer, a pesar de que esta vez fue más prueba y error, me emocioné cuando todo empezó a funcionar como esperábamos, en definitiva valió la pena el desvelo.

Omar Ramos Herrera: La realización de la tare ayudo a comprender más lo que es la zona critica, exclusión mutua además de las herramientas que no prevé java para la realización de conexiones totalmente confiables, pudimos observar como el token iba incrementando con forme iba pasando de nodo en nodo. Gracias a esta actividad se comprende porque la programación *multithreading* trae muchos beneficios siendo óptimo el tiempo de ejecución de un programa si lo distribuimos, muchos de estos temas vistos en sistemas operativos toco retomarlo para una mejor compresión de el objetivo de la actividad y además podemos ver que Java es un buen lenguaje, robusto y con grandes herramientas para hacer sistemas más seguros.