Instituto Politécnico Nacional

Escuela Superior de Cómputo

TOKEN RING

TAREA 2

Materia:

Desarrollo De Sistemas Distribuidos

Grupo:

4CV14

Profesor:

Pineda Guerrero Carlos

Alumno:

Castro Cruces Jorge Eduardo

Boleta:

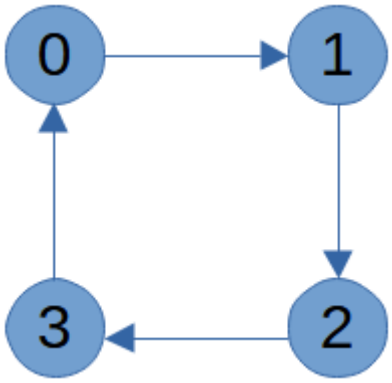
2015080213

Fecha:

Viernes, 9 de septiembre de 2021

1. Desarrollo del programa

Desarrollar un programa en Java, el cual implementará un token que se enviará de un nodo a otro nodo mediante **sockets seguros**, en una topología lógica de anillo. El anillo constará de cuatro nodos:



El token será un número entero de 64 bits.

1. Al principio el nodo 0 enviará el token al nodo 1.
2. El nodo 1 recibirá el token y lo enviará al nodo 2.
3. El nodo 2 recibirá el token y lo enviará al nodo 3.
4. El nodo 3 recibirá el token y lo enviará al nodo 0.
5. El nodo 0 recibirá el token y lo enviará al nodo 1.
6. Ir al paso 2.

Cuando un nodo reciba el token lo incrementará en 1 y desplegará el valor del token.

Cuando la cuenta en el nodo 0 llegue a 1000, el nodo 0 deberá terminar su ejecución.

Vamos a probar el programa en una sola computadora utilizando cuatro ventanas de comandos de Windows o cuatro terminales de Linux o MacOS, cada ventana ejecutará una instancia del programa.

Se deberá completar el siguiente programa (notar que este programa es un cliente y un servidor):

```
class Token

{

  static DataInputStream entrada;
```

```
  static DataOutputStream salida;
  static boolean inicio = true;
  static String ip;
  static int nodo;

  static long token;


  static class Worker extends Thread

  {

    public void run()

    {

      Algoritmo 1
    }

  }



  public static void main(String[] args) throws Exception

  {

    if (args.length != 2)

    {

      System.err.println("Se debe pasar como parametros el numero del nodo y la IP del siguiente nodo en el
anillo");

      System.exit(1);

    }



    nodo = Integer.valueOf(args[0]);
    ip = args[1];



    Algoritmo 2

  }

}
```

Se deberá implementar los siguientes algoritmos:

Algoritmo 1

1. En un bloque try:

1.1 Declarar la variable **servidor** de tipo ServerSocket,

1.2. Asignar a la variable **servidor** el objeto: new ServerSocket(20000+nodo)

1.3. Declarar la variable **conexion** de tipo Socket.

1.4. Asignar a la variable **conexion** el objeto servidor.accept().

1.5. Asignar a la variable **entrada** el objeto new DataInputStream(conexion.getInputStream()).

2. En el bloque catch desplegar el mensaje de la excepción.

Algoritmo 2

1. Declarar la variable **w** de tipo Worker.

2. Asignar a la variable **w** el objeto new Worker().

3. Invocar el método w.start().

4. Declarar la variable **conexion** de tipo Socket. Asignar null a la variable **conexion**.

5. En un ciclo:

    - 5.1 En un bloque try:
        - 5.1.1 Asignar a la variable **conexion** el objeto Socket(ip,20000+(nodo+1)%4).
        - 5.1.2 Ejecutar break para salir del ciclo.
    - 5.2 En el bloque catch:
        - 5.2.1 Invocar el método Thread.sleep(500).

6. Asignar a la variable **salida** el objeto new DataOutputStream(conexion.getOutputStream()).

7. Invocar el método w.join().

8. En un ciclo:

    - 8.1 Si la variable **nodo** es cero:
        - 8.1.1 Si la variable **inicio** es true:

            - 8.1.1.1 Asignar false a la variable **inicio**.
            - 8.1.1.2 Asignar 1 a la variable **token**.
        - 8.1.2 De otra manera:
            - 8.1.2.1 Asignar a la variable **token** el resultado del método entrada.readLong().
            - 8.1.2.2 Incrementar la variable **token**.
            - 8.1.2.3 Desplegar las variables **nodo** y **token**.
    - 8.2 De otra manera:
        - 8.2.1 Asignar a la variable **token** el resultado del método entrada.readLong().
        - 8.2.2 Incrementar la variable **token**.
        - 8.2.3 Desplegar las variables **nodo** y **token**.
    - 8.3 Si la variable **nodo** es cero y la variable **token** es mayor o igual a 1000:
        - 8.3.1 Salir del ciclo.
    - 8.4 Invocar el método salida.writeLong(**token**).

Notar que cada nodo abre un puerto diferente debido a que los cuatro nodos ejecutan en la misma computadora.

Para ejecutar el programa en cada nodo se debe pasar como parámetros el número del nodo y la IP del siguiente nodo en el anillo (en este caso localhost).

Se deberá subir a la plataforma un archivo PDF que incluya: portada, captura de pantalla de la compilación del programa, captura de pantalla de la ejecución del programa en cada ventana (capturar la pantalla cuando termina el programa), captura de pantalla correspondiente a la creación de los repositorios (keystore) del cliente y el servidor, y conclusiones.

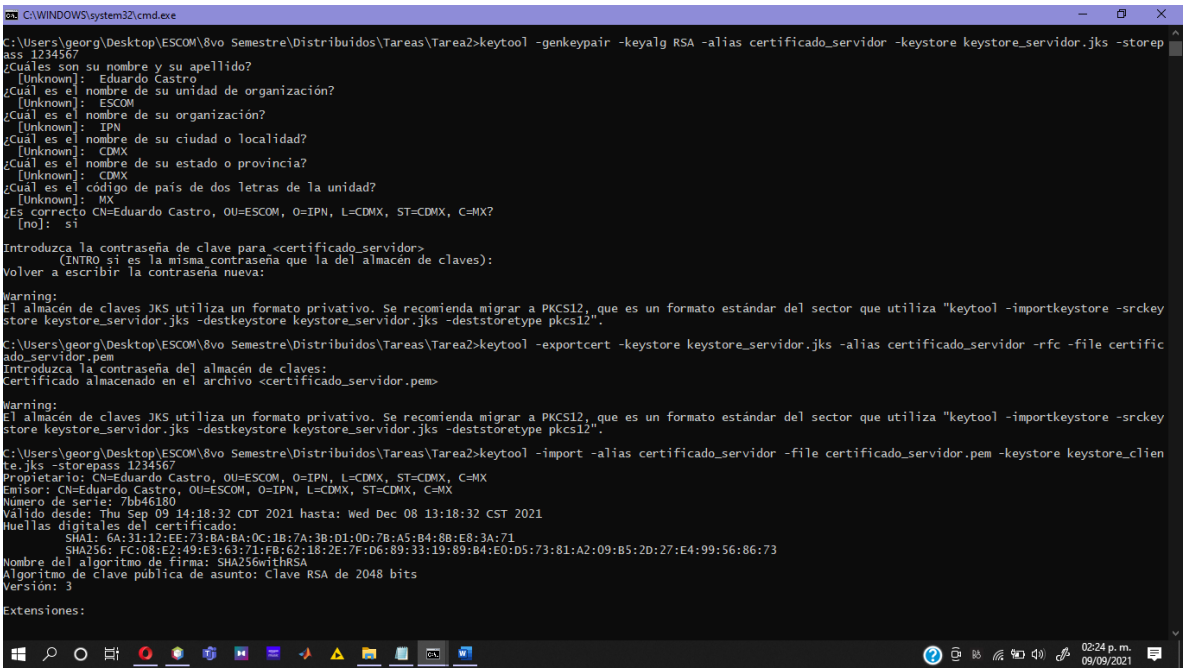
También se deberá subir a la plataforma el código fuente del programa desarrollado (como texto no como imagen).

De acuerdo a los "Lineamientos del curso" las capturas de pantalla deberán ser completas y legibles. El único formato de compresión admitido es ZIP.

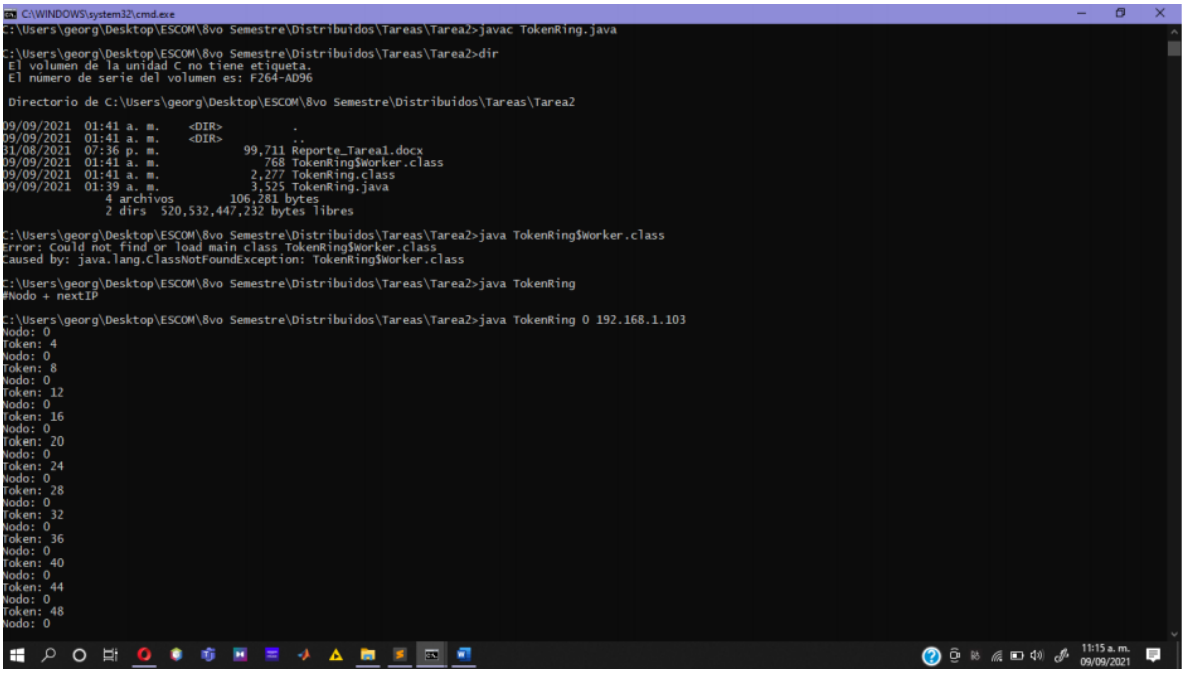
Valor de la tarea: 30% (2.1 puntos de la primera evaluación parcial)

2. Pruebas de escritorio

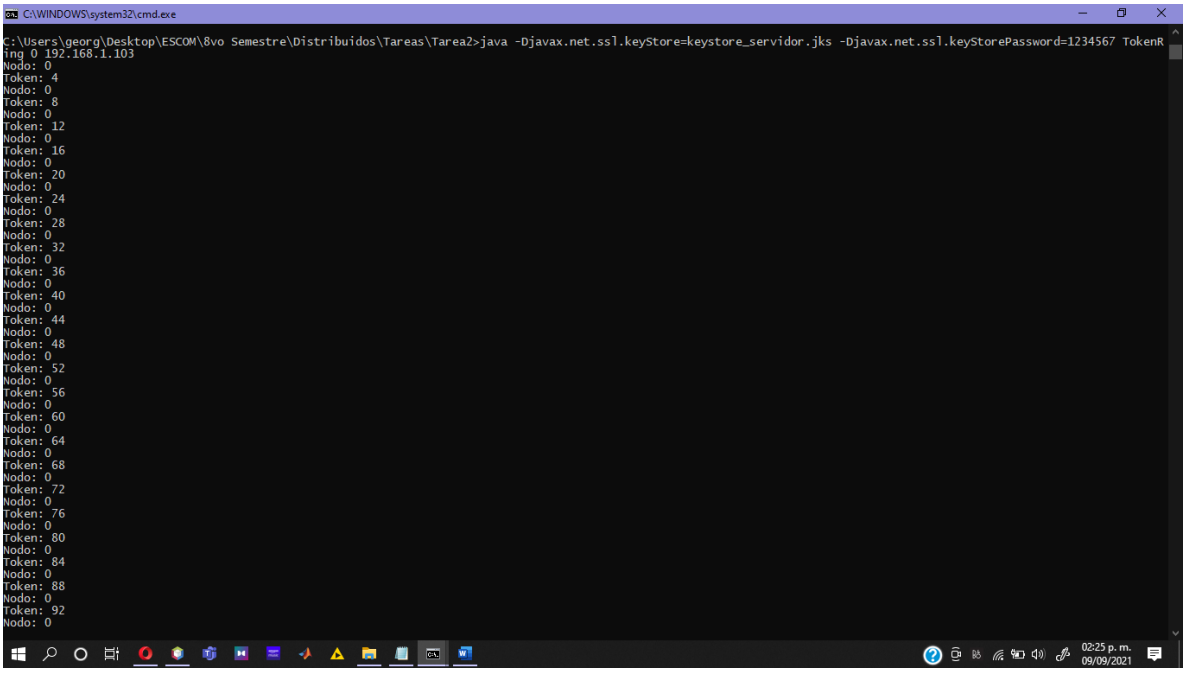
Primero se crearon los certificados correspondientes, suando los comandos vistos en clase:

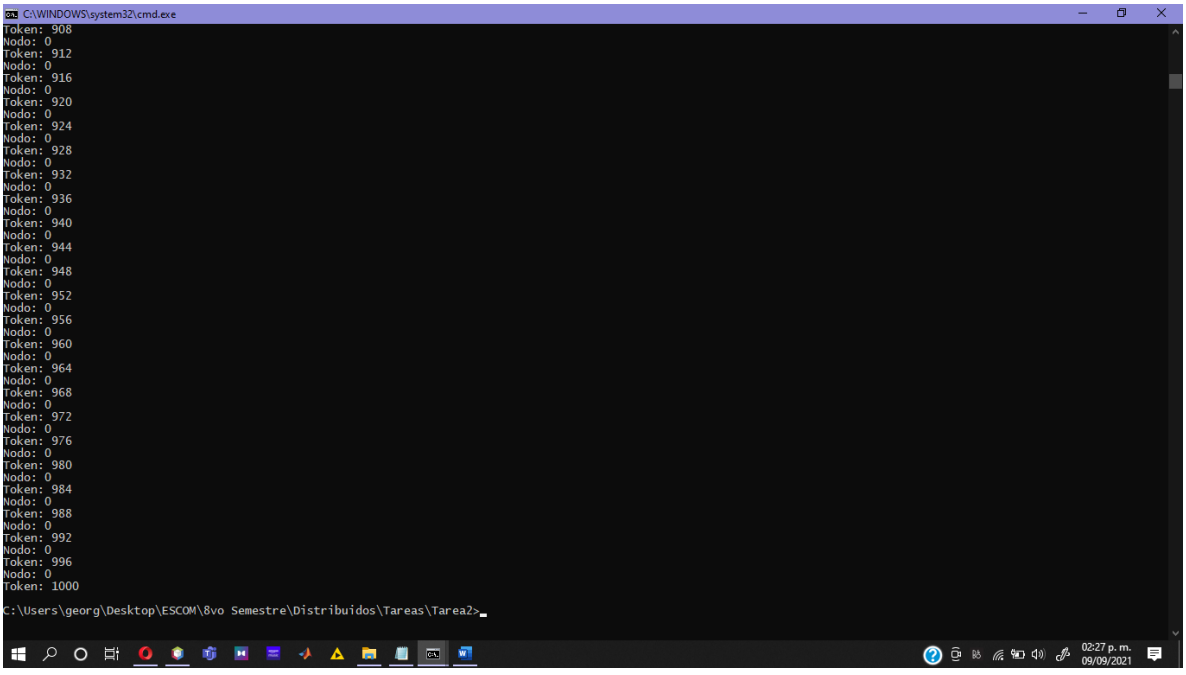


Compilación del programa en la terminal 0:

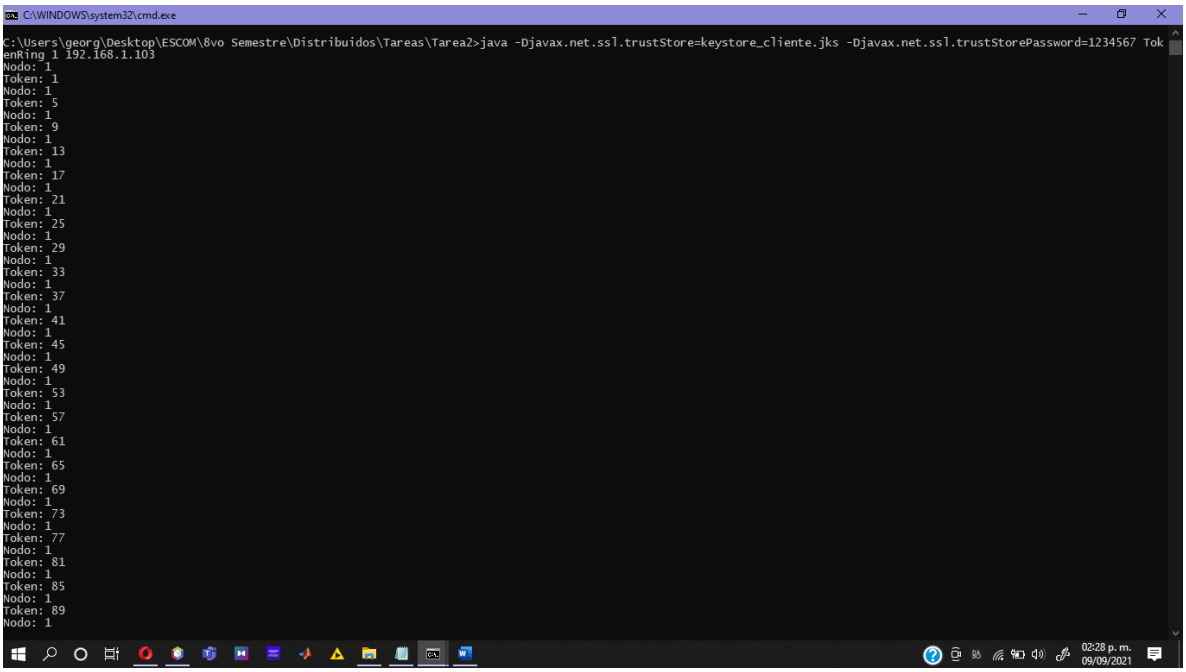


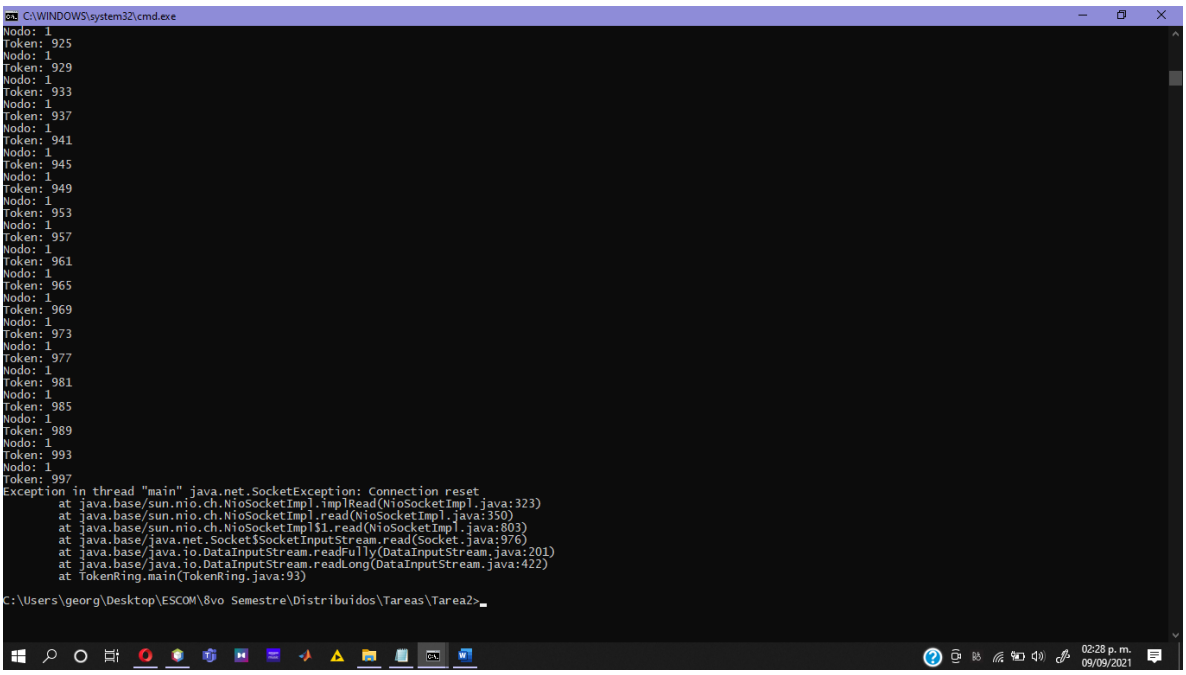
Ejecución del programa en la terminal 0 (servidor):



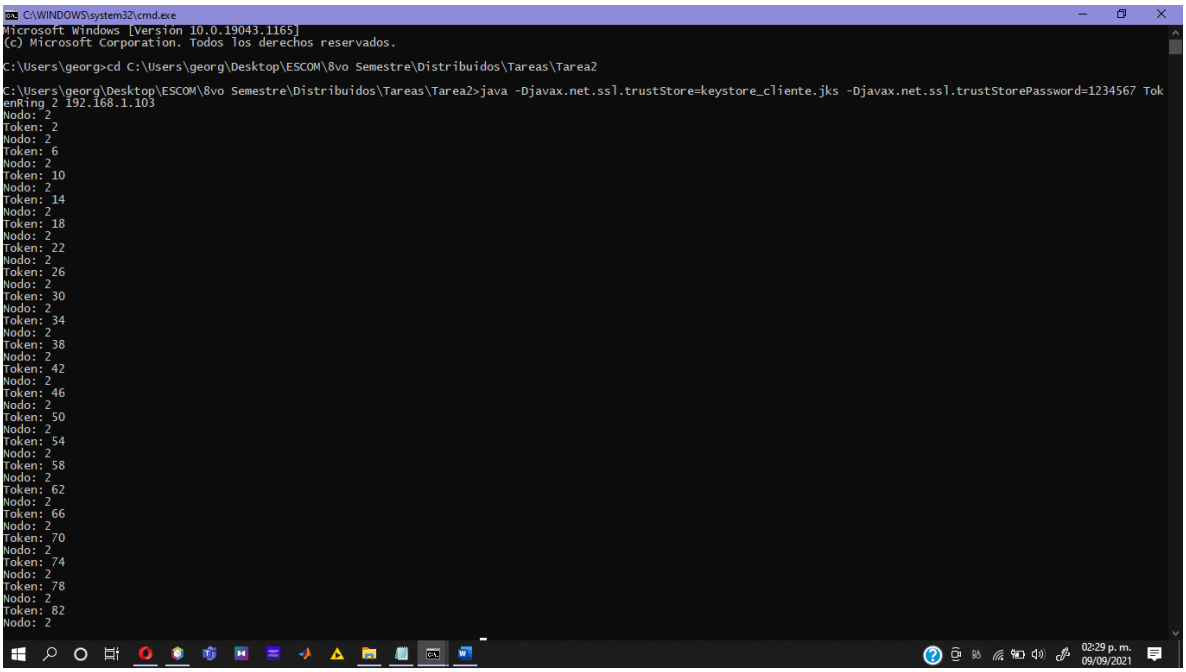


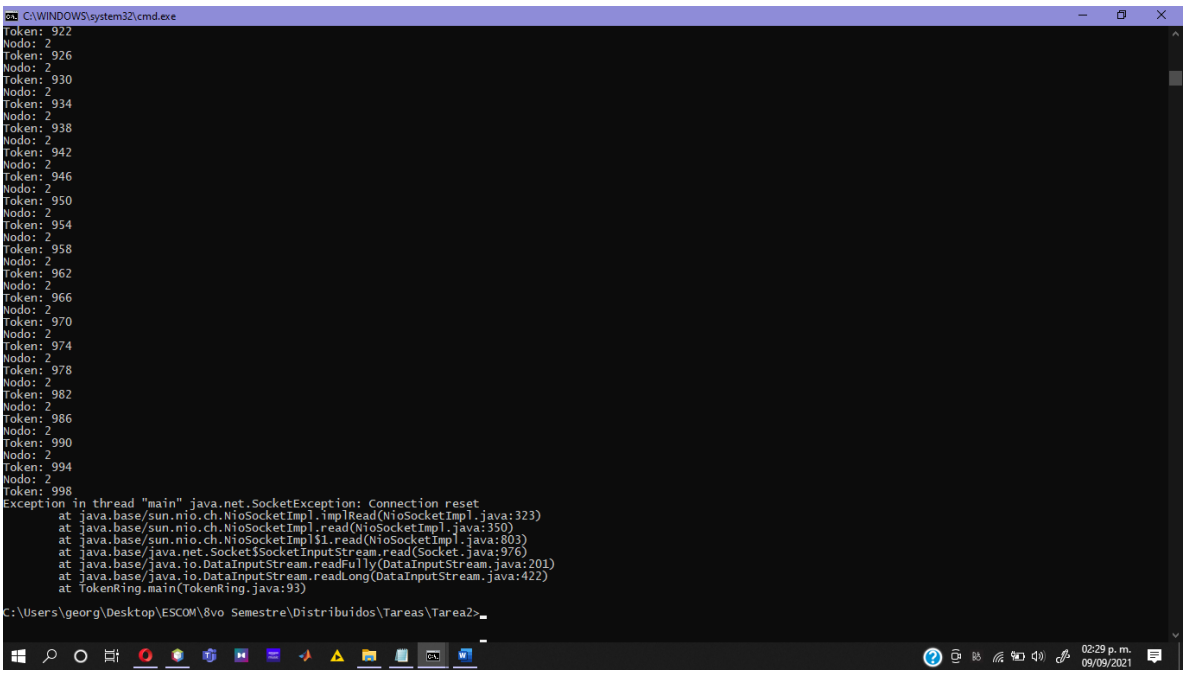
Ejecución del programa en el terminal 1 (cliente):



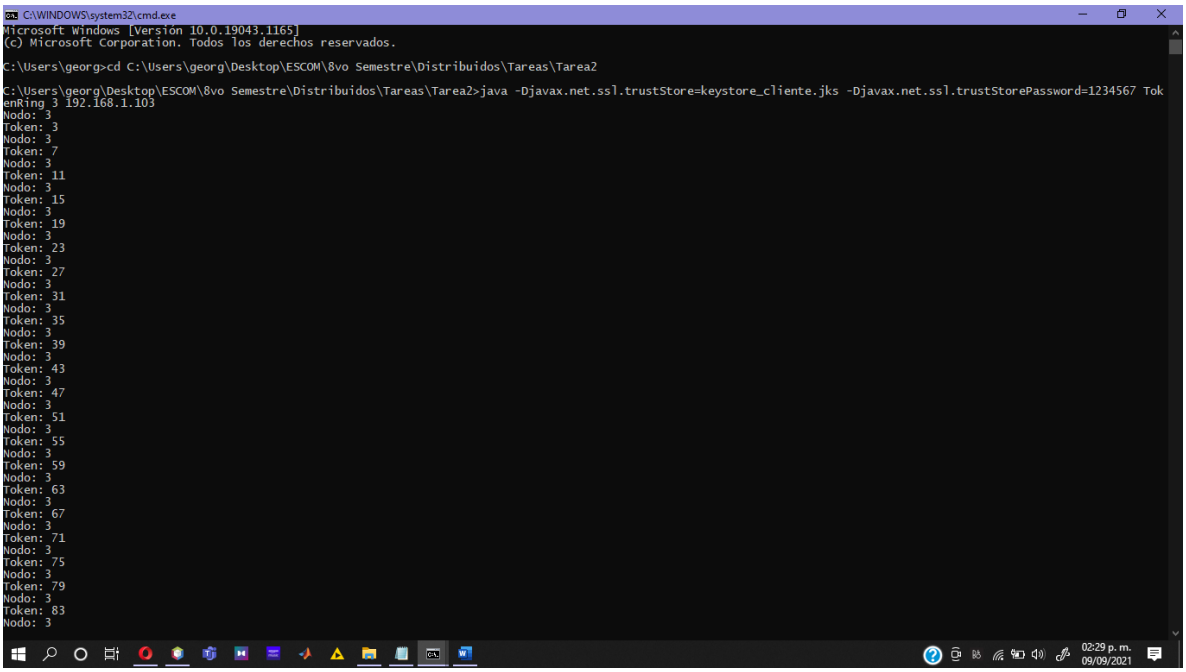


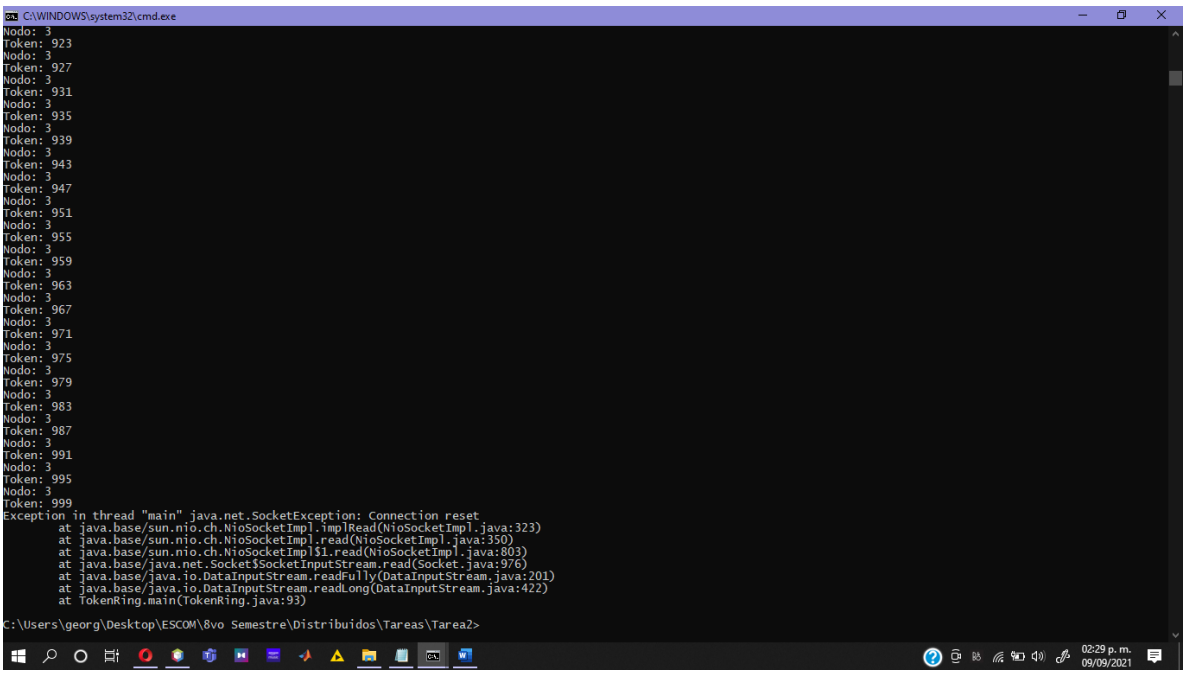
Ejecución del programa en terminal 2 (cliente):





Ejecución del programa en terminal 3 (cliente):





3. Conclusiones

Puesto que es un sistema distribuido, no importa el orden en que se ejecuten los nodos, siempre y cuando se ejecuten los 3 y no se repita ninguno para que funcione correctamente.

El problema llega cuando ejecutamos más de una vez algún cliente, porque el programa puede ejecutarse en cualquier nodo, simplemente se queda a la escucha de la ejecución del siguiente nodo iniciando el token cuando es la primera vez, se incrementa y lo manda al siguiente, este volverá a incrementar y lo manda al siguiente nodo, así se repite infinitamente en la topología de anillo. No funciona si no están conectadas entre sí, y cuando se ejecuta espera a que se conecte el siguiente nodo y no se sigue de largo, e iniciará, hasta que el último nodo se conecte al primero.