INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Desarrollo de Sistemas Distribuidos

Tarea 2

Implementación de un token-ring

Integrantes:

- Ramírez Galindo Karina
- Toledo Espinosa Cristina Aline
- Vázquez Hernández Alan Mauricio

Grupo: 4CV11

Profesor: Pineda Guerrero Carlos

Fecha de realización: 1-marzo -2022

Fecha de entrega: 1-marzo -2022

Contenido

Introducción **3**
Cliente-Servidor **3**
Socket **3**
Protocolo SSL **3**
Token **3**
Desarrollo **4**
Funcionamiento **8**
Conclusiones **11**
Toledo Espinosa Cristina Aline **11**
Vázquez Hernández Alan Mauricio **11**
Ramírez Galindo Karina **11**
Referencias **12**

Introducción

Cliente-Servidor

Como se ha visto en clase existe una arquitectura que permite la comunicación entre 2 partes, un cliente y un servidor. Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios. Un cliente realiza peticiones a un servidor, quien le da respuesta. Esto se puede aplicar a programas que se ejecutan sobre una sola máquina, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. [1]

Socket

Los sockets son los identificadores de la red de comunicaciones, son una interfaz de entrada-salida de datos que permite la intercomunicación entre procesos (emita o reciba información) incluso en máquinas distintas. La comunicación entre procesos a través de sockets se basa en la filosofía *cliente-servidor* (explicada previamente). Desde la arquitectura TCP/IP los sockets cuentan con dos elementos importantes: La dirección IP y el número de puerto. [2]

Protocolo SSL

Secure Sockets Layer, garantiza que los datos que se transfieren entre un cliente y un servidor permanezcan privados. Este protocolo permite autenticar al servidor. SSL utiliza un establecimiento de *comunicación de seguridad* para iniciar una conexión segura entre el cliente y el servidor. Durante el establecimiento de comunicación, el cliente y el servidor se ponen de acuerdo en las claves de seguridad que se deben utilizar para la sesión y los algoritmos que se deben utilizar para el cifrado. [3]

Token

Un token es una serie de los bits (componente léxico es una cadena de caracteres) que circulan en un anillo simbólico red. Cuando uno de los sistemas en la red tiene el "token", puede enviar información a las otras computadoras. Como solo hay un token para cada red de token ring, solo una computadora puede enviar datos a la vez. [4]

Desarrollo

Para el desarrollo de esta práctica se busca un programa que cumpla con la topología anillo mostrada en la *Figura 1*.

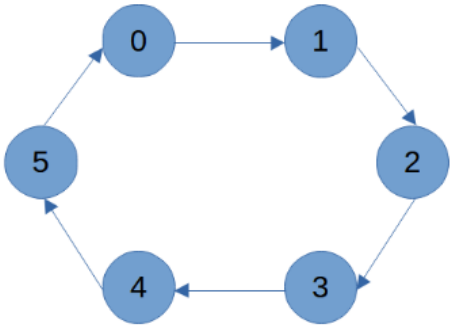

Figura 1. Topología lógica de anillo

En dicha topología cada nodo es cliente y servidor que se conectan entre sí, comenzando desde el cero y en numeración ascendente, pasando un token por cada uno de ellos. Se solicita que el nodo 0 es el que inicia la ejecución del envío de token como se observar en la *Figura 2*.

```
if (node == 0)
    sendInfo(clientOutput); //If node is 0 then start sending info
```

Figura 2. Condición inicial del programa en Java.

Una vez iniciado el proceso de incremento de token se debe recordar que en este caso en específico los clientes de los nodos únicamente enviarán información mientras que los servidores de los nodos sólo recibirán información. En la *Figura 4* se muestran los métodos para enviar y recibir el token.

Nota: "token" será el valor que previamente recibió incrementando una unidad (*Figura 3*).

```
while (true) {
    token = recieveInfo(serverInput) + 1; //Receive & increase token by 1
    System.out.println("Counter: " + token); //Print current token
    if (token >= 500 && node == 0)
        break;
    sendInfo(clientOutput); //Send token to next server
}
```

Figura 3. Incremento y envió de token.

```java
public static void sendInfo(DataOutputStream output) throws IOException {
    output.writeInt(token);
}

public static int recieveInfo(DataInputStream input) throws IOException {
    return input.readInt();
}
```

Figura 4. Métodos de envío y recepción del token Java.

Como en este caso todos los nodos serán clientes y servidores, y se ejecutará el programa en la misma computadora, es necesario conectarse mediante puertos diferentes.

Como lo muestra la *Figura 5*, cada nodo crea un servidor utilizando como referencia su propio número de nodo.

```java
try {
    server = secureServerSocket.createServerSocket( i: 5000 + node);
    connection = server.accept();
    System.out.println("The server of node: " + node
            + " has been set up succesfully");
```

Figura 5. Conexión de servidor en Java.

Por otra parte, como se observar en la *Figura 6*, cada nodo crea un cliente que se conectará al servidor del siguiente nodo (y el último nodo se conectará al primero, para completar la topología anillo).

```java
while(true) { //Waits until the connection with the server is established
    try {
        connection = secureClientSocket.createSocket( s: "localhost", i: 5000 + ((node + 1) % nodeNumber));
        System.out.println("Client connection to the node " + ((node + 1) % nodeNumber)
                + " accepted");
        break;
    } catch (IOException e) {
        Thread.sleep( millis: 100);
    }
}
return connection;
```

Figura 6. Reintento de conexión en Java.

En el caso de esta practica la comunicación se realiza mediante sockets seguros y para ello se crearon un par de llaves con ayuda de keytool como se puede observar en las *Figuras 7-10*:

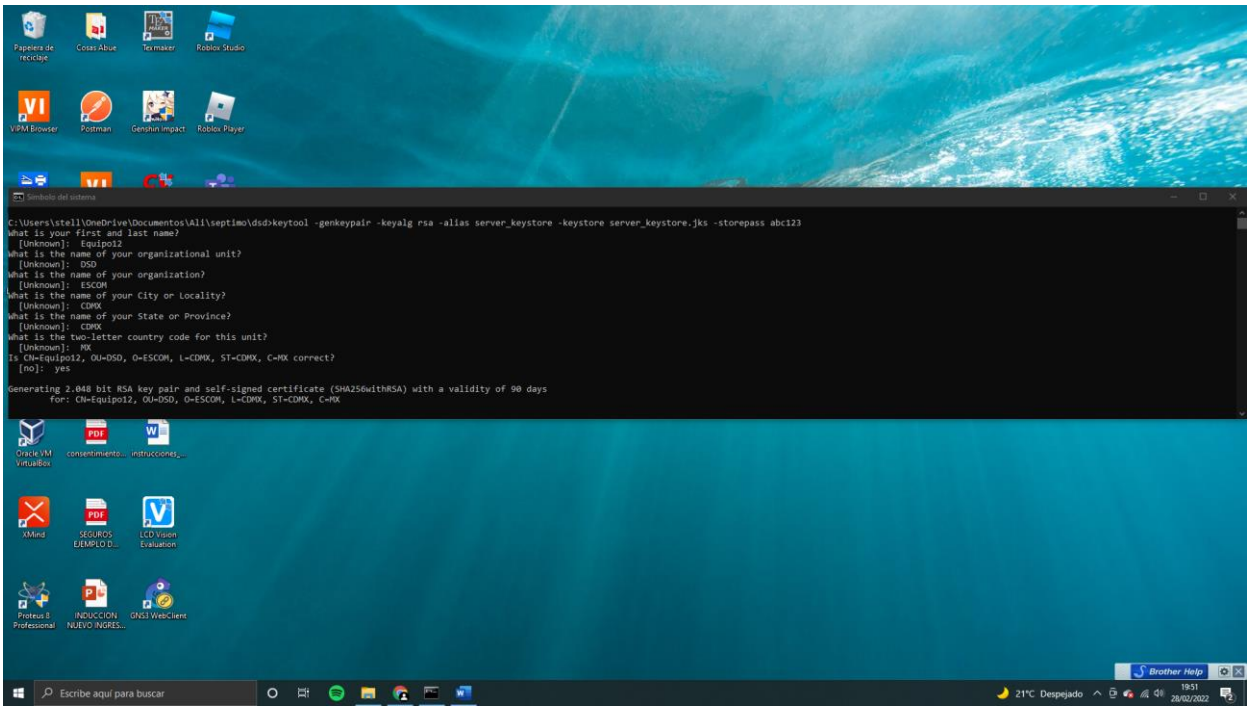

Figura 7. Creación de par de llaves.

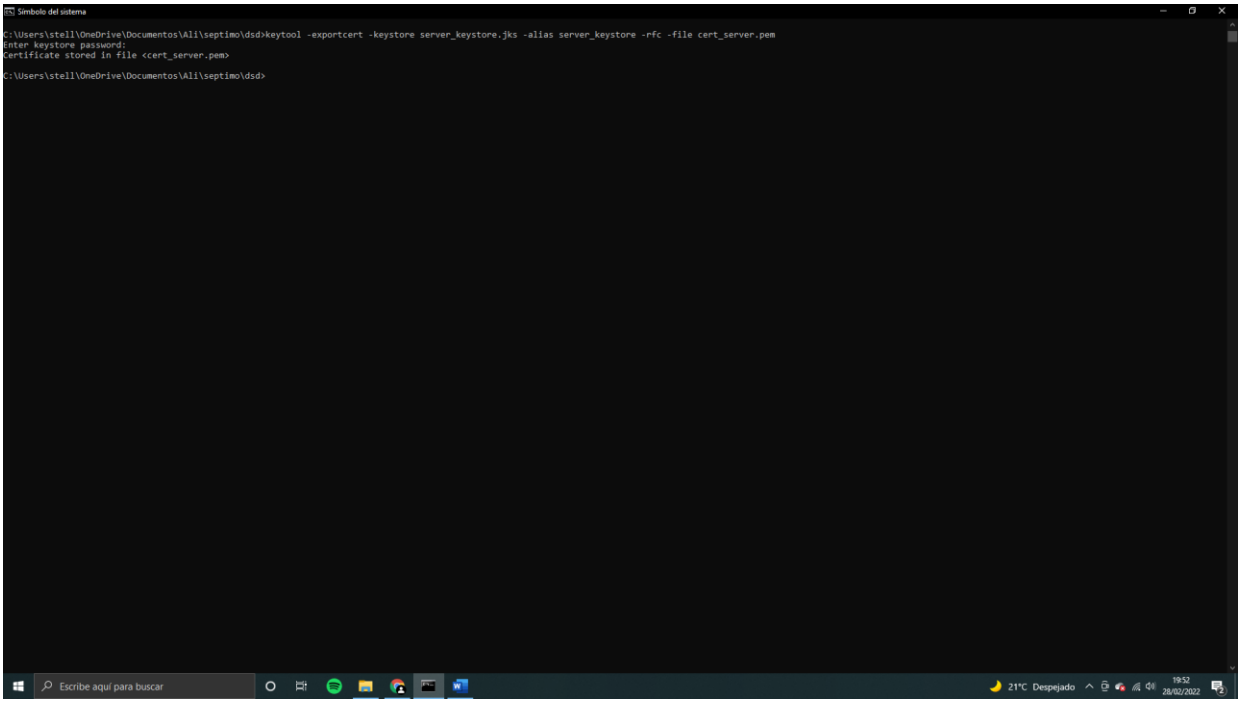

Figura 8. Exportación de certificado.

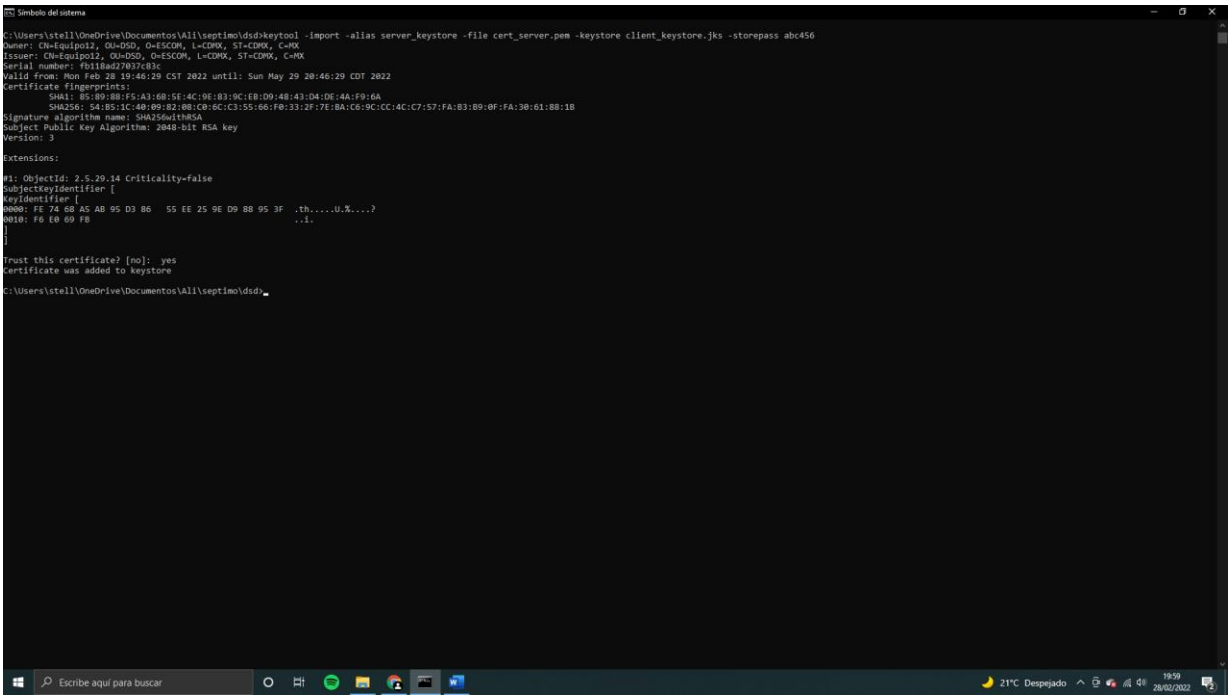

Figura 9. Importación de certificado.

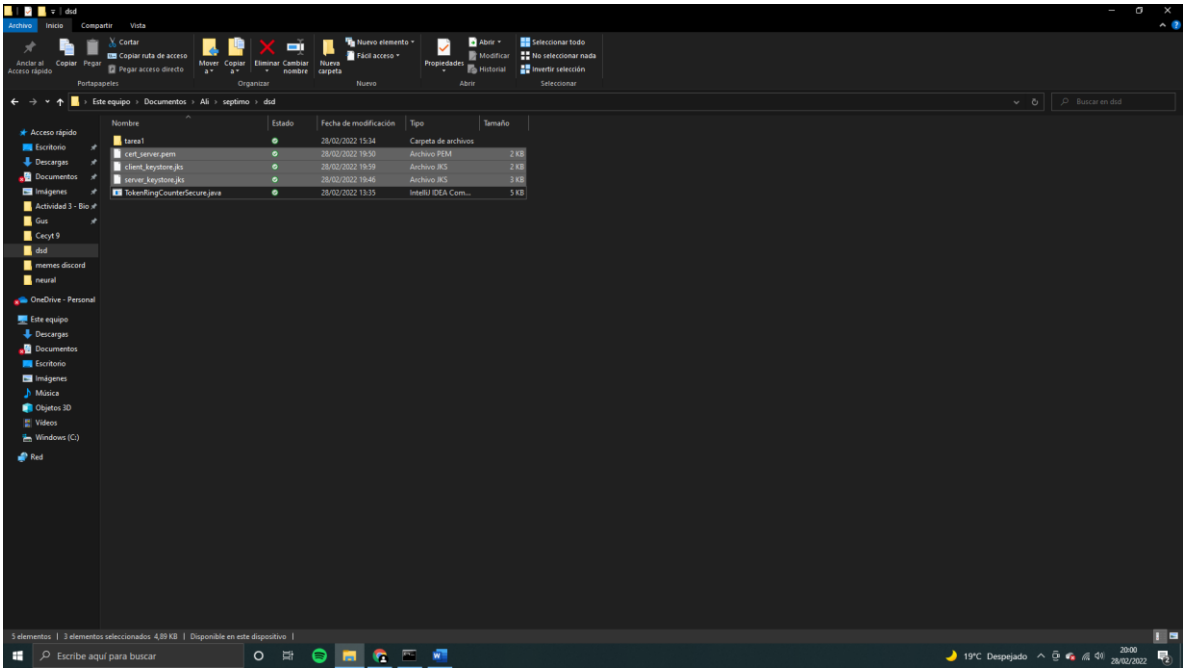

Figura 10. Keystore del cliente, servidor y certificado desde el explorador de archivos.

En la *Figura 11* se muestra la configuración de acceso a los certificados para que puedan ser utilizados por el programa.

```
//Setting the certificate credentials
System.setProperty("javax.net.ssl.trustStore","client_keystore.jks");
System.setProperty("javax.net.ssl.keyStore", "server_keystore.jks");
System.setProperty("javax.net.ssl.keyStorePassword", "abc123");
System.setProperty("javax.net.ssl.trustStorePassword","abc456");
```

Figura 11. Configuración de acceso a keystores en Java.

El programa detendrá su ejecución cuando el token sea igual o mayor a 500 en el nodo 0.

Funcionamiento

Para la ejecución de este programa se usan 6 ventanas de Windows en la cual cada una representará un nodo. Como se observa en la *Figura 12* el ultimo nodo a asignar será 0 y comenzará el envío del token.

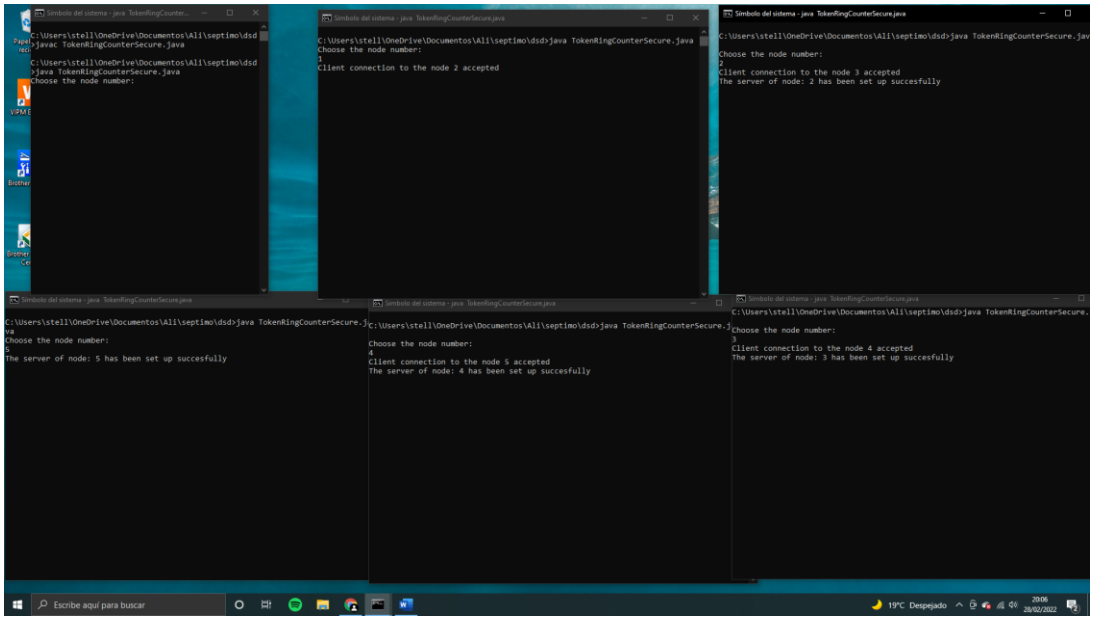

Figura 12. Ejecución del programa en 6 ventanas de comandos de Windows.

Una vez iniciado el conteo se puede observar que comienza en 0 y atravieza todos los nodos uno a uno incrementando su valor en una unidad y se detiene cuando en el nodo 0 su valor es igual o mayor a 500 (*Figura 13 y 14* respectivamente).

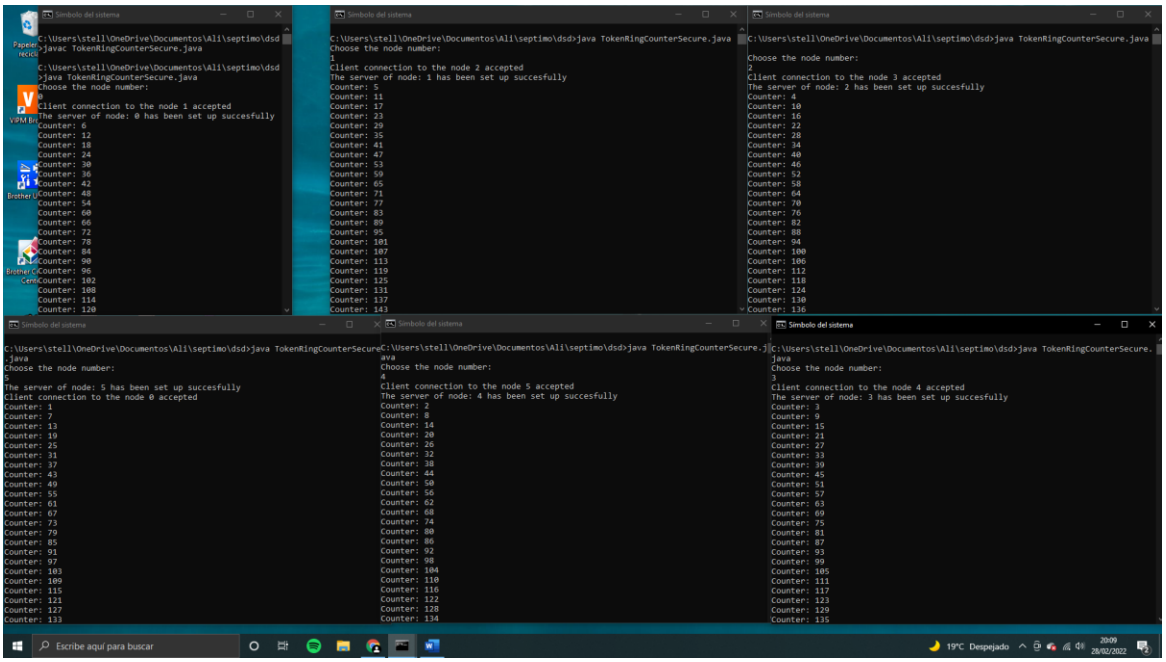

Figura 13. Ejecución del programa en 6 ventanas de comandos de Windows.

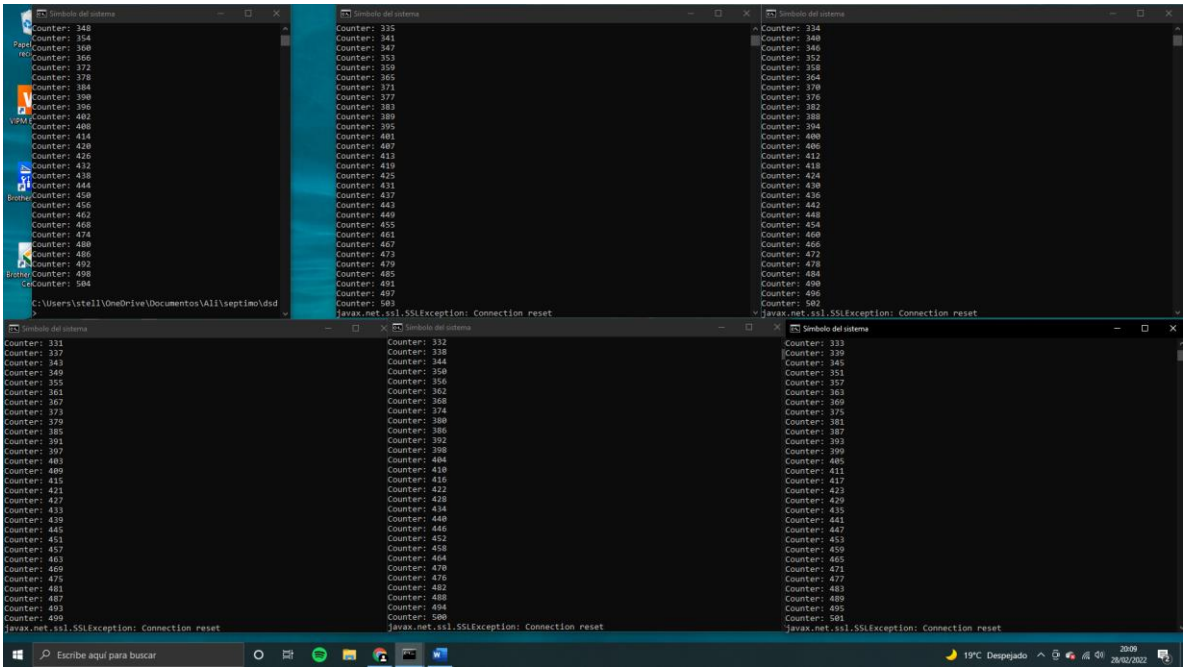

Figura 14. Ejecución del programa en 6 ventanas de comandos de Windows.

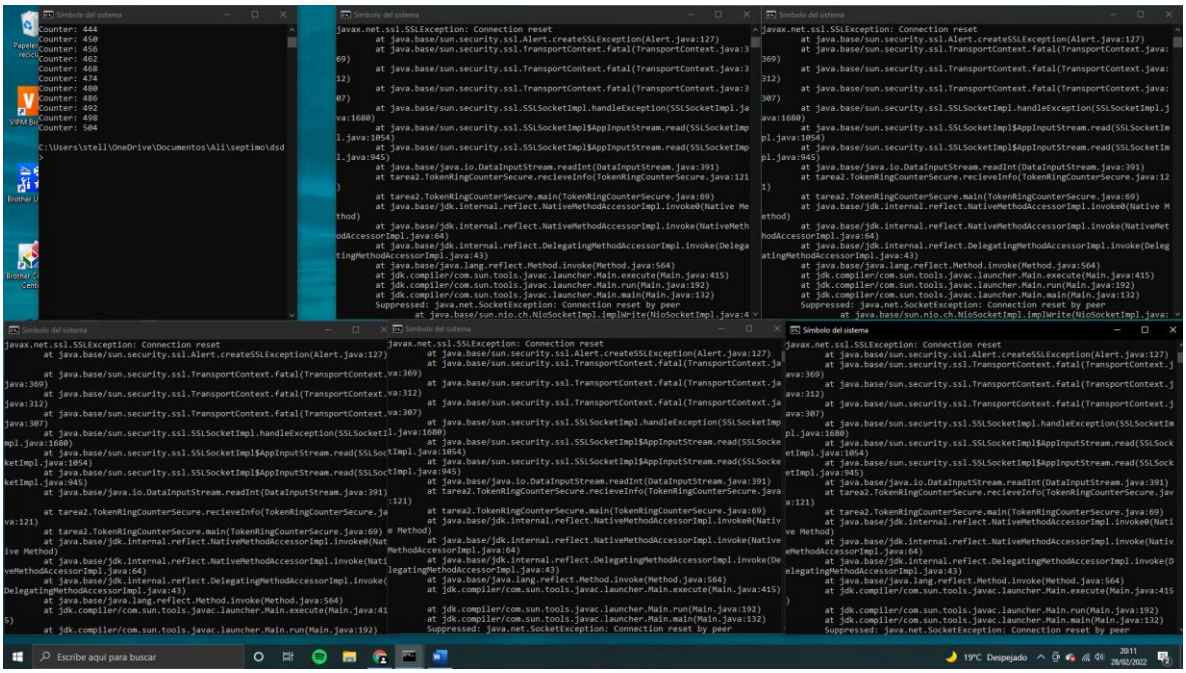

Figura 15. Ejecución del programa en 6 ventanas de comandos de Windows.

Finalmente podemos observar que al terminar el proceso el nodo 0 se desconecta lo que genera una cadena de conexiones fallidas lanzando una excepción (*Figura 15*).

Conclusiones

Toledo Espinosa Cristina Aline

El envío seguro de información es un tema muy importante ya que muchas empresas manejan información sensible que al caer en manos equivocadas podría ser desastroso por lo que debe existir un canal de comunicación segura para evitar que un mensaje sea interceptado, pero además hay que asegurar el origen del mensaje que es recibido, hoy en día existen muchos algoritmos para cifrado y creación de firmas digitales siendo RSA la más popular por su nivel de dificultad al trabajar con números primos. Otra cosa que se puede observar es el modo de operación de una topología anillo con una sola dirección en la que basta con que un servidor (representado por nodos en esta práctica) caiga para quebrantar la comunicación de todos.

Vázquez Hernández Alan Mauricio

Hoy en día, es imposible hablar de transferencia de datos por la red sin tocar el tema de la seguridad. El no implementarla significa poner en riesgo información delicada que, de caer en manos equivocadas, puede provocar grandes daños a las empresas. Por ello, el uso de herramientas como los certificados y los sockets seguros que los utilizan son de gran importancia al momento de enviar información por la red incluso si esta se encuentra dentro de la misma organización. Por otra parte, también es indispensable conocer los beneficios que nos aportan los certificados auto-firmados y los certificados emitidos por una CA (Certificate Authority). Los primeros son fáciles de crear y no tienen ningún costo, por lo que son ideales para redes dentro de una misma empresa y los otros son útiles para que aseguremos que los servicios que proveemos al público son seguros.

Ramírez Galindo Karina

Esta práctica fue de gran utilidad para tener un acercamiento a token ring que muchas veces había visto de manera teórica. El uso de este token compartido por nodos en un único sentido nos ayuda a tener un mejor control con ciertos elementos de seguridad como la generación de llaves a través de keytool y la utilización de sockets seguros, además de que tenemos la ventaja de no tener colisiones al momento de la transmisión de un nodo a otro.

Referencias

[1] Nicolas Olivares
6 junio 2015
REDES
URL: https://redespomactividad.weebly.com/modelo-cliente-servidor.html
Consultado: 25 febrero 2022

[2] Desconocido
6 agosto 2015
Sistemas Distribuidos
URL:http://wilsonvidal19.blogspot.com/p/sockets-caracteristicas-principales-los.html
Consultado: 25 febrero 2022

[3] IBM
12 agosto 2021
Protocolo SSL
URL:https://www.ibm.com/docs/es/was-zos/8.5.5?topic=SS7K4U_8.5.5/com.ibm.websphere.ihs.doc/ihs/sec_overview.html
Consultado: 25 febrero 2022

[4] techlib
N/A
Token
URL: https://techlib.net/definition/token.html
Consultado: 25 febrero 2022