



INSTITUTO POLITÉCNICO NACIONAL.
ESCUELA SUPERIOR DE CÓMPUTO.



Sistemas Distribuidos

Tarea 02: Reporte de “Implementación de un token ring”

Alumno: Oaxaca Pérez David Arturo

Grupo:

4CV12

A cargo del profesor:

PINEDA GUERRERO CARLOS

Contenido

Introducción	3
Desarrollo	4
Código	6
Capturas de pantalla mostrando la creación de los keystore	8
Captura de pantalla mostrando como se compilo el programa	10
Captura de pantalla mostrando como se ejecuto el programa en 4 ventanas distintas	11
Resultado del programa	12
Conclusiones	14

Introducción

Esta práctica consiste en desarrollar un programa que implementará un token ring, la cual es una topología de red para área local para enviar datos, siendo el token un identificador que convierte un objeto que representa una operación de control de acceso. En este caso la topología constara de 4 nodos y luce de la siguiente forma:

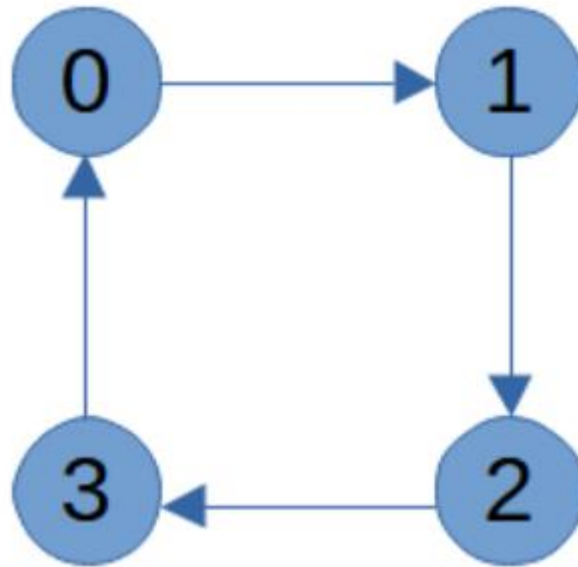


Imagen 1. Topología de estrella.

El token será un número entero de 64 bits.

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

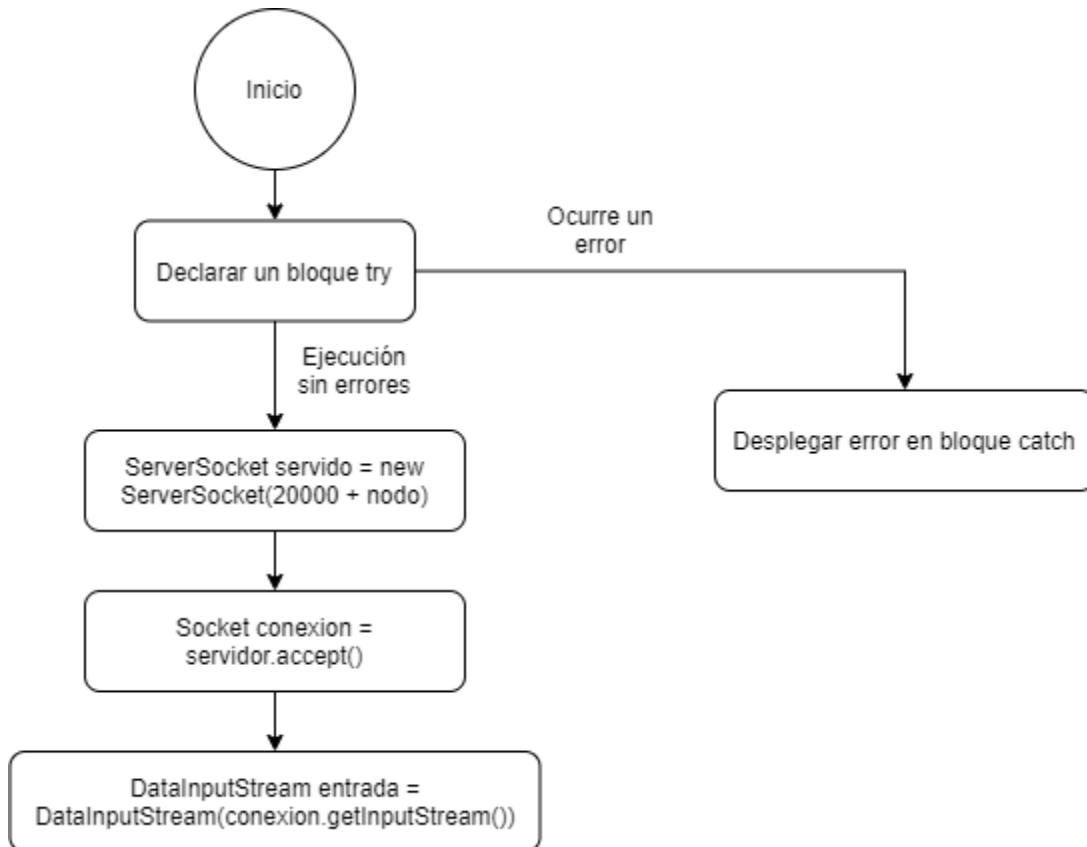
Cada nodo incrementara el token haciendo que la cuenta que empieza en el nodo 0 llegue a mil y una vez que pase eso se terminara su ejecución.

Desarrollo

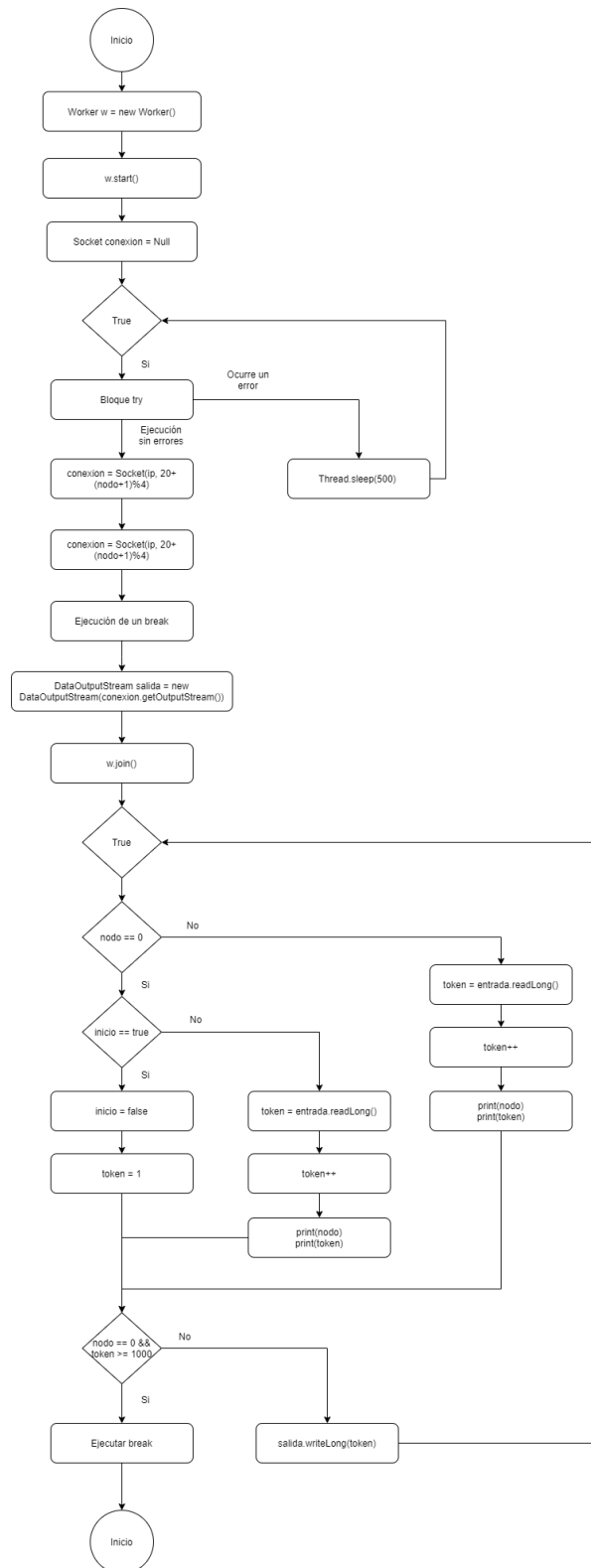
Para el desarrollo de este programa usamos nuevamente el esqueleto del programa proporcionado por la asignación de la tarea, que ya contiene la estructura de lo que serán los hilos y parte del main(). Posterior a eso desarrollare lo que son los algoritmos requeridos para hacer funcionar el programa.

Los algoritmos previamente mencionados se pueden ver en los siguientes diagramas:

Algoritmo 1:

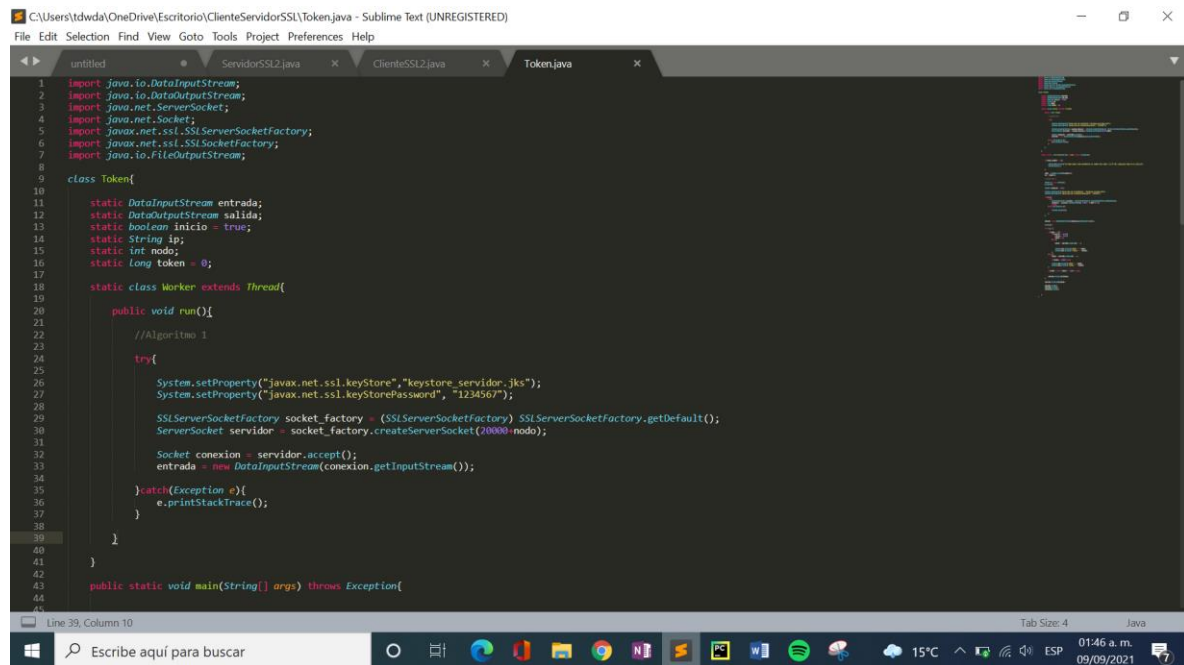


Algoritmo 2:

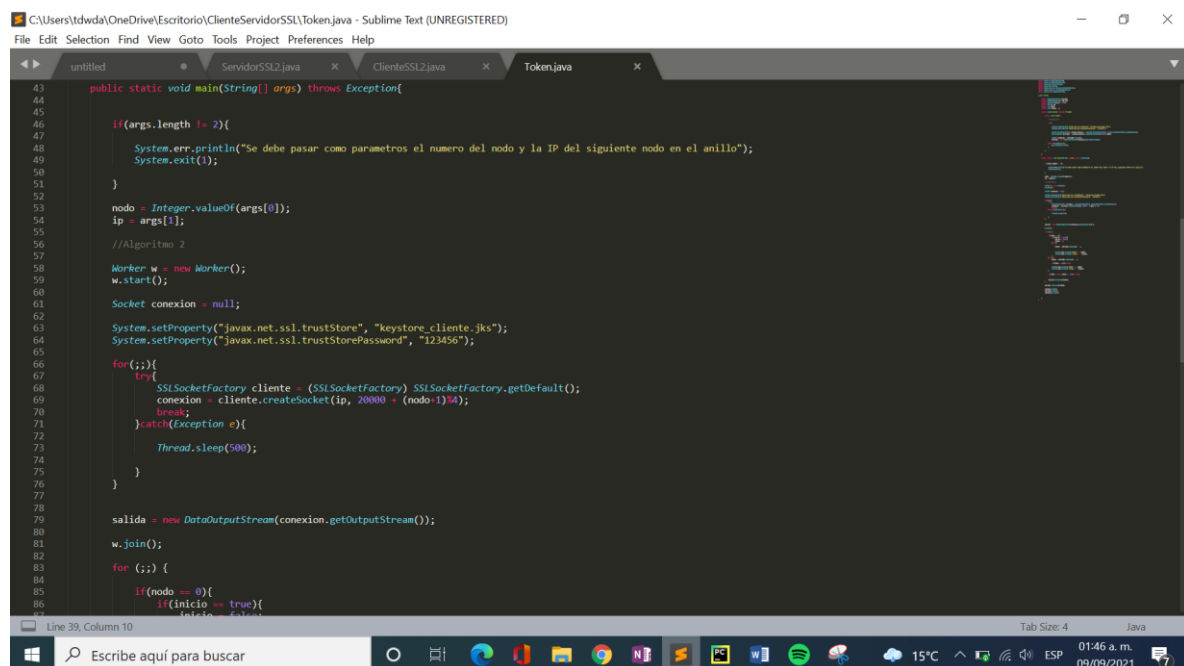


Código

Las siguientes capturas son el código completo del proyecto incluido en el archivo .zip incluido en la entrega, se ha añadido para comentar una parte que fue modificada para evitar el crasheo del programa una vez que el nodo cero se cierra al llegar al 1000, pues los demás nodos se quedan esperando una respuesta que no llegara y entonces se obtiene una excepción de connection reset o EOF.



```
1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5 import javax.net.ssl.SSLServerSocketFactory;
6 import javax.net.ssl.SSLSocketFactory;
7 import java.io.FileOutputStream;
8
9 class Token{
10
11     static DataInputStream entrada;
12     static DataOutputStream salida;
13     static boolean inicio = true;
14     static String ip;
15     static int nodo;
16     static long token = 0;
17
18     static class Worker extends Thread{
19
20         public void run(){
21
22             //Algoritmo 1
23
24             try{
25
26                 System.setProperty("javax.net.ssl.keyStore","keystore_servidor.jks");
27                 System.setProperty("javax.net.ssl.keyStorePassword", "1234567");
28
29                 SSLServerSocketFactory socket_factory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
30                 ServerSocket servidor = socket_factory.createServerSocket(20000,nodo);
31
32                 Socket conexion = servidor.accept();
33                 entrada = new DataInputStream(conexion.getInputStream());
34
35             }catch(Exception e){
36                 e.printStackTrace();
37             }
38         }
39     }
40
41 }
42
43 public static void main(String[] args) throws Exception{
44
45 }
```



```
43 public static void main(String[] args) throws Exception{
44
45     if(args.length != 2){
46         System.err.println("Se debe pasar como parametros el numero del nodo y la IP del siguiente nodo en el anillo");
47         System.exit(1);
48     }
49
50     nodo = Integer.valueOf(args[0]);
51     ip = args[1];
52
53     //Algoritmo 2
54
55     Worker w = new Worker();
56     w.start();
57
58     Socket conexion = null;
59
60     System.setProperty("javax.net.ssl.trustStore", "keystore_cliente.jks");
61     System.setProperty("javax.net.ssl.trustStorePassword", "123456");
62
63     for(;;){
64         try{
65             SSLSocketFactory cliente = (SSLSocketFactory) SSLSocketFactory.getDefault();
66             conexion = cliente.createSocket(ip, 20000 + (nodo-1)*4);
67             break;
68         }catch(Exception e){
69             Thread.sleep(500);
70         }
71     }
72
73     salida = new DataOutputStream(conexion.getOutputStream());
74
75     w.join();
76
77     for(;;){
78         if(nodo == 0){
79             if(inicio == true){
80                 inicio = false;
81             }
82         }
83     }
84 }
```

Esta es la parte modificada, al cerrarse el nodo cero envía una última vez el token antes de cerrarse y cuando los demás nodos ven que es mayor se van cerrando a su vez enviando el token hasta que el ultimo nodo se haya cerrado.

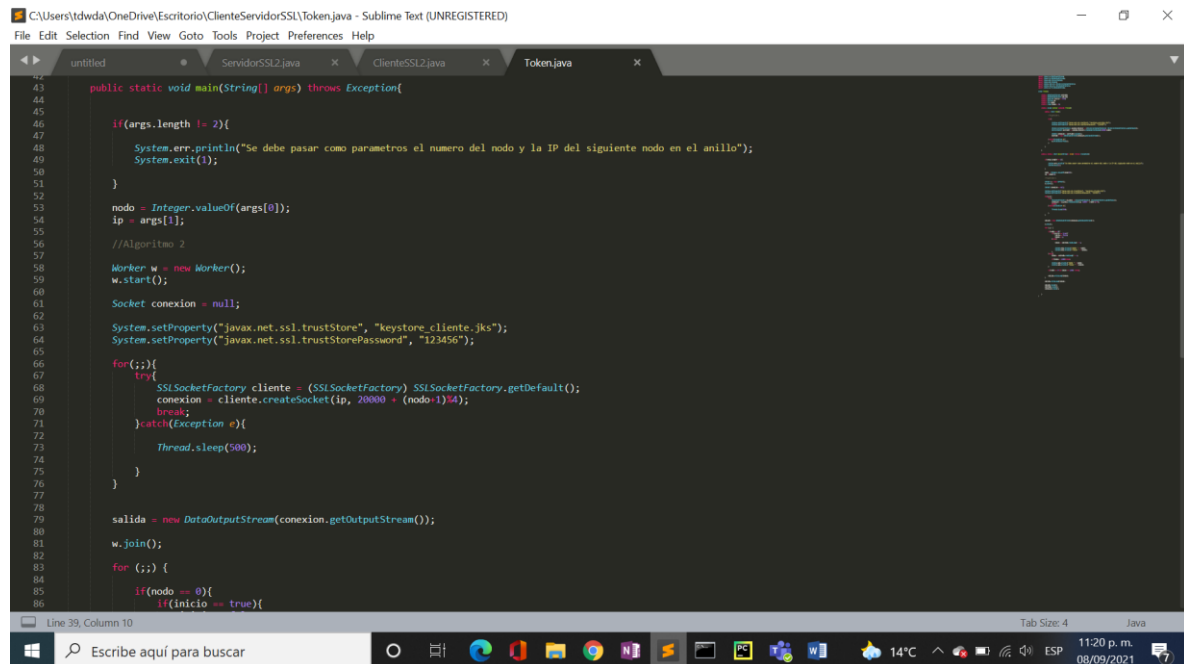
```
83 for (;;) {
84     if(nodo == 0){
85         if(inicio == true){
86             inicio = false;
87             token = 1;
88         }else{
89             token = entrada.readLong() + 1;
90         }
91         System.out.println("Nodo: " + nodo);
92         System.out.println("Token: " + token);
93     }
94 }else{
95     token = entrada.readLong() + 1;
96     if(token > 1000) break;
97     System.out.println("Nodo: " + nodo);
98     System.out.println("Token: " + token);
99 }
100 if(nodo == 0 && token >= 1000) break;
101 salida.writeLong(token);
102 }
103 salida.writeLong(token);
104 salida.close();
105 entrada.close();
106 conexion.close();
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
```

Para ejecutar los sockets seguros lo que haremos será usar la función setProperty() para usar los certificados de cliente y servidor.

La siguiente captura es la del socket seguro para servidor.

```
1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5 import javax.net.ssl.SSLServerSocketFactory;
6 import javax.net.ssl.SSLServerSocket;
7 import java.io.FileOutputStream;
8
9 class Token{
10
11     static DataInputStream entrada;
12     static DataOutputStream salida;
13     static boolean inicio = true;
14     static String ip;
15     static int nodo;
16     static long token = 0;
17
18     static class Worker extends Thread{
19
20     public void run(){
21
22         //Algoritmo 1
23
24         try{
25
26             System.setProperty("javax.net.ssl.keyStore", "keystore_servidor.jks");
27             System.setProperty("javax.net.ssl.keyStorePassword", "1234567");
28
29             SSLServerSocketFactory socket_factory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
30             ServerSocket servidor = socket_factory.createServerSocket(20000, nodo);
31
32             Socket conexion = servidor.accept();
33             entrada = new DataInputStream(conexion.getInputStream());
34
35         }catch(Exception e){
36             e.printStackTrace();
37         }
38
39     }
40
41     }
42
43     public static void main(String[] args) throws Exception{
44
45     }
```

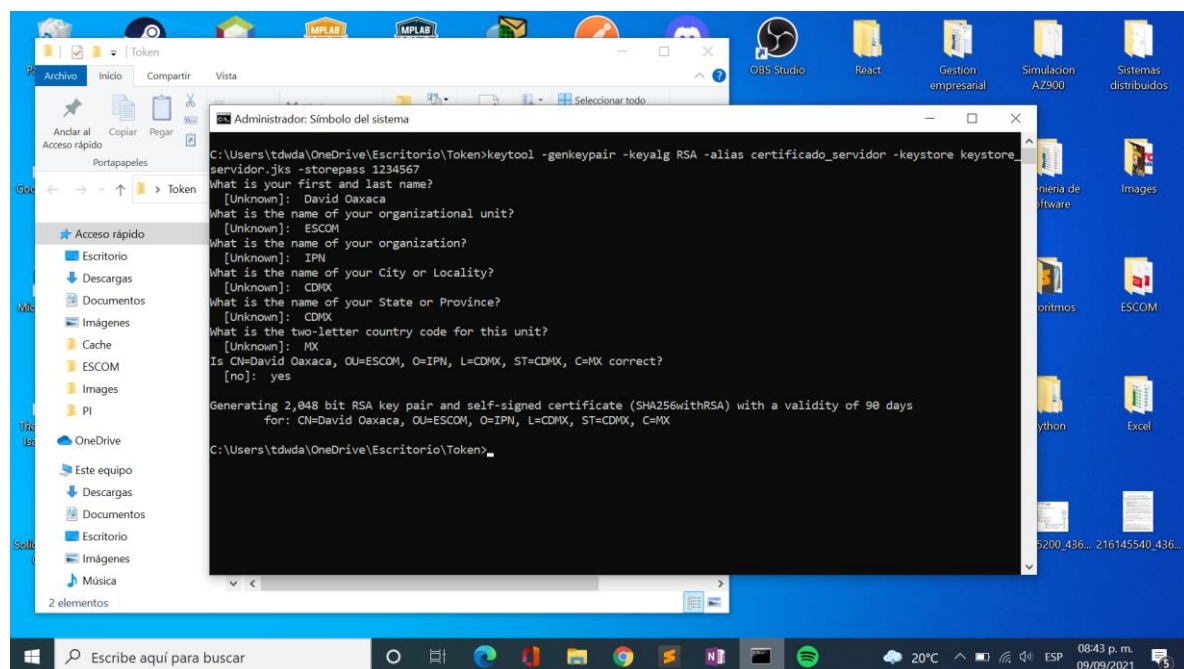
Esta captura es la del socket seguro para cliente.

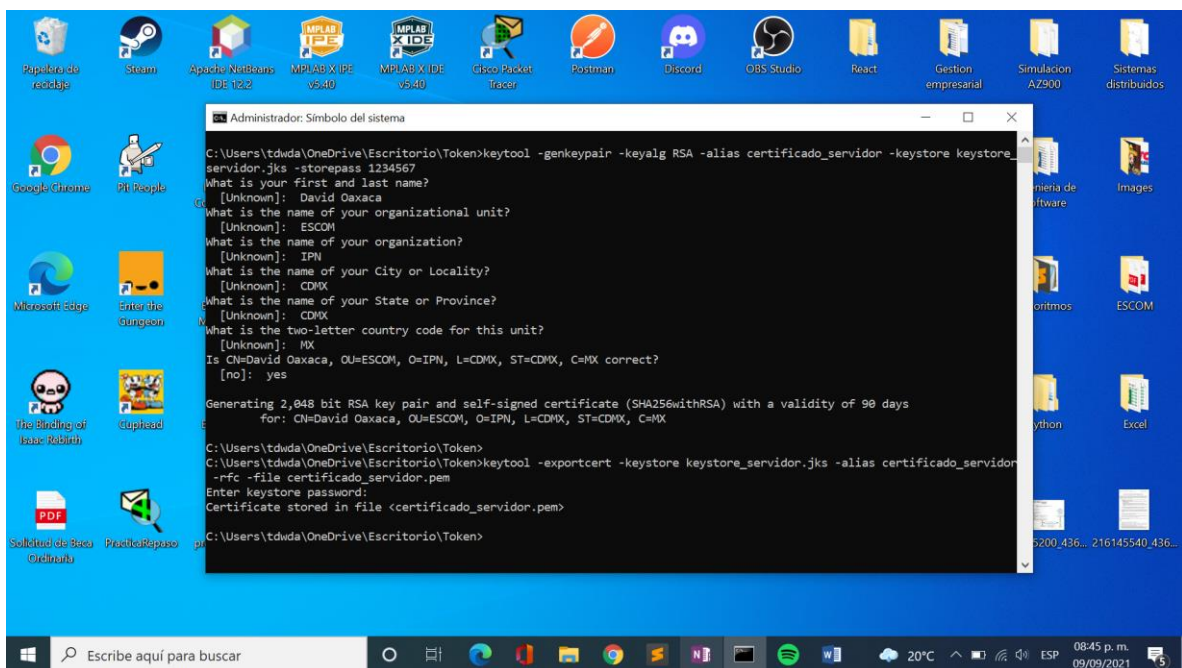
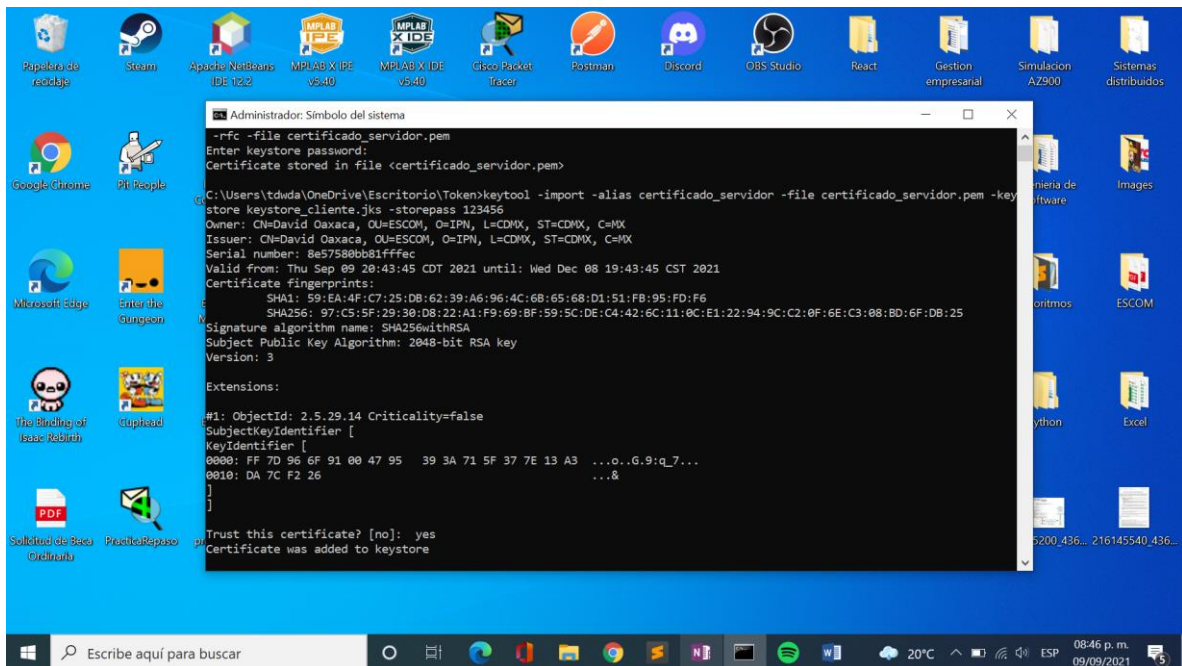


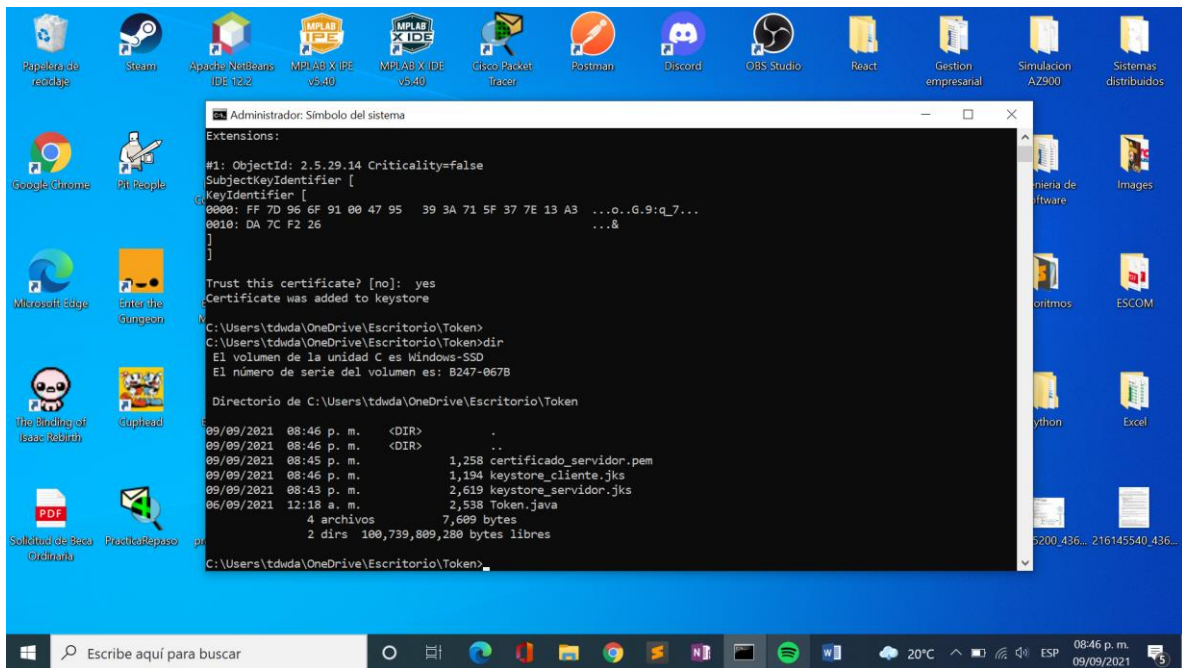
```
42
43 public static void main(String[] args) throws Exception{
44
45     if(args.length != 2){
46
47         System.err.println("Se debe pasar como parametros el numero del nodo y la IP del siguiente nodo en el anillo");
48         System.exit(1);
49     }
50
51     nodo = Integer.valueOf(args[0]);
52     ip = args[1];
53
54     //Algoritmo 2
55     Worker w = new Worker();
56     w.start();
57
58     Socket conexion = null;
59
60     System.setProperty("javax.net.ssl.trustStore", "keystore_cliente.jks");
61     System.setProperty("javax.net.ssl.trustStorePassword", "123456");
62
63     for(;;){
64         try{
65             SSLContext cliente = (SSLContext) SSLContext.getDefault();
66             conexion = cliente.createSocket(ip, 20000 + (nodo-1)*4);
67             break;
68         }catch(Exception e){
69             Thread.sleep(500);
70         }
71     }
72
73     salida = new DataOutputStream(conexion.getOutputStream());
74
75     w.join();
76
77     for(;;) {
78         if(nodo == 0){
79             if(inicio == true){
80
81
82
83
84
85
86
```

De esta manera, a la hora de ejecutar el programa simplemente se hará de forma normal.

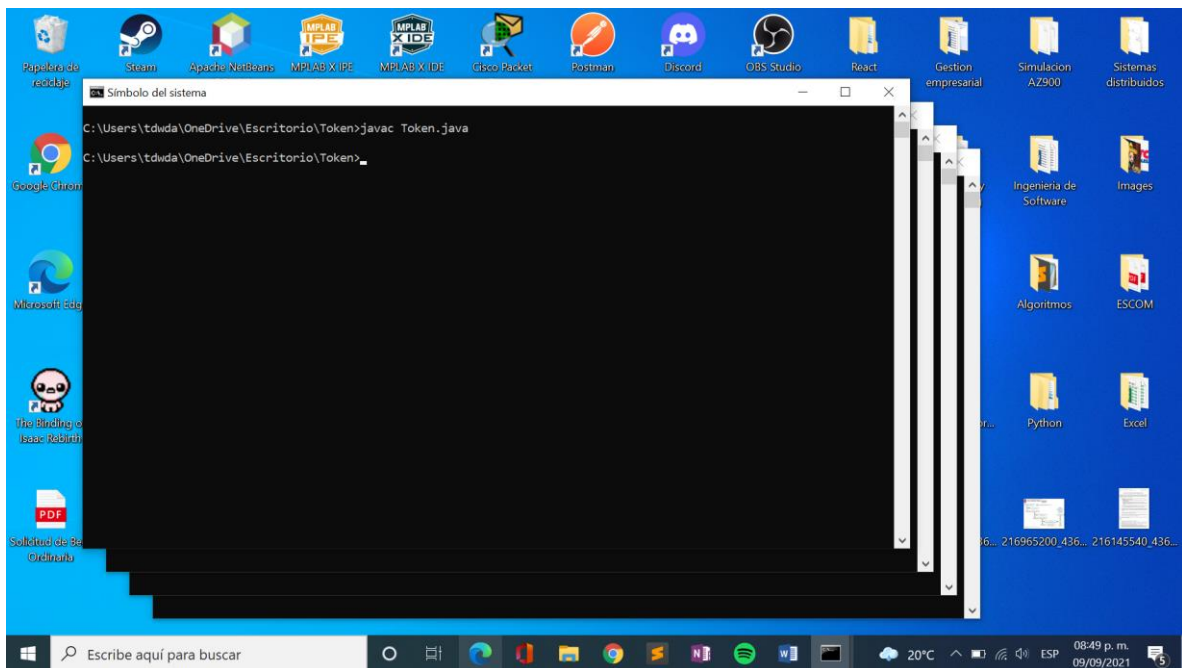
Capturas de pantalla mostrando la creación de los keystore





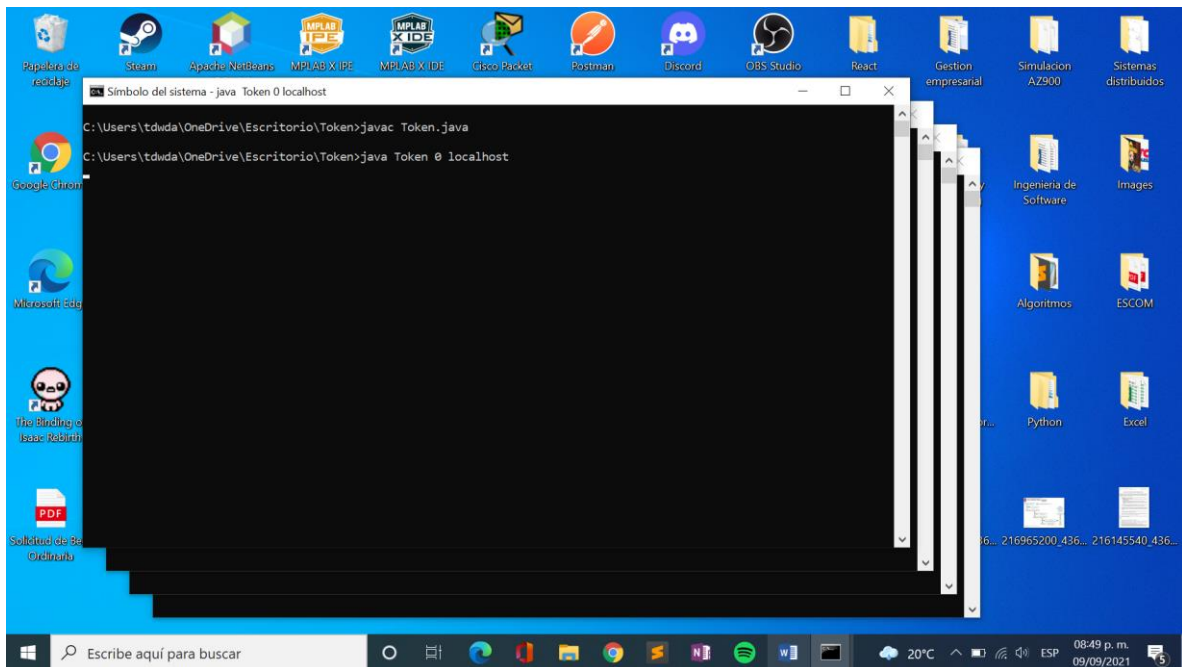


Captura de pantalla mostrando como se compilo el programa

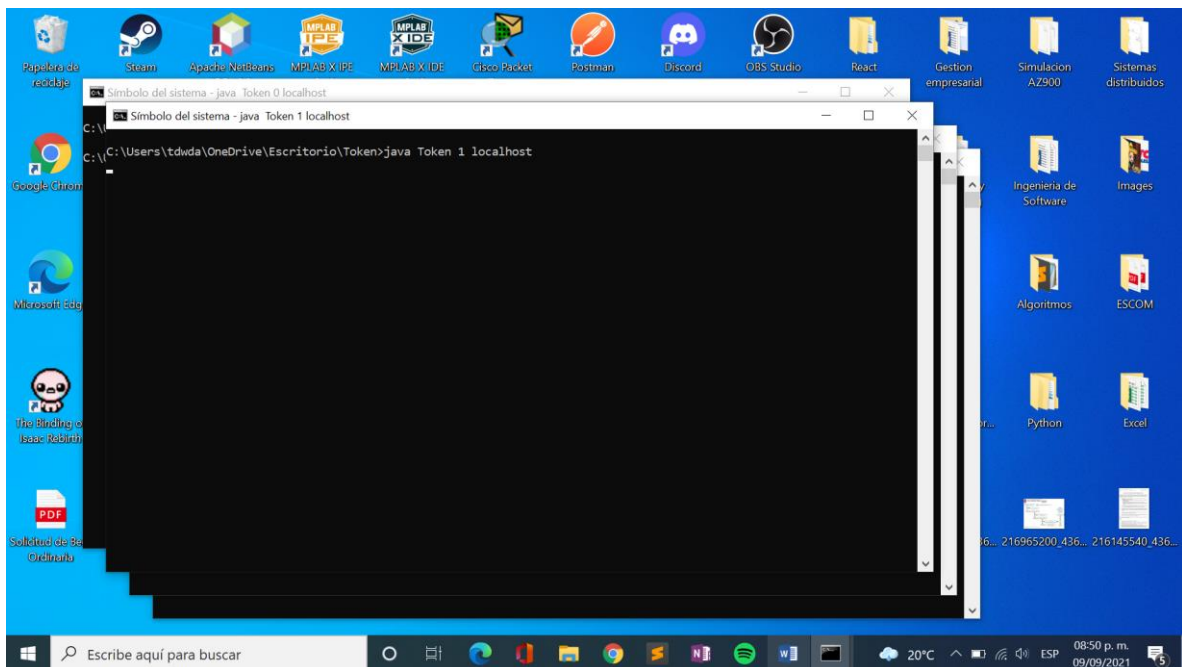


Captura de pantalla mostrando como se ejecuto el programa en 4 ventanas distintas

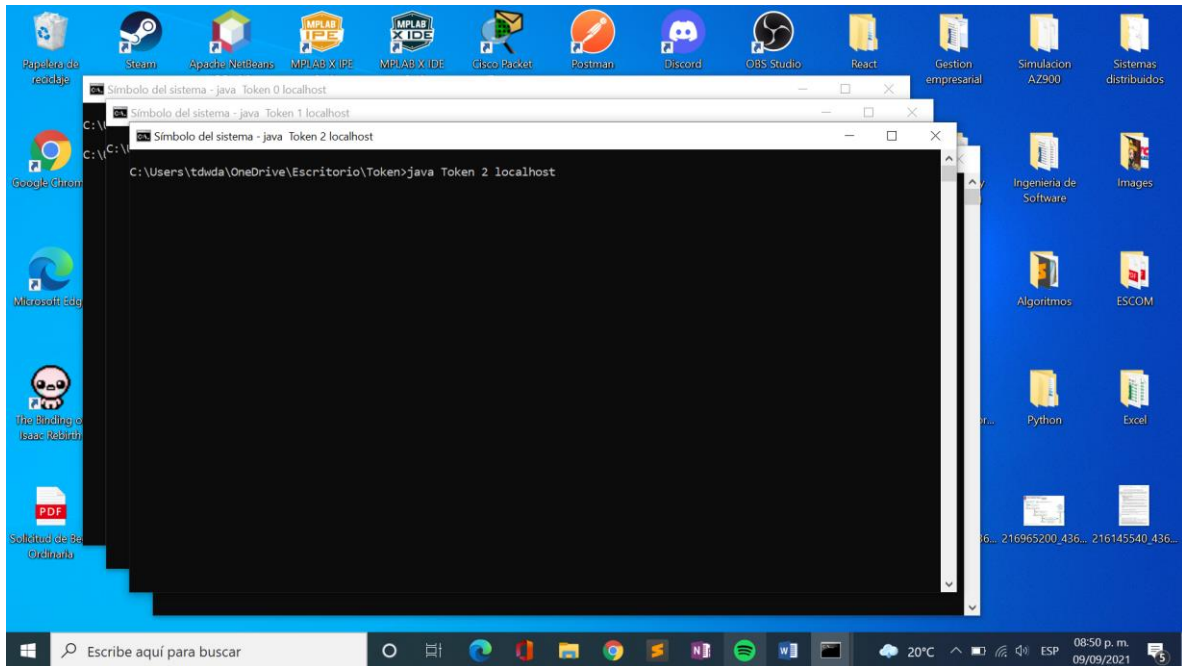
Nodo 0:



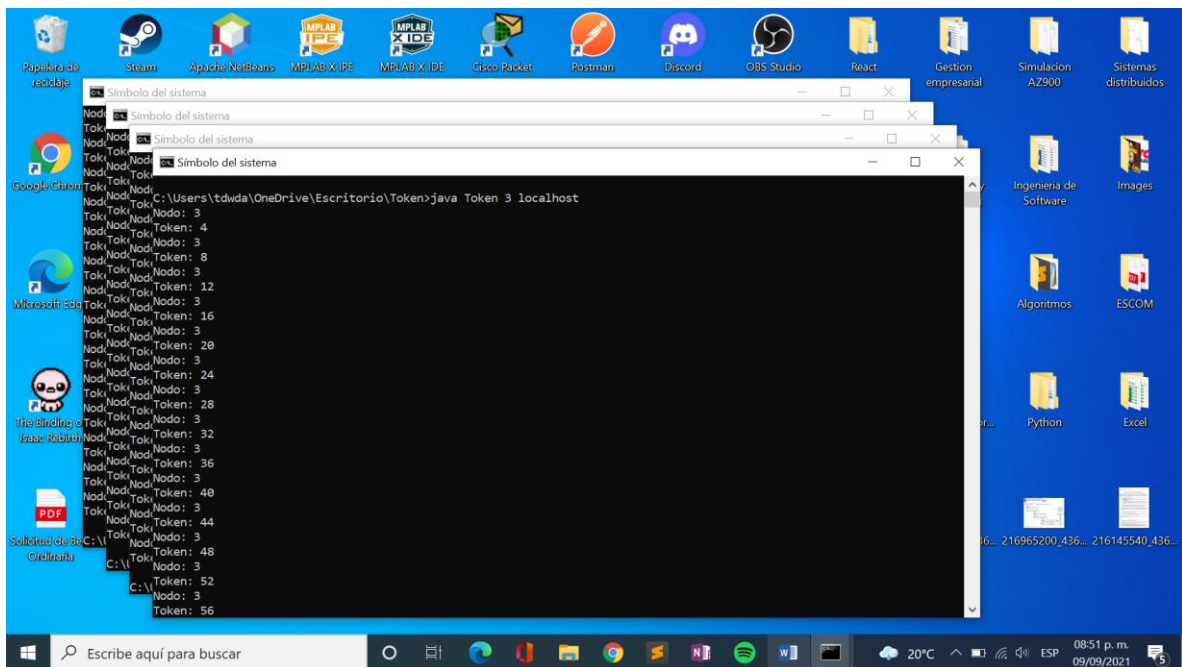
Nodo 1:



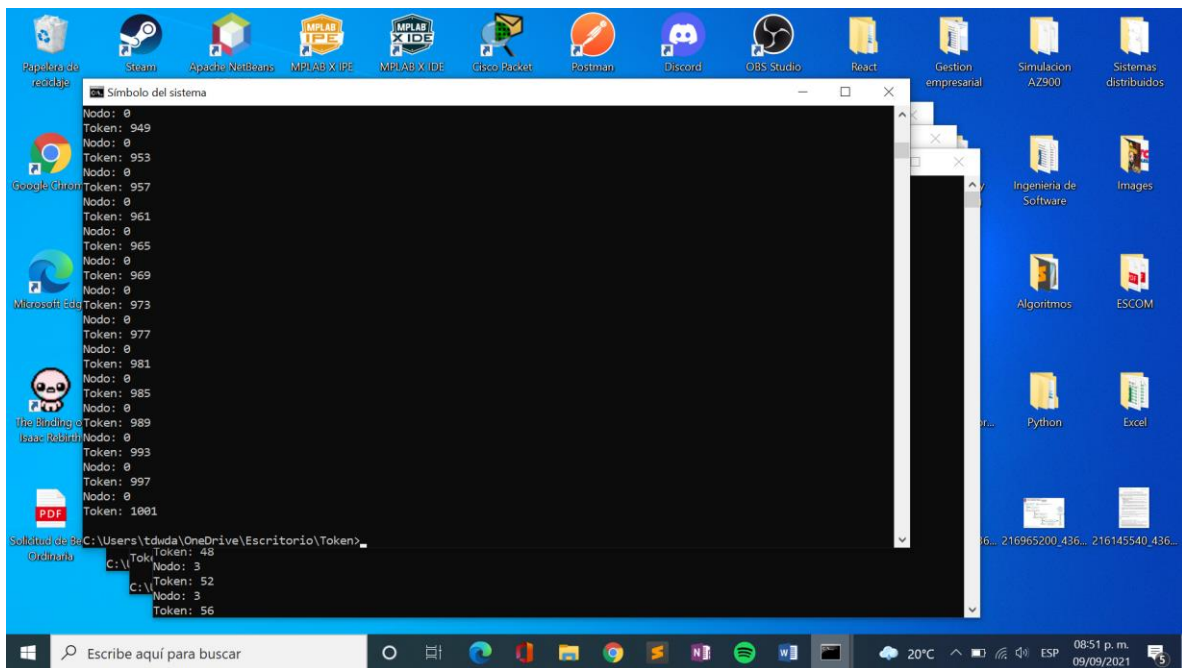
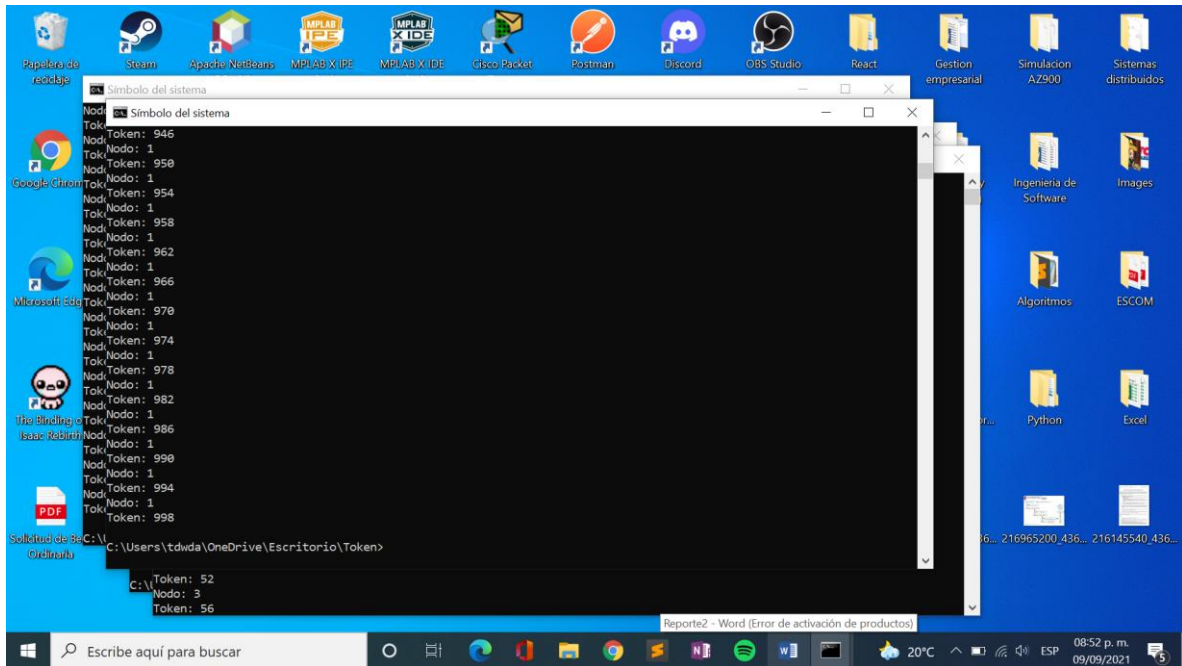
Nodo 2:

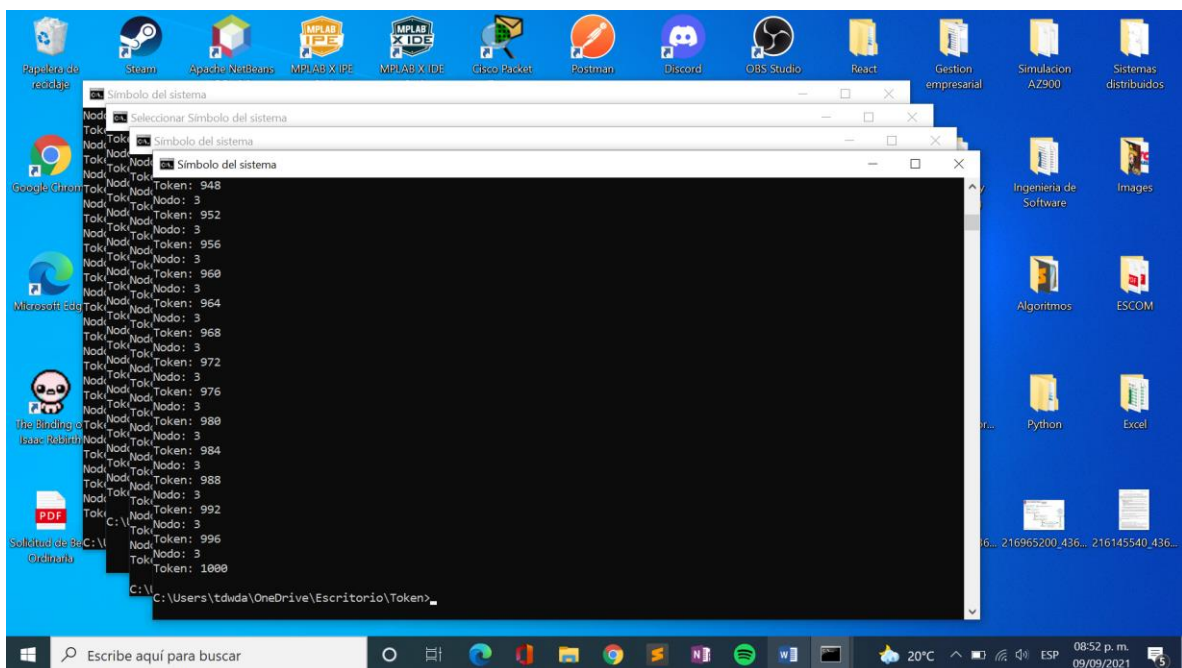
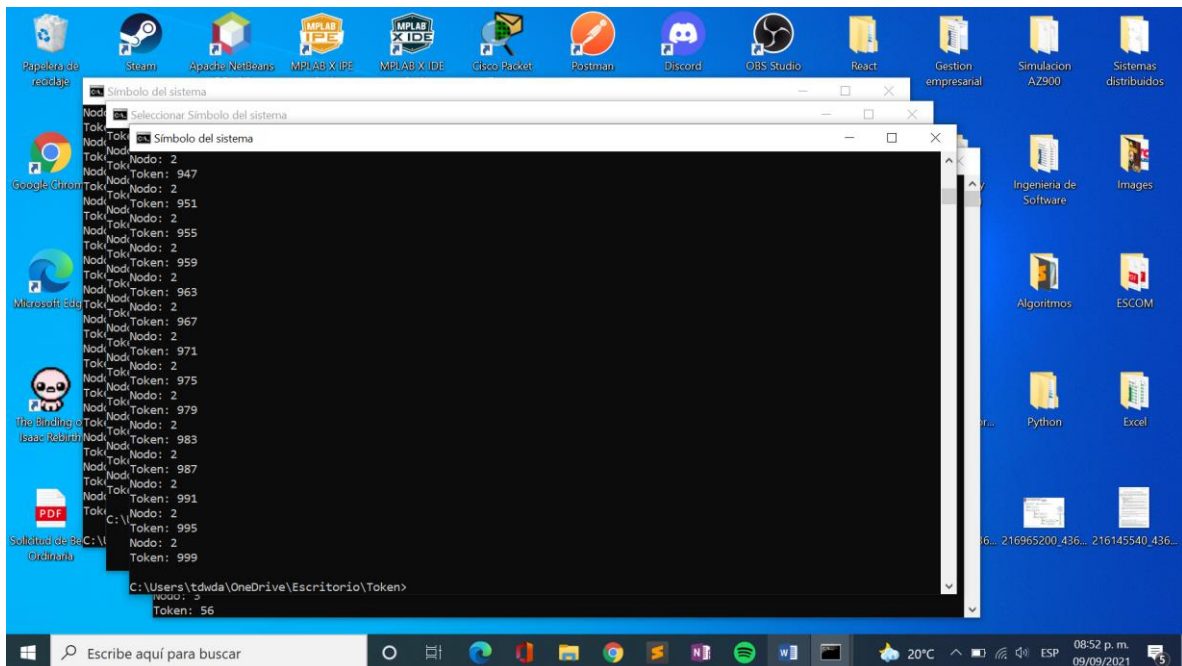


Nodo 3:



Resultado del programa





Conclusiones

Este programa resulto ser una tarea que enseñó bastante sobre la topología de token ring, sobre todo porque yo no había trabajado con esa topología, si había visto como tal la teoría, pero no había tenido un acercamiento practico. Como tal el uso de este token compartido por los nodos de la topología puede ser algo que ayude a tener un mejor control con ciertos elementos de seguridad. A lo largo de la práctica también tuve que ingeniar algunas cosas, como para hacer uso tanto de un servidor como un cliente con sockets seguros en un solo nodo usando la función setProperty o el

mensaje que evita el crasheo al cerrarse el nodo 0, en general creo que fue una práctica que ayudo mejor a comprender este tema y con el que seguimos entendiendo el uso de hilos dentro de esta clase de programas.