Programación de redes de computadoras

24 JULIO

UTPL- Sistemas Distribuidos

Creado por:

Fabricio Rafael Cantos Pérez
Cristian Sebastián Echeverria Vivanco
Carlos Humberto Guashpa Zumba
Giovanni Fernando Santana Tisalema
Wilson Alexander Morales Picoita

UTPL
UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

Contenido

Entregable 2:

Implementación de algoritmo de exclusión mutua

Integrantes del Proyecto 2

Tabla 1.

Integrantes del proyecto.

Nombre de integrantes	Porcentaje de participación
Fabricio Rafael Cantos Pérez	100%
Cristian Sebastián Echeverria Vivanco	100%
Carlos Humberto Guashpa Zumba	100%
Giovanni Fernando Santana Tisalema	100%
Wilson Alexander Morales Picoita	100%

Nota. Se presenta los nombres de cada participante y su nivel de participación.

Tabla 2.

Direcciones de red de cada participante.

Nombre de integrantes	IP Servidor	IP Ordenador 1	IP Ordenador 2	IP Ordenador 3
Fabricio Rafael Cantos Pérez	192.168.0.100	192.168.0.100 (localhost)	10.0.2.15 (Máquina Virtual)	
Cristian Sebastián Echeverria Vivanco	192.168.56.1	192.168.56.1	10.0.2.15 (Máquina Virtual)	
Carlos Humberto Guashpa Zumba	192.168.100.170	192.168.100.170	192.168.100.160	
Giovanni Fernando Santana Tisalema	192.168.72.1	192.168.72.1	192.168.255.1	192.168.72.128
Wilson Alexander Morales Picoita	192.168.100.31	192.168.100.31 (localhost)	10.0.2.15 (Máquina virtual)	192.168.100.85 (Máquina física)

Nota. Direcciones IP de cada miembro del grupo, que ejecuto el código en su propia red LAN.

Explicación del código del Reloj Lógico Vectorial

El algoritmo implementado para esta práctica se conoce como el algoritmo de exclusión mutua de Ricart-Agrawala. Este algoritmo es utilizado para coordinar el acceso a una sección crítica entre varios procesos en un sistema distribuido. Su objetivo principal es asegurar que solo un proceso pueda estar en la sección crítica en un momento dado, evitando posibles conflictos y garantizando la integridad de los datos compartidos.

Funcionamiento del algoritmo:

Inicialización:

- Cada proceso tiene una identificación única (ID) que lo distingue de otros procesos.
- Cada proceso tiene asignado un puerto de comunicación que será utilizado para establecer conexiones con otros procesos.

Conexión de los procesos:

- El servidor (RicartExclusion) está a la escucha de conexiones entrantes en el puerto especificado.
- Los clientes (Client) se conectan al servidor mediante la dirección IP y el puerto asignado al servidor.

Solicitud de acceso a la sección crítica:

- Un cliente que desea acceder a la sección crítica envía una solicitud (request) a todos los demás clientes.
- Para enviar la solicitud, el cliente crea una conexión con cada uno de los otros clientes y les envía un mensaje indicando que desea entrar en la sección crítica.

Respuestas de los otros procesos:

- Cuando un cliente recibe una solicitud de otro cliente, compara su propia prioridad con la prioridad del cliente solicitante, basándose en los IDs de los procesos.
- Si el cliente solicitante tiene una prioridad mayor (un ID más bajo), responde inmediatamente concediendo el acceso.
- Si el cliente solicitante tiene una prioridad menor (un ID más alto), responde posponiendo su respuesta hasta que él mismo haya accedido y salido de la sección crítica.

Acceso a la sección crítica:

- Un cliente puede acceder a la sección crítica cuando ha recibido respuestas afirmativas de todos los demás clientes, es decir, cuando ha recibido una respuesta positiva de todos los clientes con prioridades menores que la suya.
- Una vez dentro de la sección crítica, el cliente realiza las operaciones necesarias de manera exclusiva.

Salida de la sección crítica:

- Una vez que un cliente ha finalizado su trabajo en la sección crítica, envía un mensaje a todos los demás clientes indicando que ha salido de la sección crítica.

- Al recibir este mensaje, los otros clientes que tenían respuestas pospuestas pueden procesar su solicitud y acceder a la sección crítica si es su turno.

Mensajes de solicitud y respuesta:

- Los mensajes de solicitud y respuesta contienen información como el ID del proceso solicitante, el tipo de mensaje (solicitud o respuesta), y un timestamp para resolver posibles conflictos de prioridad.

El algoritmo de exclusión mutua de Ricart-Agrawala garantiza que ningún proceso sea bloqueado indefinidamente para acceder a la sección crítica y evita el problema del interbloqueo (deadlock) que puede ocurrir en sistemas distribuidos.

Es importante destacar que la implementación proporcionada es solo una muestra básica del algoritmo y es posible que se necesiten ajustes y mejoras para adaptarse a diferentes escenarios y condiciones de red.

Para el funcionamiento del algoritmo y la verificación de su ejecución, se debe ejecutar tanto el servidor (RicartExclusion) como los clientes (Client) en diferentes máquinas. Cada cliente intentará acceder a la sección crítica y coordinará con los demás clientes para asegurar el acceso exclusivo y ordenado a dicha sección.

Además, para demostrar la interacción y la coordinación entre los clientes, se puede realizar la captura de tráfico utilizando Wireshark en las direcciones IP y puertos correspondientes para verificar el intercambio de mensajes de solicitud y respuesta entre los procesos.

Es importante mencionar que este algoritmo de exclusión mutua puede ser más efectivo y útil en sistemas distribuidos reales con múltiples nodos y procesos, ya que, en un entorno local, como en un laboratorio virtual, pueden existir limitaciones en la concurrencia y las conexiones. Por tanto, se recomienda probar este algoritmo en un escenario distribuido más realista para evaluar plenamente su comportamiento y eficiencia en un sistema distribuido completo.

Código Fuente utilizado para el Algoritmo de Exclusión Mutua escrito en JAVA.

Para realizar esta implementación, se utilizó el código del repositorio de GitHub del usuario Madhav5589, (2015), y, además, se realiza algunas modificaciones en varias clases siguiendo las indicaciones de Chat.OpenIA. (2023), con lo cual, al final se consiguió definir una estructura que se ajuste a nuestras necesidades.

Imagen 1.

Algoritmo de Exclusión Mutua para el Servidor.

```
package com.myCompany.RicartExclusion;


import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;

public class RicartExclusion extends Thread {
    DataOutputStream dos;
    MessageSender msgSender;
    int port;
    int id;
    static int identity; // Identifying the node numbers as I get the request to connect

    public RicartExclusion(Map<Integer, Integer> ports, Integer id) {
        this.id = id;
        this.port = 8833;
    }

    @Override
    public void run() {
        ServerSocket ss;
        try {
            InetAddress ipAddress = InetAddress.getByName("192.168.100.31"); // Replace with the desired IP address
            System.out.println("Starting server on IP: " + ipAddress + ", port no: " + port);
            ss = new ServerSocket(port, 50, ipAddress);

            while (true) {
                System.out.println("Waiting for connections...");
                Socket s = ss.accept();
                SocketConnections.sockets.put(identity++, s);
                MessageReceiver m = new MessageReceiver(s, id); // Start thread to listen for incoming data
                m.start();
                dos = new DataOutputStream(s.getOutputStream());
                // dos.writeUTF("Thanks for the connect...");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

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

Nota. Se presenta el código usado para crear el Servidor en Java.

Imagen 2.

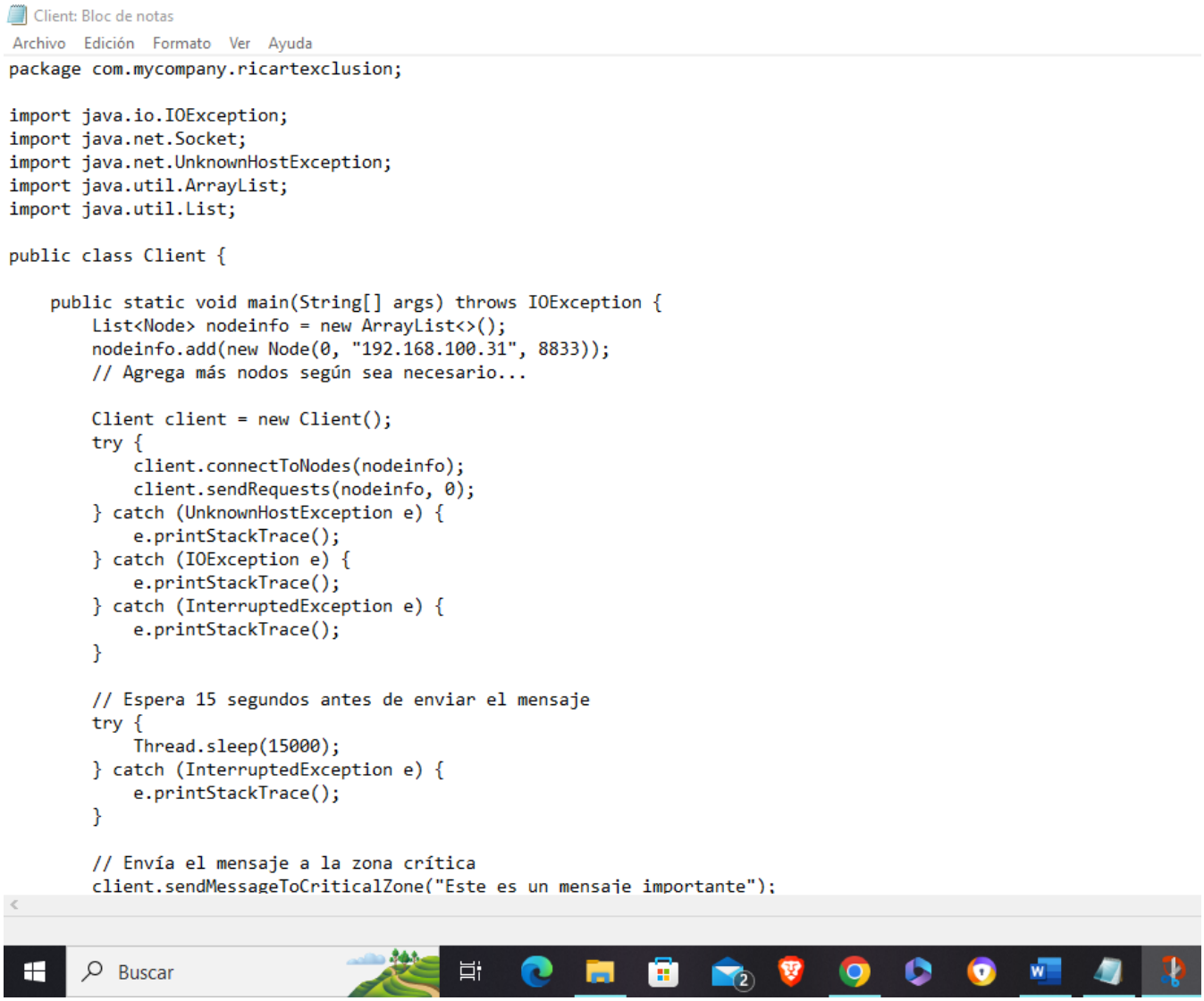
Algoritmo de Exclusión Mutua para el Servidor continuación.

```
    static int identity; // Identifying the node numbers as I get the request to connect

    public RicartExclusion(Map<Integer, Integer> ports, Integer id) {
        this.id = id;
        this.port = 8833;
    }

    @Override
    public void run() {
        ServerSocket ss;
        try {
            InetAddress ipAddress = InetAddress.getByName("192.168.100.31"); // Replace with the desired IP address
            System.out.println("Starting server on IP: " + ipAddress + ", port no: " + port);
            ss = new ServerSocket(port, 50, ipAddress);

            while (true) {
                System.out.println("Waiting for connections...");
                Socket s = ss.accept();
                SocketConnections.sockets.put(identity++, s);
                MessageReceiver m = new MessageReceiver(s, id); // Start thread to listen for incoming data
                m.start();
                dos = new DataOutputStream(s.getOutputStream());
                // dos.writeUTF("Thanks for the connect...");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        // Inicialización del mapa de puertos con ID de nodo como clave y número de puerto como valor
        Map<Integer, Integer> ports = new HashMap<>();
        ports.put(0, 8888);
        ports.put(1, 8889);
        ports.put(2, 8890);
        // ... Agrega más entradas según tus necesidades ...

        // ID del nodo deseado
        int id = 0; // Cambia este valor según el nodo deseado

        RicartExclusion ricartExclusion = new RicartExclusion(ports, id);
        ricartExclusion.start(); // Inicia el hilo
    }
}
```

Nota. Se presenta el código usado para crear el Servidor en Java, continuación.

Imagen 3.

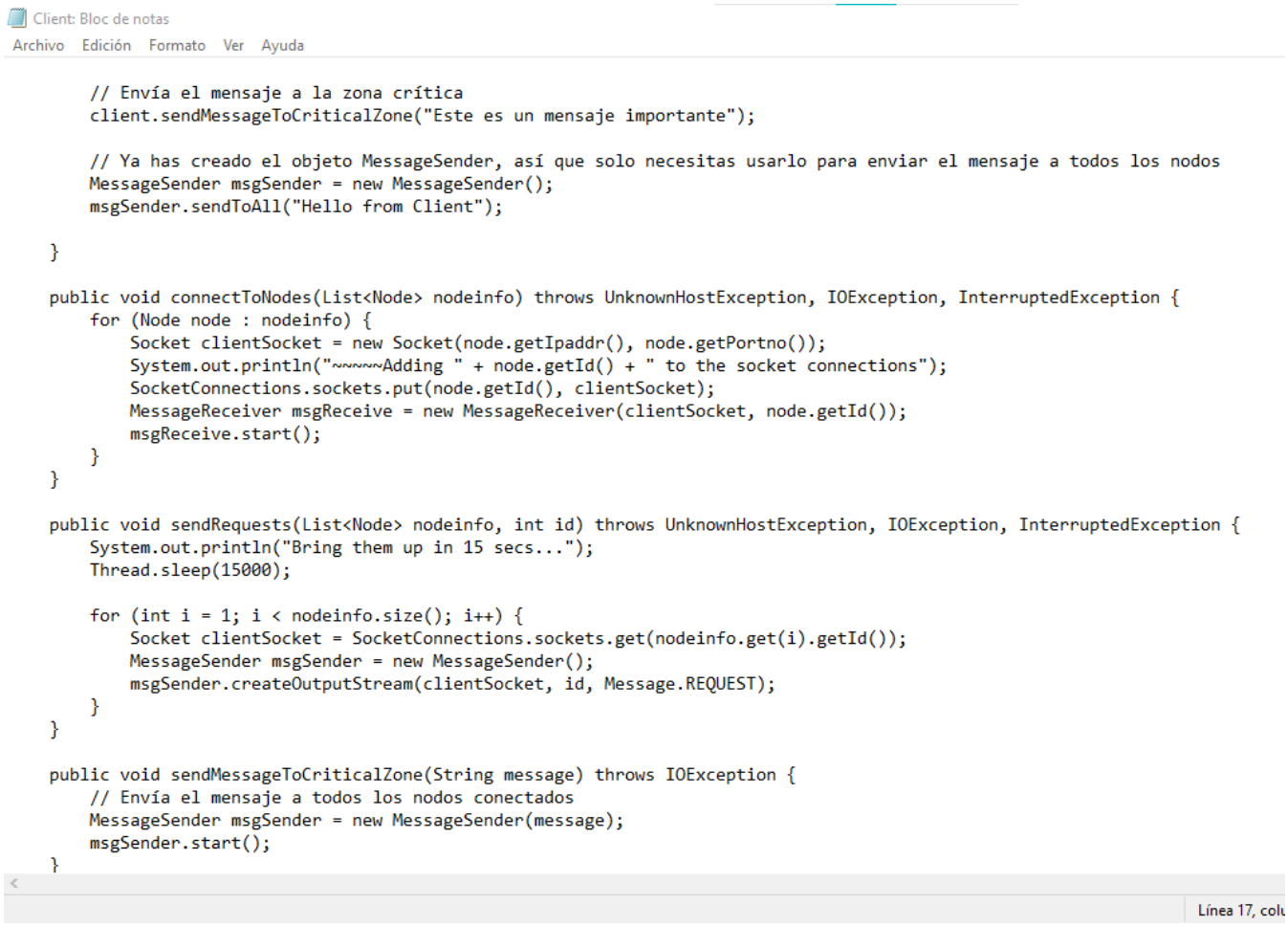
Algoritmo de Exclusión Mutua del Cliente.



Nota. Se presenta el código usado para crear el Cliente en Java.

Imagen 4.

Algoritmo de Exclusión Mutua del Cliente continuación.

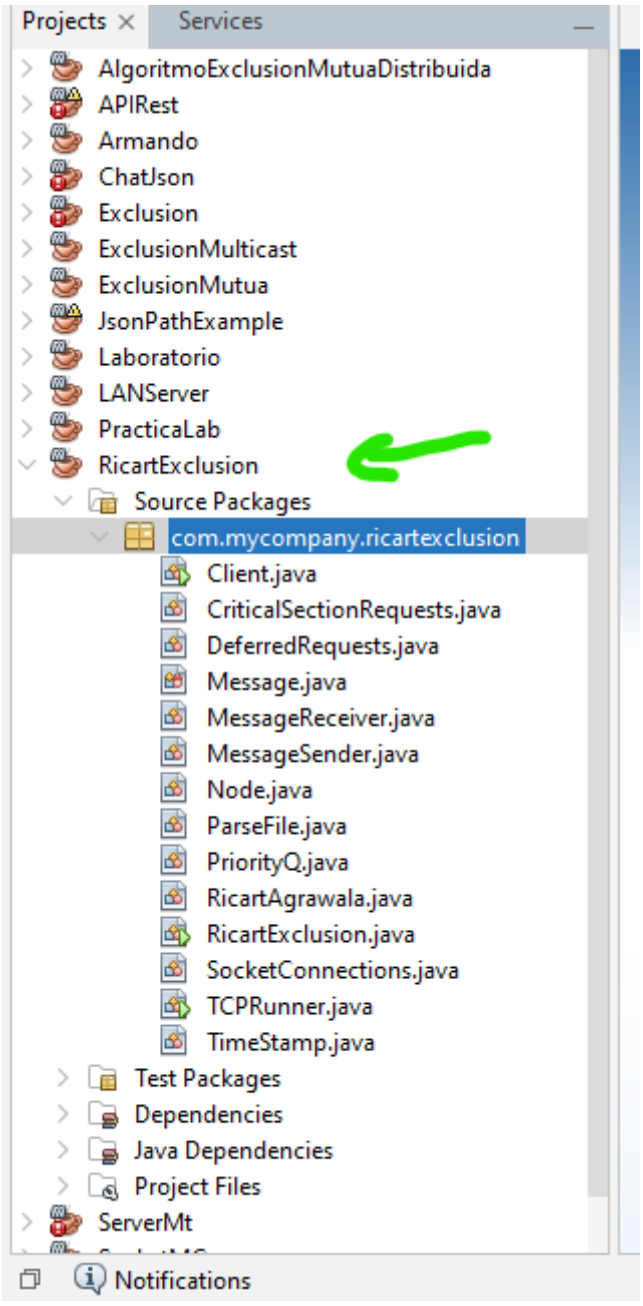


Nota. Se presenta el código usado para crear el Cliente en Java.

El resto del código del proyecto, lo puede encontrar en el repositorio de GitHub de Wilson-Morales, (2023), que es uno nuestros compañeros del grupo, el cual, como habíamos dicho, fue adaptado para cumplir los requisitos de este trabajo.

Imagen 5.

Estructura del proyecto.

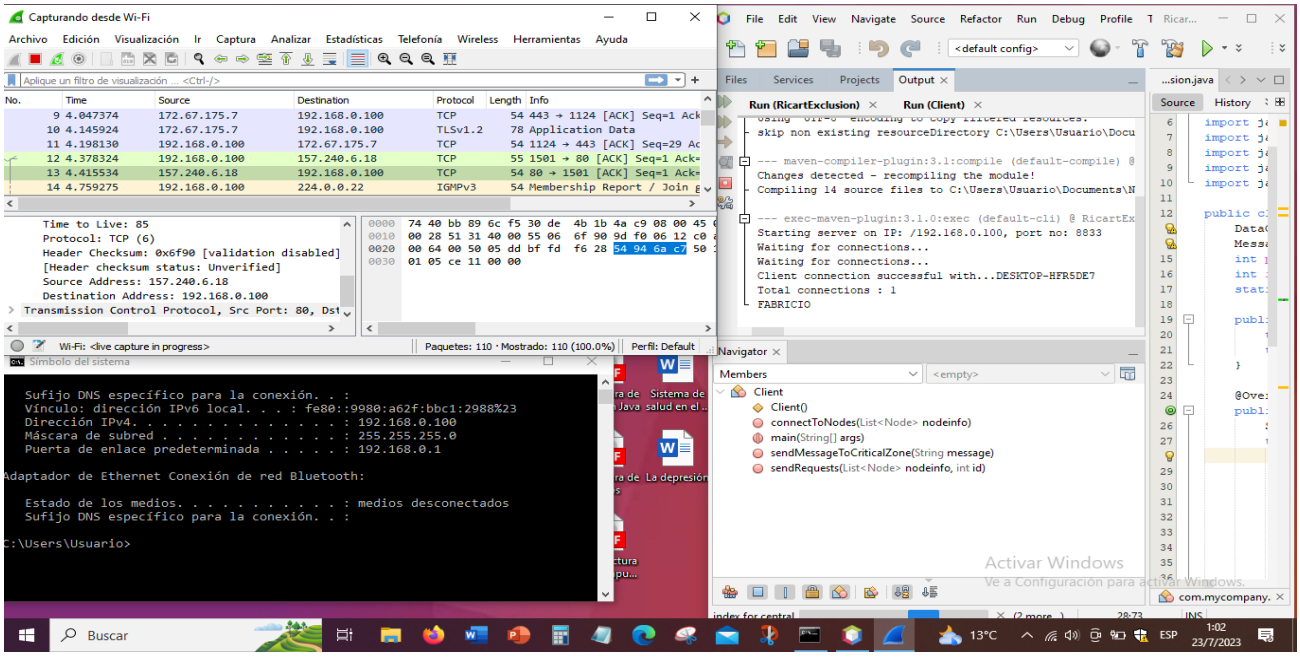


Nota. Código utilizado para construir el proyecto

Ejecución del código por cada miembro.

Imagen 6.

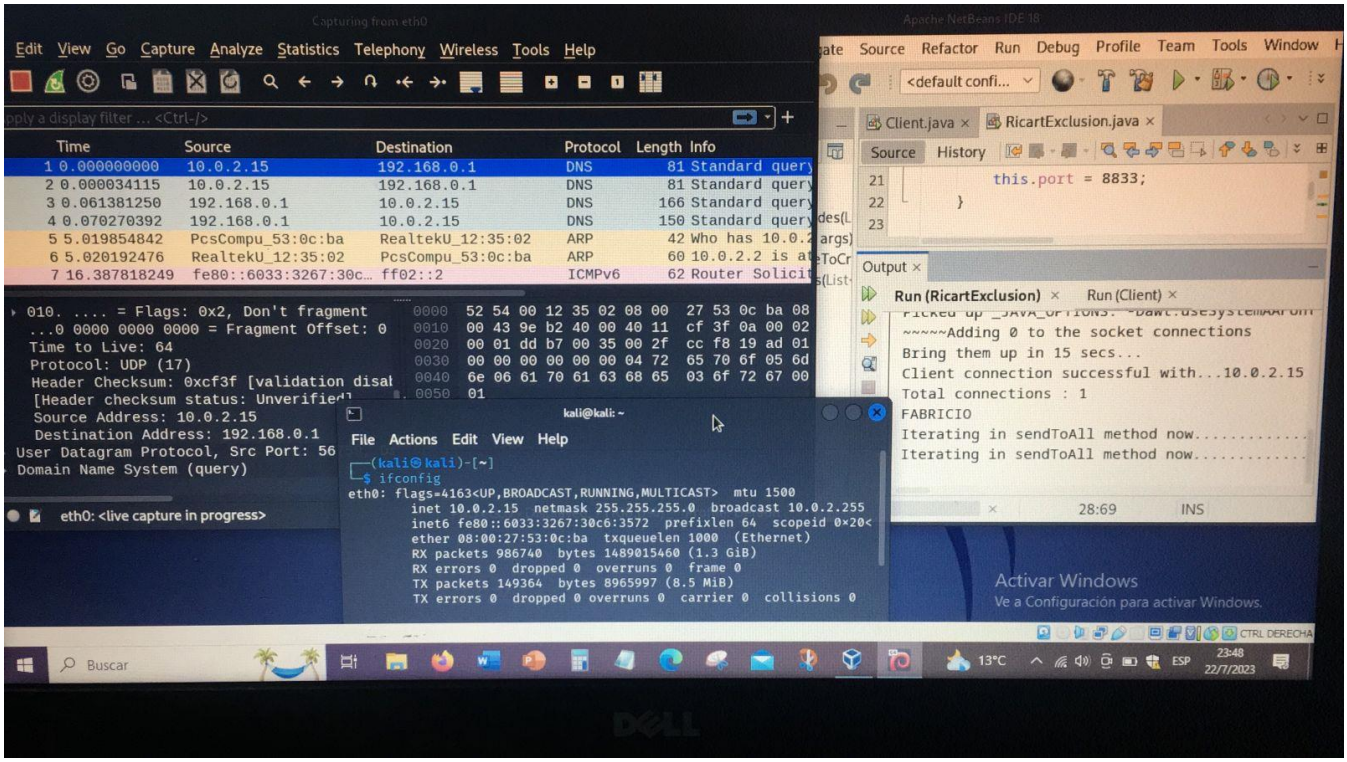
Captura de pantalla Fabricio Rafael Cantos Pérez.



Nota. El compañero muestra la conexión en localhost desde la Máquina Física.

Imagen 7.

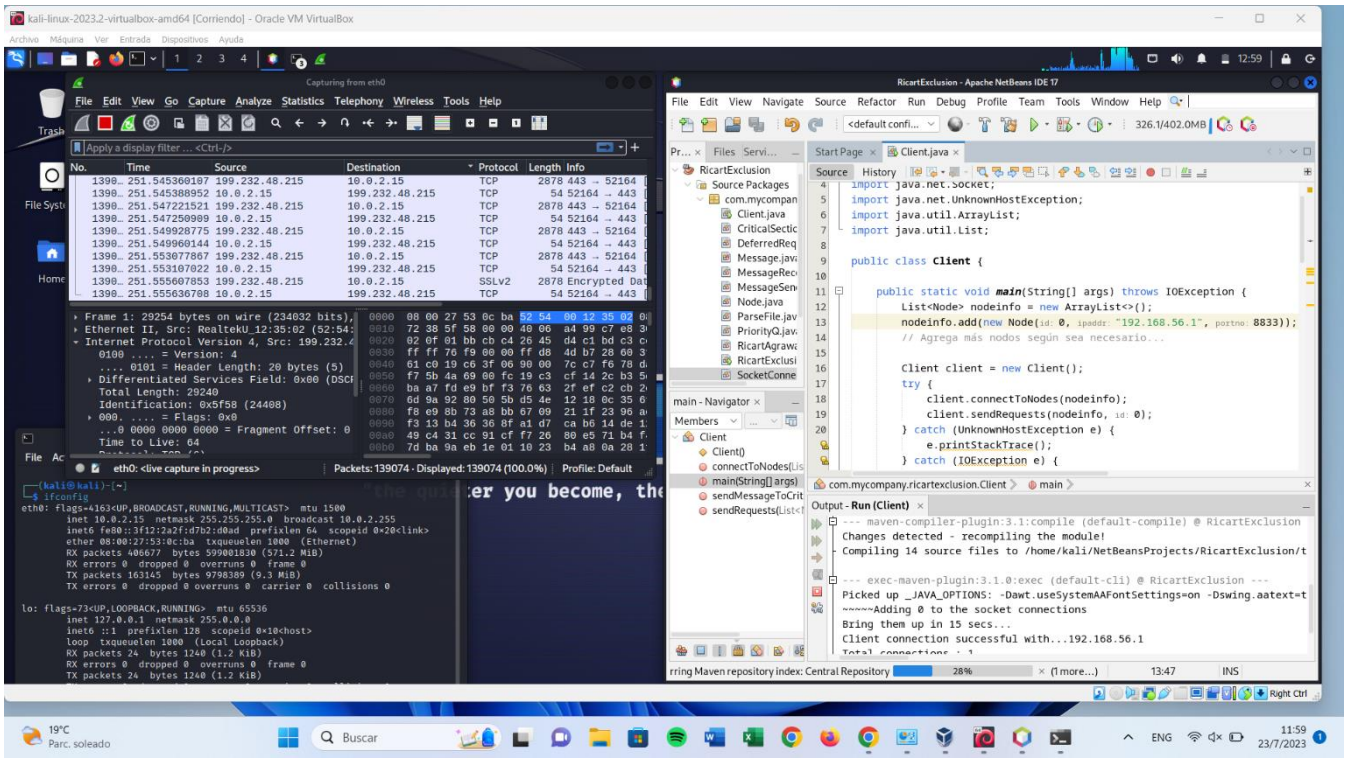
Captura de pantalla Fabricio Rafael Cantos Pérez.



Nota. El compañero muestra la conexión con el servidor desde la Máquina Virtual.

Imagen 8.

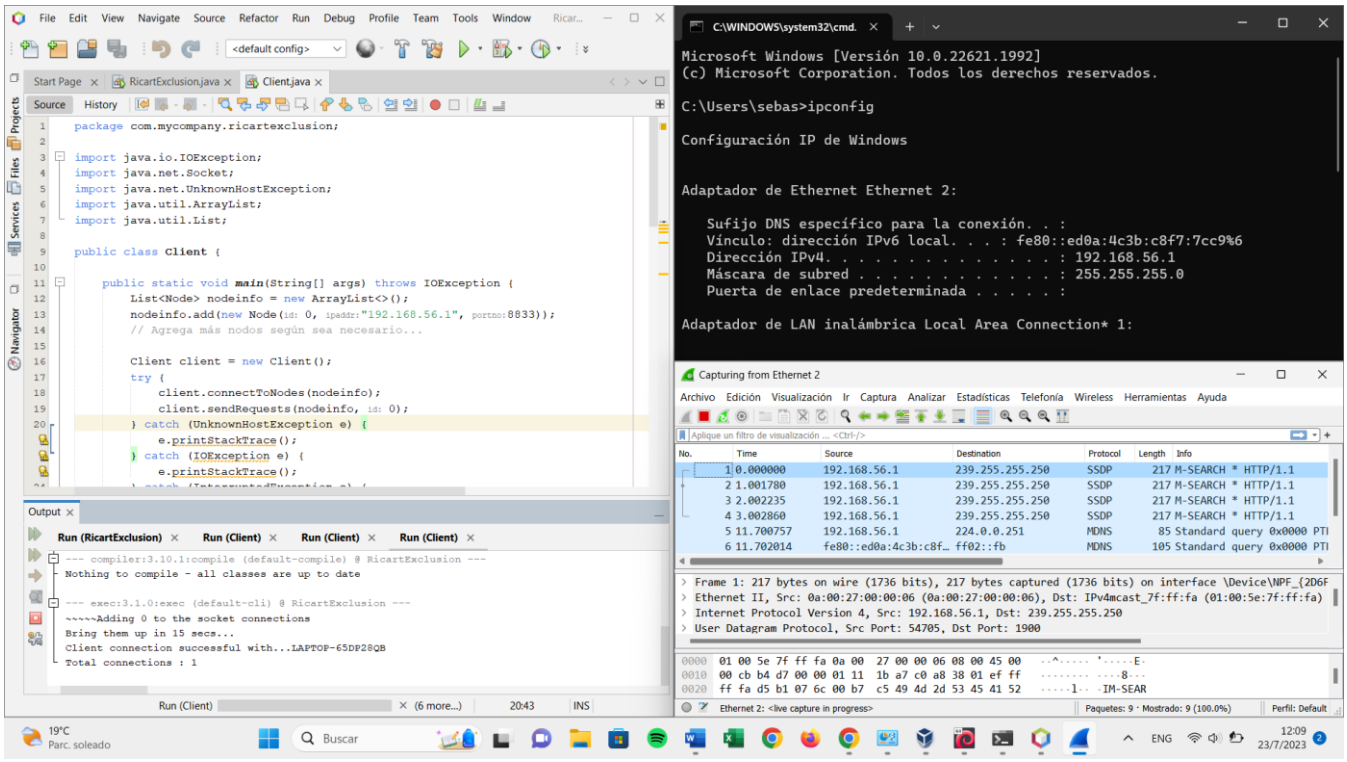
Captura de pantalla Cristian Sebastián Echeverria Vivanco.



Nota. El compañero muestra la conexión desde una maquina física a una máquina virtual.

Imagen 9.

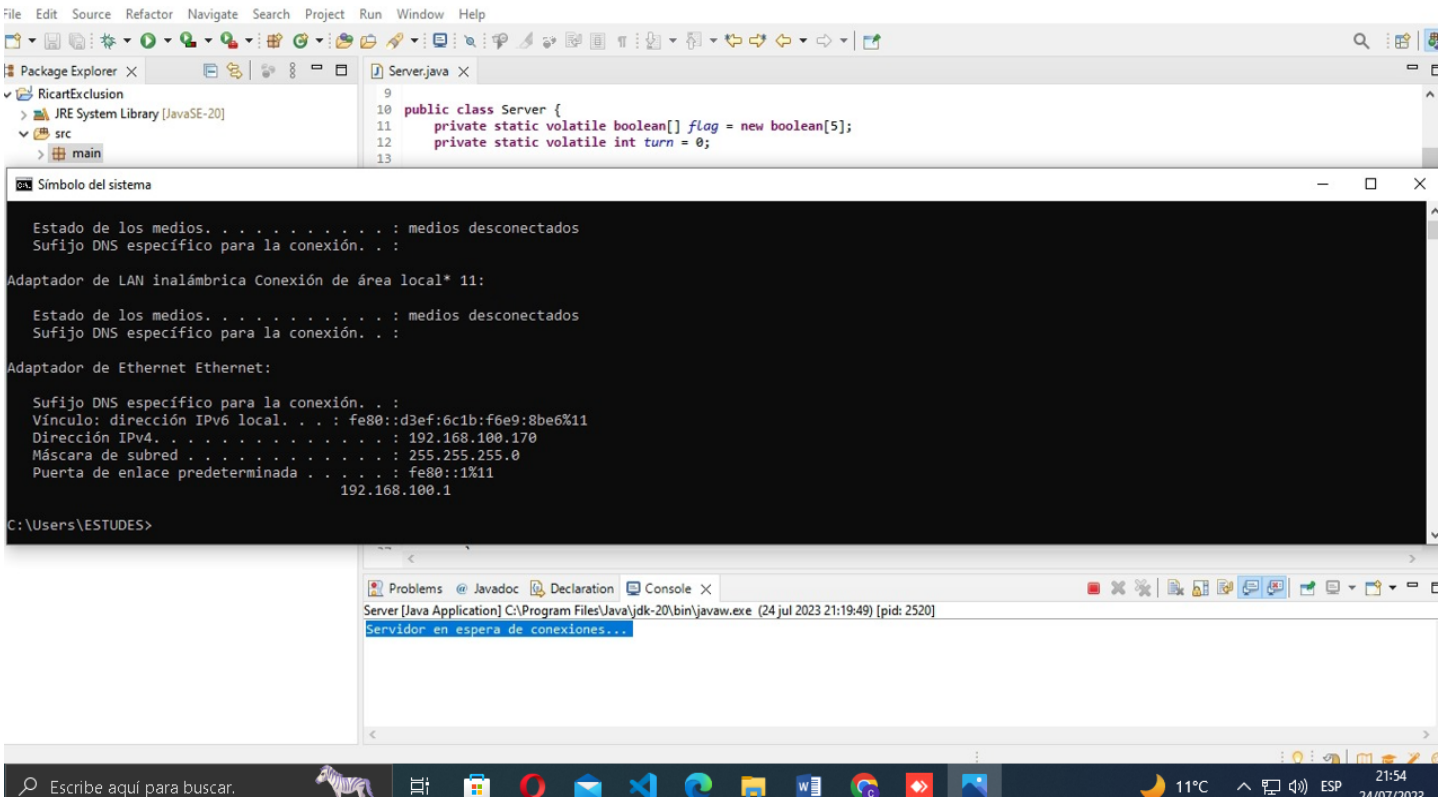
Captura de pantalla Cristian Sebastián Echeverria Vivanco.



Nota. El compañero muestra la conexión en una maquina física.

Imagen 10.

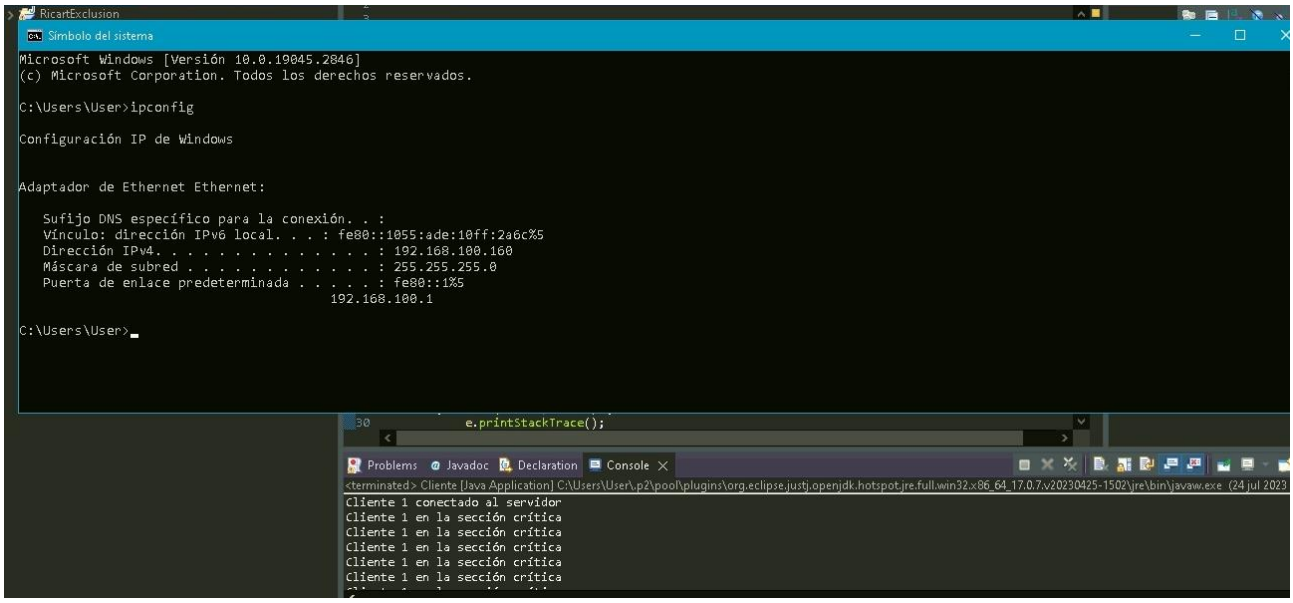
Captura de Carlos Humberto Guashpa Zumba



Nota. El compañero muestra la conexión desde una maquina física.el server conectado con la ip para que vean cual es.

Imagen11.

Captura de Carlos Humberto Guashpa Zumba

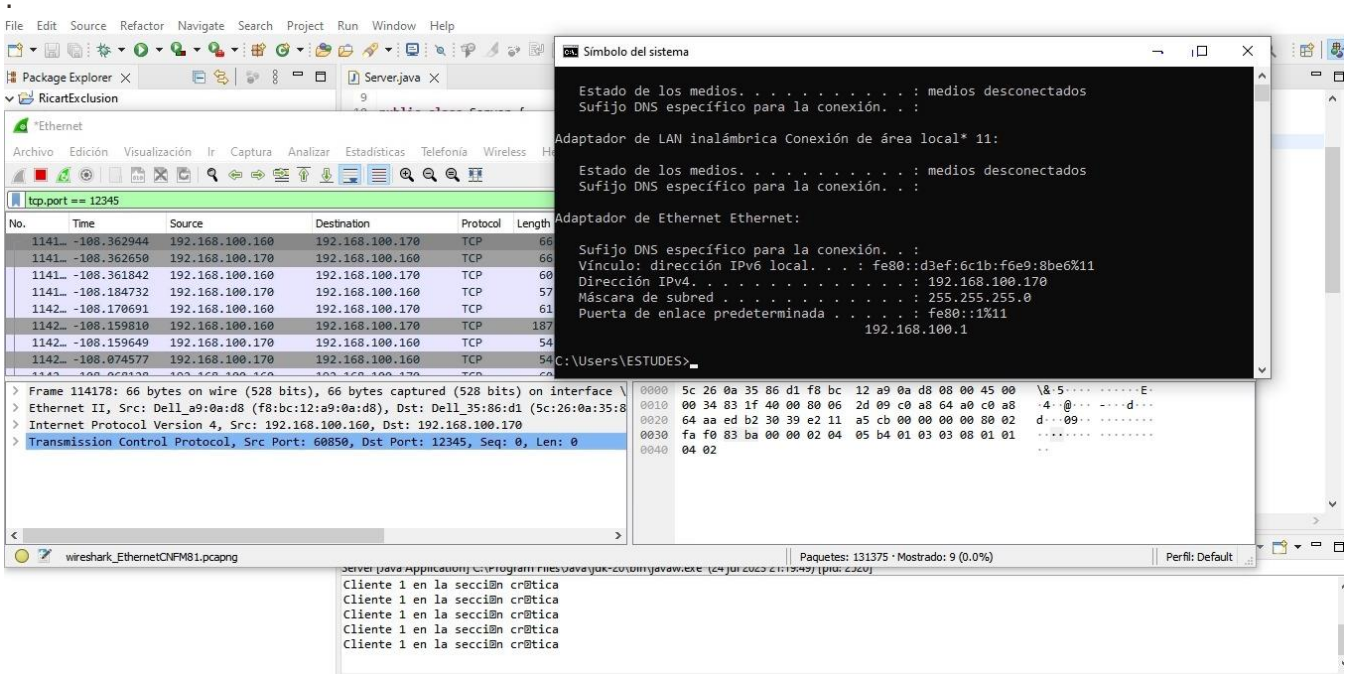


Nota. El compañero muestra la conexión desde otra máquina física. Cliente conectado al server con la ip del cliente.

Imagen 12.

Captura de Carlos Humberto Guashpa Zumba.

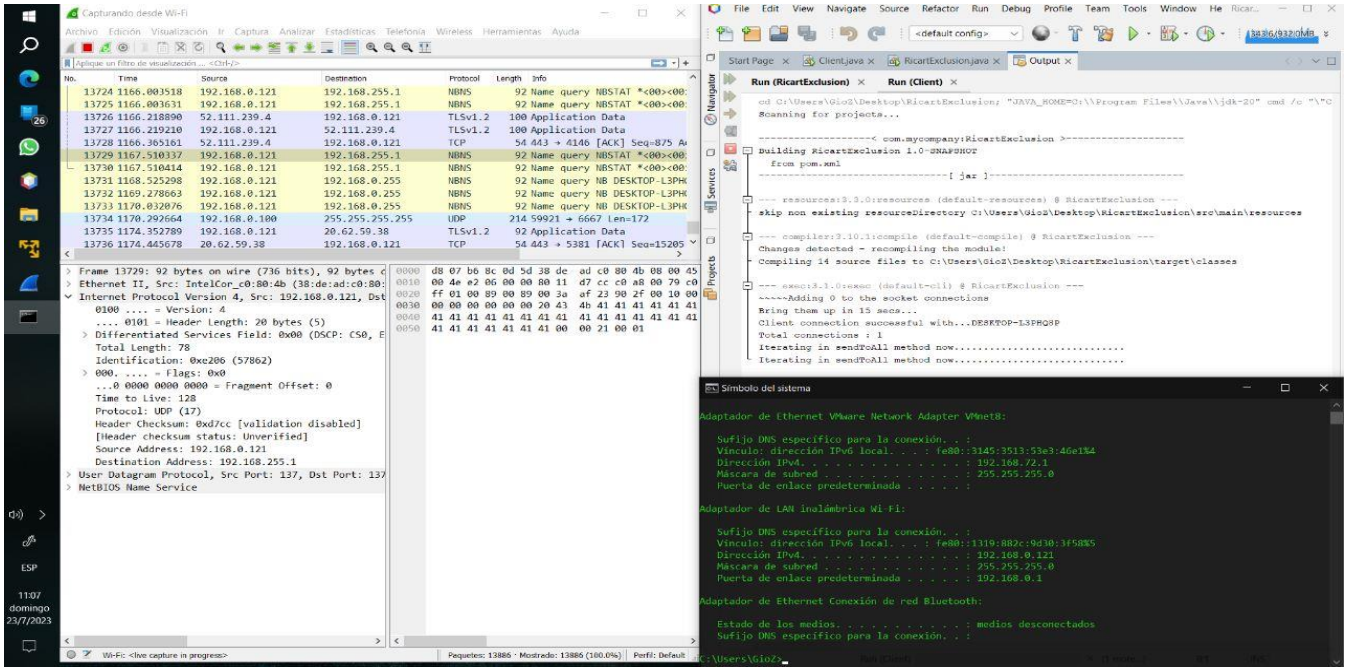
.



Nota. El compañero muestra la conexión desde otra máquina física Wireshark con la ip que estan conectado elcliente al server.

Imagen 13.

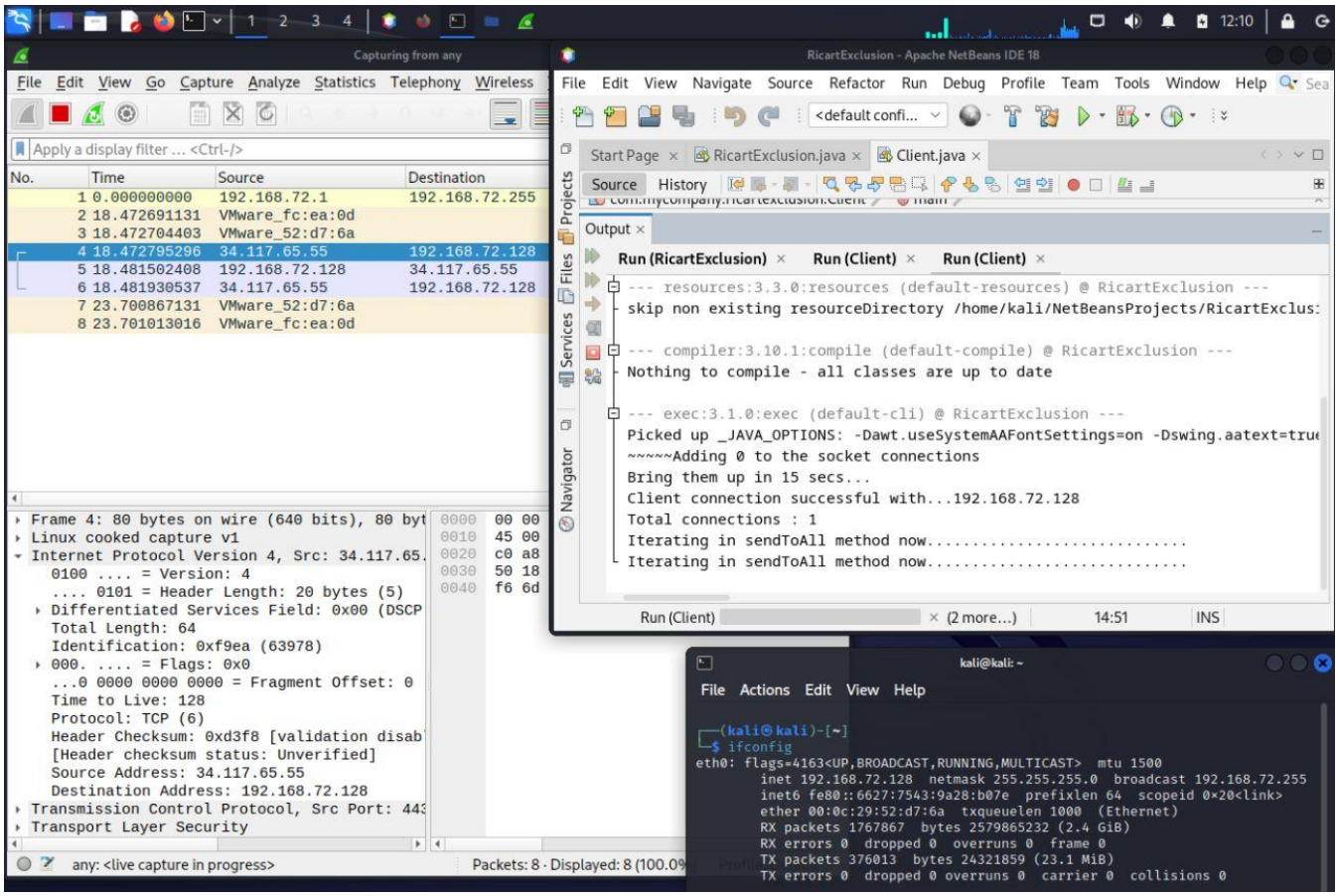
Captura de Giovanni Fernando Santana Tisalema



Nota. El compañero muestra conexión y el reloj global desde máquina física.

Imagen 14.

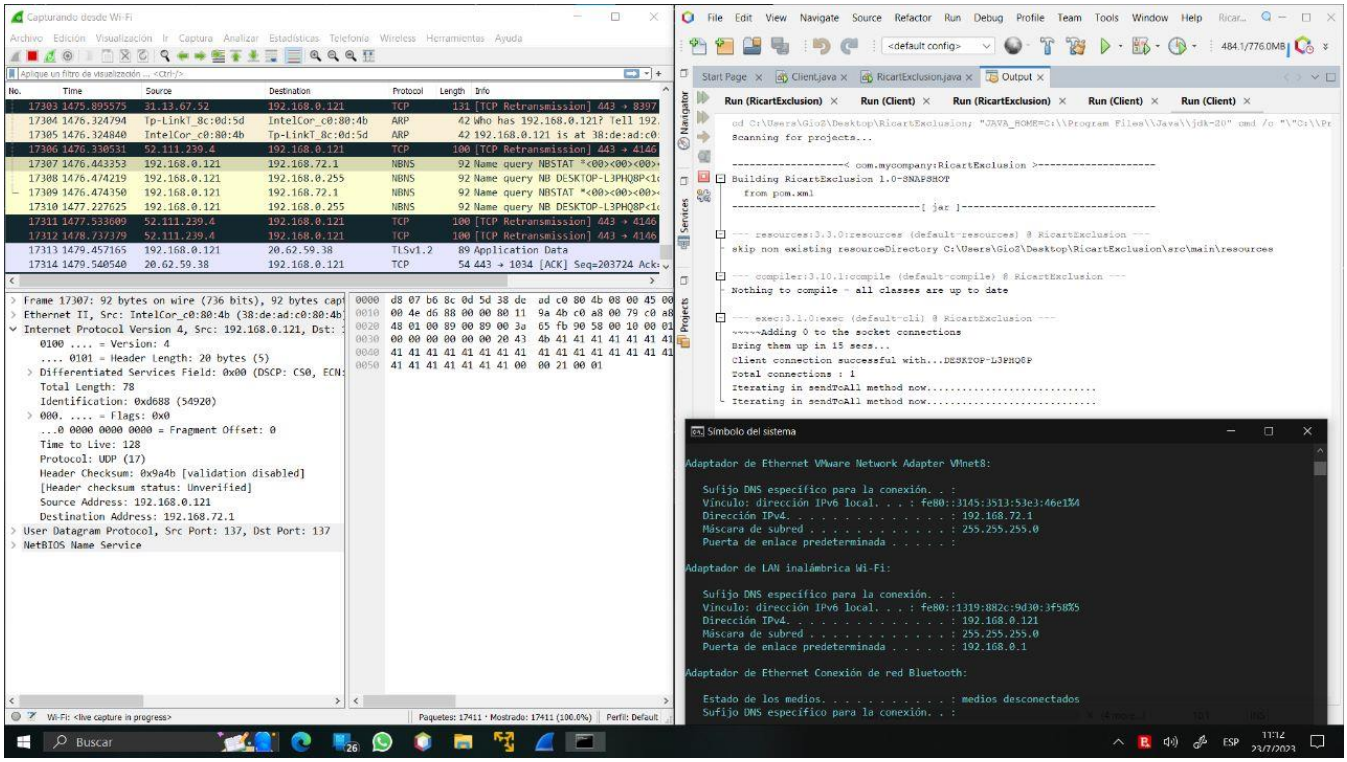
Captura de Giovanni Fernando Santana Tisalema



Nota. El compañero muestra conexión desde una máquina virtual.

Imagen 15.

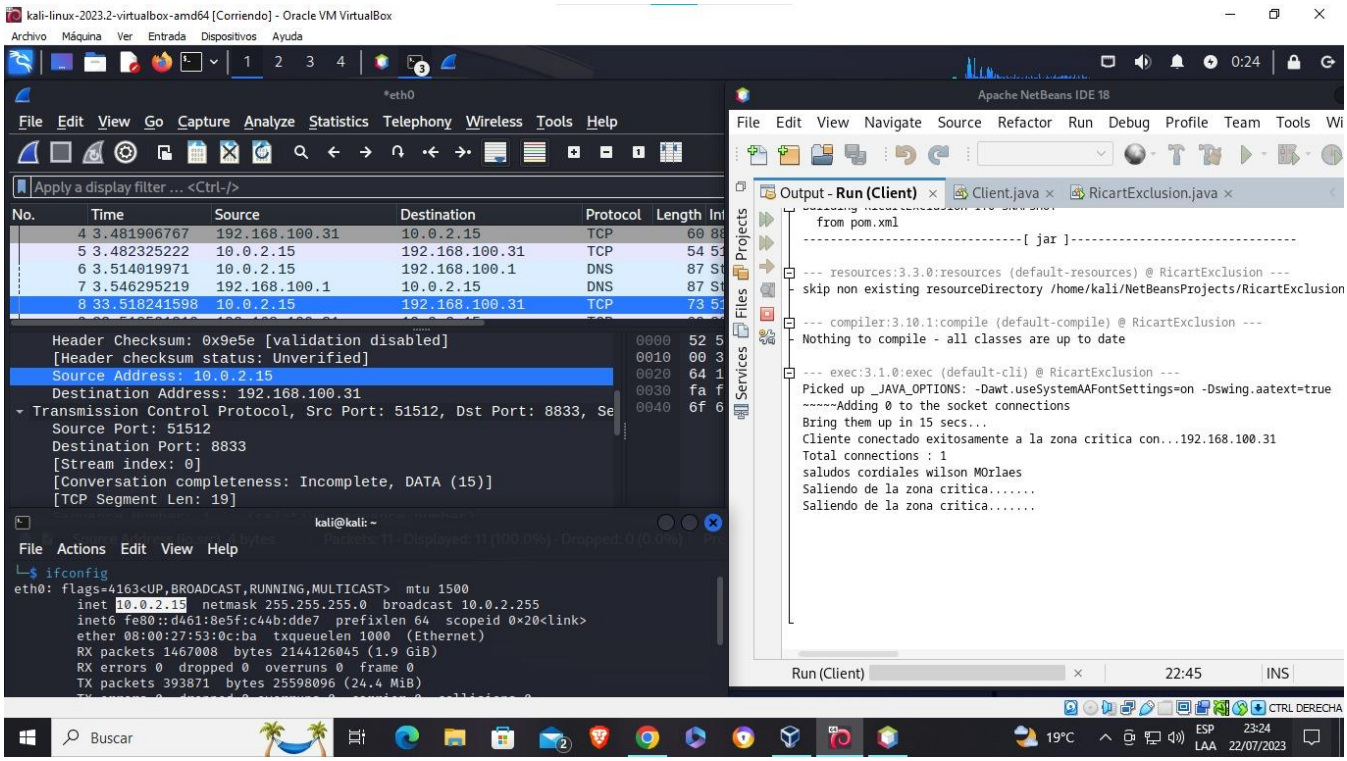
Captura de Giovanni Fernando Santana Tisalema



Nota. El compañero muestra la conexión y reloj global desde localhost en máquina física.

Imagen 16.

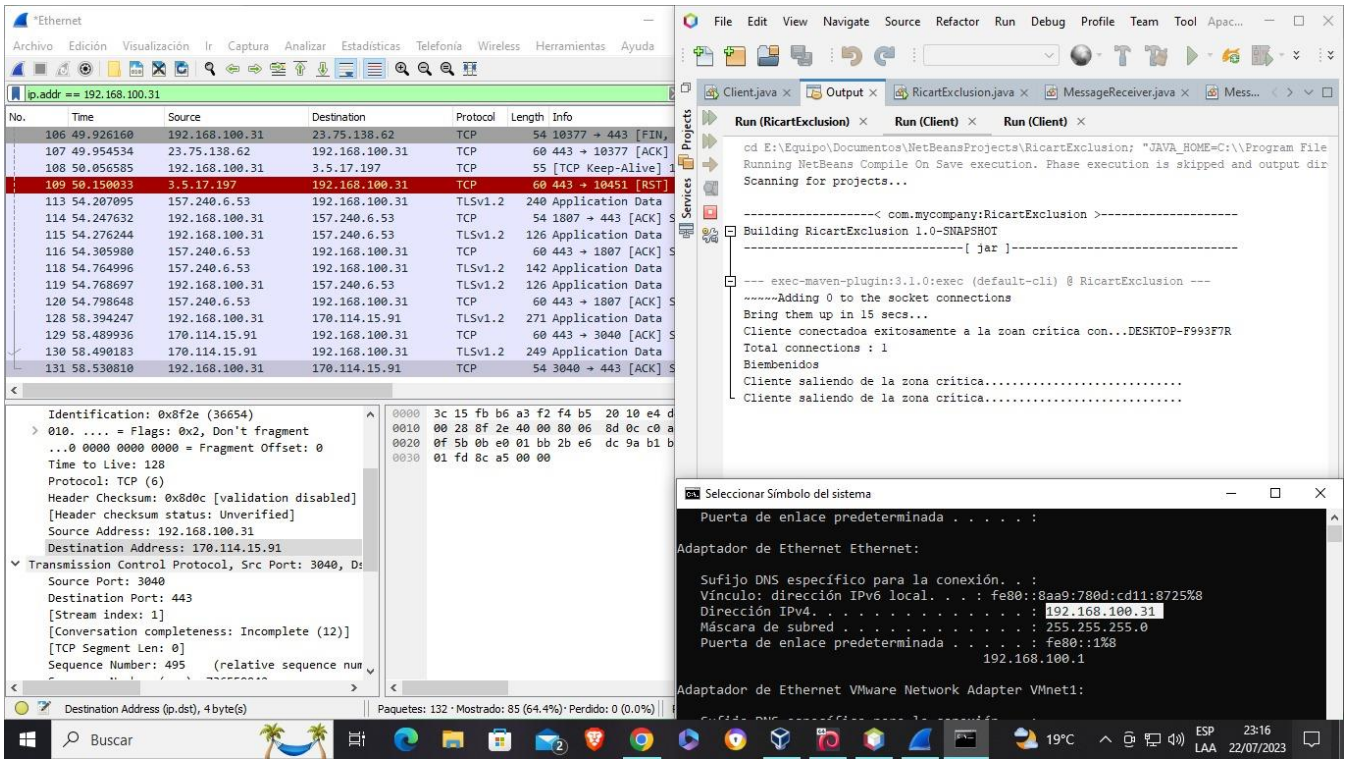
Captura de pantalla de Wilson Alexander Morales Picoita



Nota. El compañero muestra la conexión con el servidor desde máquina virtual.

Imagen 17.

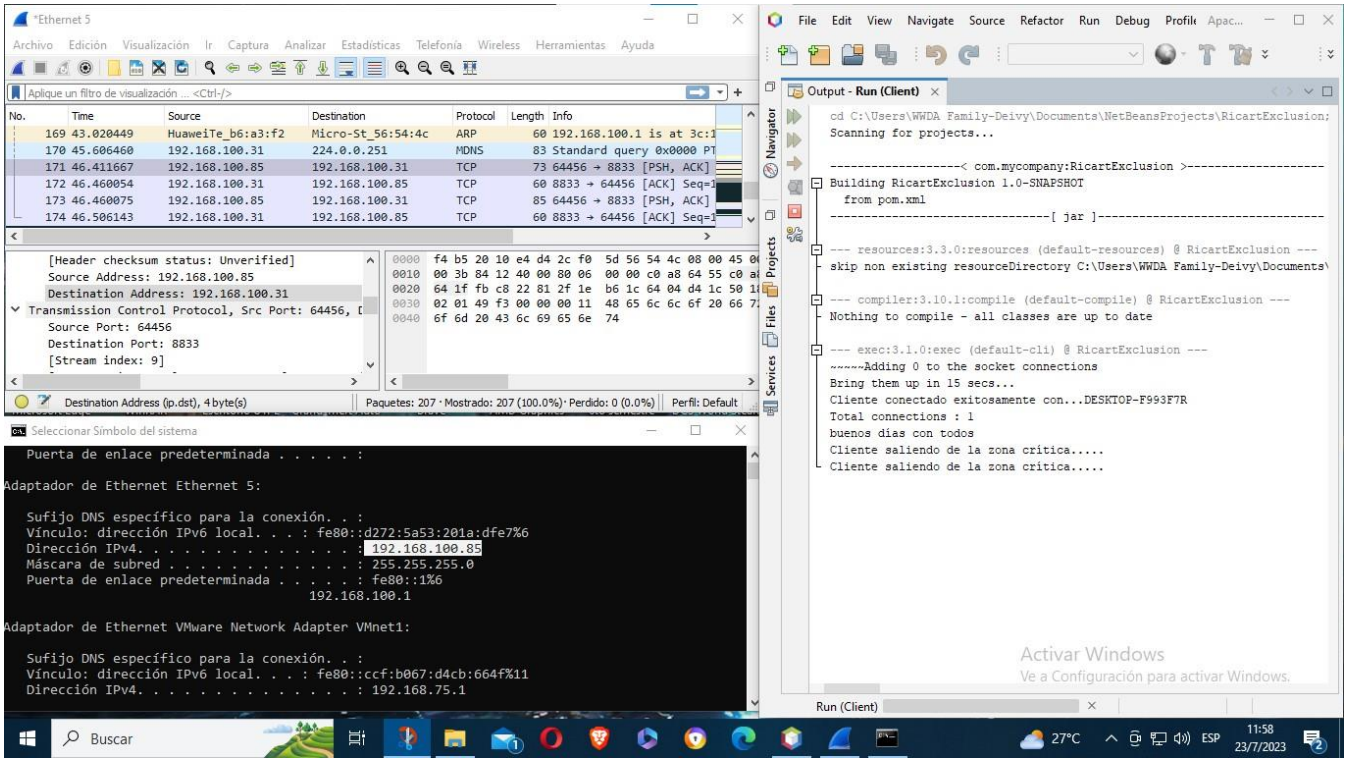
Captura de pantalla de Wilson Alexander Morales Picoita



Nota. El compañero muestra conexión en localhost desde máquina física.

Imagen 18.

Captura de pantalla de Wilson Alexander Morales Picoita



Nota. El compañero muestra conexión desde otra máquina físcia a la máquina física del servidor.

Conclusiones

la implementación y prueba del algoritmo de exclusión mutua de Ricart-Agrawala en un entorno local y en grupo ha sido una experiencia valiosa para entender los desafíos y beneficios de la programación concurrente y la coordinación de procesos en sistemas distribuidos. A través de esta actividad, hemos obtenido importantes aprendizajes y conclusiones:

- **Comunicación entre procesos**: La implementación de la comunicación entre procesos mediante sockets ha permitido comprender cómo los distintos nodos se comunican y coordinan para acceder a la sección crítica. Hemos comprendido cómo los mensajes de solicitud y respuesta son fundamentales para lograr una sincronización adecuada y evitar conflictos en el acceso a recursos compartidos.

- **Coordinación y concurrencia**: Trabajar en grupo para implementar el algoritmo nos ha brindado una visión más clara de cómo los procesos deben coordinarse y competir para acceder a la sección crítica. Hemos observado cómo la concurrencia puede dar lugar a situaciones de bloqueo o inanición si no se manejan adecuadamente.

- **Diseño y robustez**: Durante la implementación, nos hemos enfrentado a desafíos al diseñar el algoritmo y garantizar su correcto funcionamiento en un entorno local. La robustez del algoritmo se ha demostrado al evitar situaciones de interbloqueo y garantizar que un proceso no espere indefinidamente para acceder a la sección crítica.

- **Interacción y colaboración en grupo**: Trabajar en equipo ha sido fundamental para el éxito de la implementación. La comunicación efectiva, la coordinación y el reparto de tareas han sido aspectos esenciales para lograr una implementación exitosa del algoritmo. La colaboración nos ha permitido enfrentar desafíos de manera conjunta y encontrar soluciones más rápidamente.

- **Limitaciones del entorno local**: Aunque la prueba del algoritmo se realizó en un entorno local, hemos identificado que las limitaciones de concurrencia y el número de nodos involucrados pueden afectar la experiencia y los resultados. Sería interesante realizar pruebas en un entorno distribuido más amplio para evaluar completamente el rendimiento y la eficiencia del algoritmo.

En general, el trabajo en grupo y la implementación del algoritmo de exclusión mutua de Ricart-Agrawala nos ha permitido obtener una comprensión más profunda de los desafíos y complejidades de los sistemas distribuidos. A través de la coordinación y colaboración, hemos adquirido habilidades en programación concurrente y en el diseño de algoritmos que pueden ser aplicadas en contextos más amplios y complejos. Además, hemos comprendido la importancia de considerar las limitaciones del entorno y la concurrencia al diseñar sistemas distribuidos en la práctica. En general, ha sido una experiencia enriquecedora que ha fortalecido nuestros conocimientos y habilidades en el campo de la informática distribuida.

Bibliografía.

Chat.OpenIA. (2023). *Código realizado siguiendo sugerencias de ChatGPT.* Recuperado de: https://chat.openai.com/

Coulouris, G., Dollimore, J., & Kindberg, T. (2012). Sistemas Distribuidos: Conceptos y Diseño. Pearson Educación. ISBN: 978-84-7829-347-4

Tanenbaum, A. S. (2013). Sistemas Distribuidos. Alfaomega Grupo Editor. ISBN: 978-607-707-248-0

Madhav5589, (2015). *Ricart-Agrawala-algorithm-for-distributed-mutual-exclusion.* GitHub. Recuperado de: GitHub - madhav5589/Ricart-Agrawala-algorithm-for-distributed-mutual-exclusion

Wilson-Morales, (2023). *Algoritmo-Exclusion-Mutua*. GitHub. Recuperado de: https://github.com/Wilson-Morales/Algoritmo-Exclusion-Mutua