Programación de Servicios y Procesos

Daniel Esteban Castiblanco Guzmán
Jonatan Camilo Igua Contreras

Docente

Pedro Jose Rivera Osorio

Materia

Interconectividad

NRC

4445

Corporación Universitaria Minuto de Dios

Ingeniería de Sistemas

Bogotá D.C.

03/10/2023

1. Componentes que permiten la interconectividad del socket:

- **Multiplataforma:**

Java es un tipo de lenguaje orientado a objetos considerado multiplataforma ya que su código es bastante compatible con diferentes sistemas operativos, permitiendo solo programar una sola vez y ejecutar en cualquier parte.
El socket implementado fue construido con el lenguaje de java, por medio del IDE de NetBeans, este socket permite ser comprimido en un archivo JAR, permitiendo ser ejecutado en diferentes computadores, en el caso de este socket en específico funciona con tres o más computadoras, una que cumplira como servidor y las otras dos de clientes, este programa permite tener múltiples clientes siempre y cuando todos se encuentren en la misma red, ya que para realizar la conexión de los clientes con el servidor se usa el protocolo TCP/IP.

- **Modelo cliente servidor:**

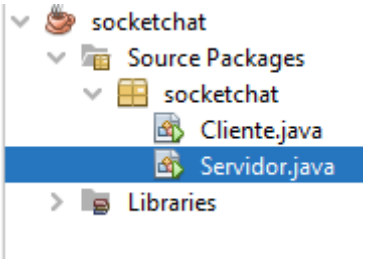


ilustración 1, Modelo cliente servidor

Se aplicó el modelo por medio de dos clases, una de Cliente y otra de Servidor, dentro de la clase cliente se usó un Socket, en la clase servidor se aplicó un ServerSocket, el código cumple con este modelo ya que la clase servidor ofrece un servicio por medio de la red LAN, a muchos clientes, este servidor escucha las peticiones de los clientes por medio de un puerto y los clientes se conectan al servidor para poder comunicarse entre sí.
Como usa una arquitectura centralizada el servidor es el pilar de la comunicación, si este se cae o falla toda la comunicación de los clientes se interrumpe (Faci, 2019).

- **Gestores de recursos de intercambio:**

En el estudio del socket seleccionado los gestores de recursos de intercambio son los flujos de entrada y salida de información, ya que estos flujo son necesarios para el intercambio de datos, para enviar y recibir los datos de la conexión de los clientes.
En este socket el flujo de salida se creó con ObjectOutputStream, usado para escribir los objetos de una clase serializada, esto permite enviar los datos en una especie de paquete "PaqueteEnvio" desde un cliente al servidor y del servidor al otro cliente.

Por parte del flujo de entrada se creó un ObjectInputStream, utilizado para leer los objetos serializados del flujo de entrada del socket , para leerlos y mostrarlos en el servidor.

- **Comunicación máquina a máquina: (De un PC a otro PC)**

Dentro de la comunicación para que el socket pudiera transmitir información de un PC a otro , se implementó gracias a la biblioteca de funciones de socket en Java la forma de plantear la utilización de la IP de cada equipo para poder comunicar un mensaje el cual se envía a través de un número de puerto por parte del servidor que interconecta los equipos para generar la conexión y que el programa permite generar esa transmisión, para poder realizar esta comunicación se necesita conocer las direcciones IP de los clientes y los puertos de escucha (Faci, 2019).

2. Manejo de socket con las capas del modelo OSI:

Uso del modelo Osi: El modelo OSI, define cómo se pueden comunicar los sistemas por medio de una red, como el envío de datos como texto o archivos de un remitente a un destinatario, este modelo establece los estándares o las reglas de los componentes de la arquitectura de las aplicaciones para realizar una comunicación adecuada por medio de la red (proofpoint, 2023).

Este modelo se usa en el programa del socket, ya que al realizar una comunicación entre diferentes máquinas por medio de una red física, por medio del TCP/IP y contando con una interfaz gráfica.

A Continuación se estudiarán las capas del modelo y su relación con el socket

Aplicación: Esta es la capa número 7, esta es la capa con las que los usuarios están más familiarizados, enfocándonos en el socket utilizado la aplicación de chat o la clase de cliente con la clase cliente componen la aplicación ya que estas clases contienen la lógica de envío y recepción de mensajes de texto.

Presentación: En la capa número 6 es la que se encarga de mostrar la información a los usuarios finales, al referirnos al socket de chat, esta capa se podrían encargar de la codificación de los datos, que son enviados entre los clientes, como el nick, ip, mensaje, en este programa se realiza por medio de una clase serializada, esta capa de encargaría de serializar y deserializar los objetos, como el objeto 'PaqueteEnvio' implementado en el programa.

Sesión: La capa número 5 se encarga de la comunicación entre dos o más dispositivos, debe crear un puente o una sesión por donde los dispositivos se puedan comunicar entre sí, en el contexto del socket utilizado, la gestión de conexiones y de desconexión es una función de esta capa en el programa.

Transporte: Esta es la capa número 4, se encarga de tomar los datos que se desean envían y los divide en partes más pequeñas, estos segmentos contienen la información de encabezado para que al viajar por la red el dispositivo receptor pueda volver ensamblar las partes y generar el paquete de datos para ser leído. El socket en la capa de transporte usa el protocolo

TCP (Protocolo de control de Transmisión), este protocolo permite generar una comunicación fiable entre el cliente - servidor de la aplicación.

Red: La capa número 3 es la que encarga de dividir los datos enviados en paquetes con toda la información del mensaje dentro, esto permite enviar la información por medio de una red, si es una red local esta capa no es de mucha ayuda porque como no es necesario enrutar el mensaje esta capa no tiene que redirigir los paquetes a sus diferentes direcciones de destino, en el caso del estudio del Socket, se utiliza una red pero es una red local, ya sea por medio de equipos físicos o por medio de la virtualización.

Enlace de Datos: La capa número 2 permite la comunicación en diferentes redes, esta es la responsable de la transmisión por la misma red, permitiendo que los datos tengan un enlace confiable de comunicación, este enlace puede ser por medio de un cable Ethernet o por medio de un medio inalámbrico, volviendo al enfoque del socket este ya maneja de manera automática esta capa, gracias a la infraestructura física que utiliza el socket y los protocolos de comunicación como el TCP /IP.

Física: La primera capa del modelo osi es la más inferior de todas, esta se encarga de los elementos físicos utilizados para la transferencia de datos, como los cables, los conectores o los routers, el socket construido utiliza los elementos físicos del dispositivo en donde se esté ejecutando el software, como el cable de Ethernet, si usamos la aplicación en un entorno virtualizado usamos los recursos virtuales de los sistemas operativos creados y los recursos de la máquina principal que emula el entorno virtualizado.

3. Ejecución del socket:

Lenguaje utilizado en el socket: Para la creación del socket se usó como lenguaje de programación Java.

Sintaxis de Java utilizada para socket:

Para la creación del socket se creó un paquete llamado ‘socketchat’ dentro de este paquete hay creadas dos clases una llamada Cliente.java y Servidor.java, permitiendo simular un modelo cliente servidor.

A Continuación se explicarán las partes más importantes del código del socket y su funcionamiento

Clase Servidor

```java
/*Esta primera linea de codigo declara una clase llamada servidor,
con la palabra public esto significa que la clase servidor es accesible
y visible desde cualquier parte del codigo, esta clase 'Servidor', es la clase
principal*/
public class Servidor {
    /*Se encuentra el metodo main, es el metodo principal de java, es el metodo
    que siempre se ejecuta al iniciar cualquier programa, los parametros String[] args
    permiten pasar argumentos desde la linea de comandos, pero como se esta usando un IDE,
    no se estan utilizando*/
    public static void main(String[] args) {
        /*Se crea una instancia de la clase 'ServidorCliente', llamada 'unServidor'
        esta instacia se utiliza para iniciar el servidor de este programa, cargando
        la interfaz grafica del servidor*/
        ServidorCliente unServidor = new  ServidorCliente();
        /*Permite asignar la tarea o función de que cuando se cierre el Frame o la
        ventana de la clase servidor el programa se cancele o termine su
        ejecución, esta opcion se logra con JFrame.EXIT_ON_CLOSE*/
        unServidor.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
    }
}
```

```java
/*Se crea la clase 'ServidorCliente', esta clase es la encargada de la interfaz
del servidor ya que hereda o extends de la clase 'JFrame', clase utilizada para
crear una ventana, tambien se implementa la interfaz 'Runnable', esto para poder
ejecutar un hilo, este hilo se usa para escuhar las peticiones que envian los
clientes por medio del socket*/
class ServidorCliente extends JFrame implements Runnable{
    /*Se declara una variable llamada 'areaTexto' de tipo JTextArea, esta variable
    se usa para poder mostrar los mensajes o la información que es enviada por
    el cliente en la interfaz grafica del servidor*/
    private JTextArea areatexto;
    /*Se crea el constructor de la clase 'Servidor', en este constructor, se
    configura la interfaz grafica y se crea un hilo para las conexiones con los
    clientes*/
    public ServidorCliente(){
        /*Se definen las dimensiones y la posición de la ventana del servidor
        * X: La cordenada 'x' de la esquina superior izquierda de la ventana
        * Y: La coordenada 'y' de la esquina superior  izquierda de la ventana
        * width: Ancho de la ventana
        * heignt: Altura de la ventana*/
        setBounds(x: 600,y: 300,width: 280,height: 350);
        /*Se crea un objeto de 'JPanel' llamado 'milamina', este objeto se usa
        como panel de la interfaz, adentro de este panel estaran los demas elementos
        como los botones*/
        JPanel milamina= new JPanel();
        /*Se llama al metodo 'setLayout', con el objeto 'milamina', para  utilizar
        un el diseño con 'BorderLayout' para organizar los elementos por regiones
        como sur, este, oeste o centro*/
        milamina.setLayout(new BorderLayout());
        /*Se crea un objeto JTexArea llamado 'areatexto', es una caja de texto
        se usa para mostrar la informacion que los clientes envian*/
        areatexto=new JTextArea();
        /*Se usa el metodo 'add', del objeto 'milamina', para añadirle un componente
        a este panel, en este case se paso el componente 'areatexto', al cual por medio
        de 'BorderLayout.CENTER', se posiciono el componente en el centro del panel*/
        milamina.add(comp: areatexto,constraints: BorderLayout.CENTER);
        /*Se crea un objetode 'JLabel', llamado 'jLabelText', el cual se usa para
        mostrar un mensaje en la interfaz del servidor*/
        JLabel jLabelText = new JLabel(text: "Esperando mensaje del cliente...");
        /*Con el metodo add del objeto milamina se agrega el jLabelText, en la
        parte sur del panel*/
        milamina.add(comp: jLabelText, constraints: BorderLayout.SOUTH);
        /*Se usa el metodo add del JFrame que representa la instacia de la clase
        'ServidorCliente', para añadir el JPanel al Frame*/
        add(comp: milamina);
        /*Se usa el metodo setVisible, sobre el Frame para poder configurar la
        visiblidad de la ventana, como esta true, la ventana sera visible en la
        interfaz grafica*/
        setVisible(b: true);
        /*Se crea un objeto Thread llamado 'mihilo', para poder ejecutar un hilo
        en segundo plano, se pasa como argumento 'this', como argumento al constructor
        del hilo para poder indicar que este hilo se va a ejecutar el cosigo de la
        clase actual*/
        Thread mihilo = new Thread(target: this);
        /*Se utiliza el metodo 'start', para iniciar la ejecucion del hilo en segundo
        plano*/
        mihilo.start();
    }
```

En la clase servidor, primero nos encontraremos la clase principal llamada ‘Servidor’, dentro de esta clase encontramos el método principal main, el cual tiene una instancia creada de la clase ‘ServidorCliente’ con un objeto llamado unServidor.

Más adelante se crea la clase ‘ServidorCliente’ que extends o hereda de la clase JFrame para poder crear la interfaz gráfica de la aplicación, por otro lado se implementa Runnable porque se implementa un hilo que se ejecutará en segundo plano.
Se crea un constructor de la clase con todos los elementos para la interfaz gráfica y además se declara un hilo (Thread) llamado mihilo.

```
/*El @override indica que se esta implementando el metodo run de la interfaz
Runnable, */
  @Override
  /*El metodo run define las acciones que debe ejecutar el hilo en segundo plano
  se usa como public void, porque necesita ser accedido desde cualquier parte del
  programa, se usa void ya que este metodo no devuelve algun valor cuando se
  ejecuta, solo se usa para realizar acciones en segundo plano pero no devuelve
  algun resultado especifico*/
    public void run() {
        /*Esta esctructura el try- catch, se usa para manejar excepciones, para
        validar errores o situaciones que sucedan durante el programa, evitando
        que el programa se cierre sin alguna notificación de error*/
        try {
        //Codigo estar en la escucha hilo
        /*Se crea un objeto 'ServerSocket', utilizado para abrir un puerto
        de escucha en el servidor, se crea 'unServerSocket', una instancia del
        'ServerSocket', en donde se especifica el puerto 536 que sera el puerto
        de escucha del servidor*/
        ServerSocket unServidor = new ServerSocket(port: 536);//puerto de escucha abierto
        /*Se crean variables de tipo texto para almacenar la informacion
         del paquete del cliente que fue enviado por el socket del cliente*/
        String nick, ip, mensaje;
        /*Se crea una variable llamada 'paquete_recibido' que es de tipo 'PaqueteEnvio'
        se usa para almacenar los objetos de tipo 'PaqueteEnvio', que son enviados
        por la red.
        PaqueteEnvio es una clase de la clase Cliente, se usa para representar
        el paquete de datos que es enviado por el cliente al servidor, este paquete
        contiene la informacion de nick, ip, mensaje*/
        PaqueteEnvio paquete_recibido;
        //Se crea bucle infinito para escuchar siempre mensajes, por medio un hilo en segundo plano
        while(true){//Ciclo para escuchar siempre peticiones del cliente
            //aceptar conexion que venga para ese puerto
            /*Se acepta una conexion de un cliente en el servidor, el metodo
            accept() se usa para esperar y aceptar las conexiones de clientes*/
            Socket misocket = unServidor.accept();
            /*Se usa para tener una notificacion en consola que se acepto la
            conexion del cliente*/
            System.out.println(x: "Si entro a mensaje");
            //Flujo de entrada para recoger lo que envio el flujo paquete el cliente
            //(Flujo de mensaje)
            /*Se crea un flujo de entrada con 'ObjectInputStream' usado para leer
            objetos desde la conexion 'misocket', la conexion del cliente, este flujo
            de entrada permite leer objetos serializados, del flujo se crea una
            variable llamada 'paquete_datos', para poder interacturar con el
            flujo de entrada*/
```

```
                ObjectInputStream paquete_datos = new ObjectInputStream(in: misocket.getInputStream());
                //Guardar en el objeto lo que viene en la red
                /*La variable 'paquete_recibido' anteriormente creada, almacena
                la lectura del objeto 'paquete_datos', obtenido de la red, con el
                metodo readObject().
                'paquete_recibido' es un objeto de la clase 'PaqueteEnvio', clase
                del cliente, que contine la informacion enviada por el cliente, como
                nombre de usuario, ip y el mensaje*/
                paquete_recibido= (PaqueteEnvio) paquete_datos.readObject();//leer el flujo de datos
                //Obtener del paquete los datos
                /*Se recupera el valor de el nombre de usuario 'nick' del objeto
                'paquete_recibido', se usa el metodo getNick() de la clase 'PaqueteEnvio'
                para obtener el valor del nombre y guardarlo en la variable nick de la
                clase servidor
                Este proceso se realiza con los demas datos*/
                nick = paquete_recibido.getNick();
                ip = paquete_recibido.getIp();
                mensaje = paquete_recibido.getMensaje();
                /*Se agrega al componente 'areatexto' con el metodo append la información
                obtenida del paquete enviado por el cliente y leido por el flujo de entrada,
                para ser mostrada en el servidor*/
                areatexto.append("\n" + nick + ": " + mensaje + " para " + ip);

                //<<Codigo para reenviar informacion del servidor al otro cliente>>
                /*Se crea una nueva conexion de socket llamada 'enviaDestinatario'
                a la cual se le pasa por parametro la variable ip, obtenida del objeto del
                cliente, la cual tiene la IP  del cliente al que se le envia el
                mensaje, ademas un puerto de escucha que seria el puerto devuelta el
                cual debe estar vinculado a un socket de servidor para que reciba
                el objeto del servidor principal*/
                Socket enviaDestinatario = new  Socket(host: ip,port: 540);//puerto devuelta
                //Se crea un flujo de salida
                ObjectOutputStream paqueteReenvio = new ObjectOutputStream(out:enviaDestinatario.getOutputStream());
                /*se usa el flujo de salida paqueteReenvio, para escribir el objeto
                de 'paquete_recibido', enviado por el cliente con el metodo writeObject,
                con el fin de enviar el 'paqueteReenvio', con la informacion del mensaje al
                cliente receptor*/
                paqueteReenvio.writeObject(obj:paquete_recibido);
                //cerrar flujo datos
                paqueteReenvio.close();
                enviaDestinatario.close();
                //cerrar socket,cerramos la conexion
                misocket.close();
            }
            } catch (IOException ex) {
                Logger.getLogger(name: ServidorCliente.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(name: ServidorCliente.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
    }
}
```

Continuando con la clase de servidor, se aplicó un '@override public void run ' que será el método que utiliza el hilo para ejecutar el código en segundo plano que se encuentre dentro de este método,

Creamos una instancia de la clase 'ServerSocket', que permite que la aplicación pueda establecer una conexión a un equipo en donde esté estipulado el puerto que escuche las peticiones del cliente.

Se usan las variables nick, ip, mensaje de tipo String para guardar la información de nombre, ip y mensaje que envío el cliente al servidor, se crea una instancia de la clase de cliente 'PaqueteEnvio',

como es necesario que el servidor desde su inicio esté constantemente escuchando peticiones del cliente.

Dentro de este while infinito está la lógica de entrada y salida de datos, se crea un Socket llamado 'misocket', para tomarlo de referencia al socket del cliente y por medio del método accept( ) del ServerSocket, aceptamos la conexión o la petición del socket del cliente que

envía información al puerto del ServerSocket, una vez que se acepta la conexión se puede comenzar a crear los flujos de comunicación.
En este socket se crea los flujos de entrada con ObjectInputStream, usado para poder leer el objeto ‘paquete_recibido’ enviado por el cliente, con ayuda del objeto del flujo de entrada con el método readObject( ), se leen los datos del objetos del cliente, con ayuda de las variables anteriormente creadas se almacenan los gets de los datos del objeto recibido para al final mostrar la información en el campo del servidor.
Luego de recibir la información del cliente leer y imprimir la información, se procede a reenviar esta información al cliente de destino, se crea un Socket al cual se le pasa la información de la variable ip, recuperada anteriormente del objeto recibido, después le pasamos el numero del puerto del ‘ServerSocket’, imple,implementado en la clase cliente para recibir la información, se crea un el flujo de salida con un ObjectOutputStream, en donde con el método writeObject le pasamos el ‘paquete_recibido’, que fue el objeto recibido del cliente, para proceder a reenviarlo al cliente de destino por medio de la dirección Ip, que se encuentra en este paquete, al final se cierran los flujos utilizados y el socket.

Clase Cliente

```java
/*Esta primera linea de codigo declara una clase llamada Cliente,
con la palabra public esto significa que la clase Cliente es accesible
y visible desde cualquier parte del codigo, esta clase 'Cliente', es la clase
principal*/
public class Cliente {
    /*Se encuentra el metodo main, es el metodo principal de java, es el metodo
    que siempre se ejecuta al iniciar cualquier programa, los parametros String[] args
    permiten pasar argumentos desde la linea de comandos, pero como se esta usando un IDE,
    no se estan utilizando*/
    public static void main(String[] args) {
        /*Se crea una instancia de la clase 'VistaCliente', llamada 'unCliente'
        esta instacia se utiliza para iniciar el cliente de este programa, cargando
        la interfaz grafica del cliente*/
        VistaCliente unCliente = new VistaCliente();
        /*Permite asignar la tarea o función de que cuando se cierre el Frame o la
         ventana de la clase cliente el programa se cancele o termine su
         ejecución, esta opcion se logra con JFrame.EXIT_ON_CLOSE*/
        unCliente.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
    }
}
/*Se crea la clase 'VistaCliente', esta clase es la encargada de la interfaz
del servidor ya que hereda o extends de la clase 'JFrame', clase utilizada para
crear una ventana de cliente*/
class VistaCliente extends JFrame{
    /*Se crea el constructor de la clase 'VistaCliente', en este constructor, se
    configura la interfaz grafica del cliente*/
    public VistaCliente(){
        /*Se definen las dimensiones y la posición de la ventana o frame  del cliente
        * X: La cordenada 'x' de la esquina superior izquierda de la ventana
        * Y: La coordenada 'y' de la esquina superior  izquierda de la ventana
        * width: Ancho de la ventana
        * heignt: Altura de la ventana*/
        setBounds(x: 600,y: 300,width: 280,height: 350);
        /*Se crea una instancia de la clase 'LaminaVistaCliente' con el nombre de
        'milamina', para poder interacturar con los elementos de la clase 'LaminaVistaCliente'*/
        LaminaVistaCliente milamina = new LaminaVistaCliente();
        /*Se usa el metodo add del JFrame , para añadir el el objeto milamina al Frame*/
        add(comp: milamina);
        /*Se usa el metodo setVisible, sobre el Frame para poder configurar la
        visiblidad de la ventana, como esta true, la ventana sera visible en la
        interfaz grafica*/
        setVisible(b: true);
    }
}
```

```java
/*Se crea la clase 'LaminaVistaCliente', esta clase es la encargada de los
componentes de la interfaz del cliente ya que hereda o extends de la clase 'JPanel',
clase utilizada para crear un panel dentro de un frame en donde colocar los componentes
visuales, tambien se implementa la interfaz 'Runnable', esto para poder
ejecutar un hilo, este hilo se usa para escuchar los objetos reenviados de la clase
servidor y mostrarlos en la clase cliente, permite ver los mensajes recibidos de un
cliente*/
class LaminaVistaCliente extends JPanel implements Runnable{//se crean todos elementos par
    /*Se crea el constructor de la clase 'LaminaVistaCliente', en este constructor, se
    configura los componentes de al interfaz grafica y se crea un hilo para la conexion co
    clase servidor para recibir el paquete reenviado*/
    public LaminaVistaCliente(){
        /*Se crea una instancia de JTextField, llamada nick con un ancho de 5 columnas
        permite ingresar texto en la interfaz, este campo es para que el cliente ingrese
        su nombre de usuario*/
        nick = new JTextField(columns:5);
        /*Con el metodo add añade este campo al JPanel*/
        add(comp:nick);
        /*Se crea una etiqueta JLabel llamada texto que muestra el texto -CHAT-
        en la interfaz*/
        JLabel texto=new JLabel(text:"-CHAT-");
        add(comp:texto);//Con el metodo add añade este campo al JPanel
        /*Crea una instancia de JTextField, llamada ip con un ancho de 8 columnas
        permite ingresar texto en la interfaz, este campo es para que el cliente ingrese
        la IP de destino o del dispositivo que desea enviar el mensaje*/
        ip = new JTextField(columns:8);
        add(comp:ip);//Con el metodo add añade este campo al JPanel
        /*Se crea componente de area de texto, con los parametros de numeros de filas
        y columnas que indica el area inicial del area de texto, en este campo se
        podran observar los mensajes enviados del otro cliente*/
        campochat = new JTextArea(rows:12,columns:20);//cordenada campo
        add(comp:campochat);//Con el metodo add añade este campo al JPanel
        /*Crea una instancia de JTextField, llamada campo1 con un ancho de 20 columnas
        permite ingresar texto en la interfaz, este campo es para que el cliente ingrese
        el mensaje a enviar por la red*/
        campo1 = new JTextField(columns:20);//campo escritura
        add(comp:campo1);//Con el metodo add añade este campo al JPanel
        /*Se crea un boton de JButton llamado 'miBoton', con el texto 'Enviar', usado
        para que el cliente envie los mesajes por la red*/
        miBoton = new JButton(text:"Enviar");
        /*Se crea una instancia de la clase EnviaTexto, para poder asociar una
        accion o evento al boton miBoton que se creo anteriormente*/
        EnviaTexto mievento = new EnviaTexto();
        /*Se agrega una accion de escucha al boton, para que cuendo el boton sea clicado,
        se ejecute el codigo de la clase EnviaTexto*/
        miBoton.addActionListener( l:mievento);//boton a la escucha
```

```
        add(comp: miBoton);//Con el metodo add añade este campo al JPanel

        /*Este programa incorpora el protocolo FTP aparte del socket*/
        /*Se crea un boton de JButton llamado 'consultarArchivo', con el texto
        'Ver archivos servidor FTP', usado para que el cliente pueda consultar
        los archivos del servidor FTP local un servidor aparte del servidor del socket*/
        consultarArchivo = new JButton(text: "Ver archivos servidor FTP");
         /*Se crea una instancia de la clase FTP, para poder asociar una
        accion o evento al boton consultarArchivo que se creo anteriormente*/
        FTP unftp = new FTP();
        consultarArchivo.addActionListener( l: unftp);
        add(comp: consultarArchivo);//Con el metodo add añade este campo al JPanel

        /*Se crea un objeto Thread llamado 'mihilo', para poder ejecutar un hilo
        en segundo plano, se pasa como argumento 'this', como argumento al constructor
        del hilo para poder indicar que este hilo se va a ejecutar el cosigo de la
        clase actual
        Se usa para crear una servidor socket el cual recibira los objetos que son enviados
        por la clase servidor*/
        Thread mihilo = new Thread(target: this);
        /*Se utiliza el metodo 'start', para iniciar la ejecucion del hilo en segundo
        plano*/
        mihilo.start();
    }
```

En la clase cliente se cuenta con un método principal, en donde hay una instancia de la clase 'VistaCliente', en el constructor de esta clase se estipula la posición del frame como la creación de la instancia de la interfaz gráfica del chat, creada de la clase 'LaminaVistaCliente'.

En la clase 'LaminaVistaCliente' se hereda de la clase de JPanel y se aplica un Runnable por motivos de la implementación de un hilo, dentro de esta clase se establecen los campos para el nombre, la dirección ip, el mensaje a mandar y el botón llamado 'miBoton' al cual se crea un evento de acción con un ActionListener al cual se le pasa una instancia de una clase llamada 'EnviaTexto', para enviar el mensaje al servidor y que este lo reenvie al cliente de destino, también se crea un botón llamado 'consultarArchivo', ya que en este programa se creó una clase FTP, con la cual se realiza un conexión a un servidor FTP local, este botón permite visualizar los archivos de este servidor por medio de protocolo FTP, esta funcionalidad es completamente independiente del estudio del socket, por otro lado se crea un hilo para poder recibir los mensajes de respuesta del otro cliente en segundo plano.

```java
//paquete de datos de texto
//serializable capas de convertirce en bits y reconvetir los bits para enviar por la red
/*Se crea la clase 'PaqueteEnvio', para crear objetos que contengan los parametros
de nick, ip y mensaje, estos objetos se empaquetaran y se enviaran por la red
por medio de la comunicacion del socket*/
  class PaqueteEnvio implements Serializable{ //Crear objeto para pasar los tres parametros
    /*variables para almacenar los datos del cliente para el envio del mensaje*/
    private String nick, ip, mensaje;
    /*Se crean los metodos get y set, get para poder obtener el valor
    del atributo guardado en las variables privadas anteriormente creadas
    y se usa el metodo set para escribir el valor del atributo en la variable
    */
    public String getNick() {
        return nick;
    }
    public void setNick(String nick) { //almacenar
        this.nick = nick;
    }
    public String getIp() { //consultar
        return ip;
    }
    public void setIp(String ip) {
        this.ip = ip;
    }
    public String getMensaje() {
        return mensaje;
    }
    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }
  }
```

Antes de analizar la clase ‘EnviaTexto’, para el funcionamiento del botón que contendrá el Socket, se crea una clase llamada ‘PaqueteEnvio’, la cual es serializable, en la cual se crean las variables nick, ip, mensaje con sus respectivos set y get.

```
/*Se crea la clase 'EnviaTexto', que implementa la interfaz 'ActionListener'
lo que permite que esta clase se utilice para manejar eventos de escucha como
de boton, se usa cuando el usuario presiona el boton de Enviar en la interfaz
Esta clase se estipulo privada porque es una clase interna que solo se usa dentro
de la clase 'LaminaVistaCliente', permitiendo que solo la clase 'LaminaVistaCliente'
púeda crear instancias y utilizarlas*/
private  class EnviaTexto implements ActionListener{
    /*Se tiene un metodo 'actionPerformed(ActionEvent e)', Se usa @Override, para
    indicar que el metodo de la clase 'EnviaTexto', esta modificando el metodo
    el metodo 'actionPerformed('*/
    @Override
    public void actionPerformed(ActionEvent e) {
        //aparezca la conversacion de los clientes en campo
        /*Al componente campochat, se le agrega con el metodo append, el componente
        campo1, el campo en donde el cliente escribe el mensaje, este mensaje
        se recupera con el metodo getText()*/
        campochat.append("\n" + campo1.getText());
        /*Esta esctructura el try- catch, se usa para manejar excepciones, para
        validar errores o situaciones que sucedan durante el programa, evitando
        que el programa se cierre sin alguna notificación de error*/
        try {
        /*Se crea un socket llamado 'misocket', para iniciar una conexion con un
        servidor que tiene la IP "192.168.0.17" con un puerto de escucha 536 en donde el
        cliente swe conecte al servidor con esta IP y este puerto*/
        Socket misocket = new Socket(host: "192.168.0.17",port: 536);
        /*Se crea un objeto de la clase 'PaqueteEnvio', llamado 'datos',se usa para
        empaquetar los datos que el cliente va a enviar al servidor*/
        PaqueteEnvio datos = new  PaqueteEnvio();//empaquetar datos
        /*Se da el valor, al atributo nick con el objeto datos, se usa el metodo getText()
        para recuperar lo que se escribio en el campo y se guarda en el metodo set
        del atributo de la clase PaqueteEnvio, esto se hace con todos los atributos*/
        datos.setNick(nick: nick.getText());//guardamos datos del campo
        datos.setIp(ip: ip.getText());//guardamos datos de la ip
        datos.setMensaje(mensaje:campo1.getText());//guardamos el mensaje escrito
        /*Se crea un flujo de salida llamado 'paquete_datos'*/
        ObjectOutputStream paquete_datos = new ObjectOutputStream( out:misocket.getOutputStream());
        /*Se usa el objeto de flujo de salida, para escribit el objeto serializado
        'datos', el cual esta relacionado con 'misocket' , para poder enviarlo por la red*/
        paquete_datos.writeObject(obj:datos);
        //cerrar flujo
        paquete_datos.close();
        //cerrar socket
        misocket.close();
         } catch (IOException ex) {
        Logger.getLogger(name: Socket.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
```

Ya creada la clase anterior se crea la clase ‘EnviaTexto’ que implementa un ActionListener, ya que esta clase se le pasará como instancia al botón para que cuando presionemos el botón este ejecute todo el código de esta clase, dentro del try catch creado por el método actionPerformed, se crea un Socket llamado ‘misocket’ el cual tiene como parámetros la ip del servidor y el puerto de escucha de este servidor en este caso es el puerto 536, este mismo puerto está estipulado en la clase servidor.

Después de crea una instancia de la clase ‘PaqueteEnvio’ llamada ‘datos’, luego usamos este objeto para poder actualizar los sets de los campos de la clase ‘PaqueteEnvio’, con los valores recuperados de los gets, de las variables de los campos de la interfaz, para así guardar los datos ingresados por el usuario en una clase o objeto el cual se enviará por la red al servidor.

Se crea un flujo de salida con 'ObjectOutputStream' llamado 'paquete_datos' con este objeto usamos el método 'writeObject' al cual le pasamos como parámetro el objeto 'datos', el cual tiene los datos del nick, ip y campo1, haciendo esta operación este objeto serializado viaja por el puente virtual creado por el socket a la ip del servidor en donde entra por medio el puerto estipulado en el socket el cual está en constante escucha de solicitudes de cliente.

Con esto el mensaje se envía al servidor el cual tiene su lógica ya explicada, este servidor reenvía el mensaje al destinatario por medio de la ip pero este proceso de hace con un 'ServerSocket', el cual está implementado en la misma clase cliente pero por medio de un hilo de segundo plano.

```
@Override
/*Se crea el metodo run del hilo para poder crear un servidor dentro del cliente
para escuchar los mensajes que envia la clase de servidor*/
public void run() {
    try{
        /*Se crea un objeto 'ServerSocket', utilizado para abrir un puerto
        de escucha en el servidor, se crea 'servidor_cliente', una instancia del
        'ServerSocket', en donde se especifica el puerto 540 que sera el puerto
        de escucha del servidor que escuchara los mensajes reenviados por la clase
        Servidor*/
        ServerSocket servidor_cliente = new  ServerSocket(port: 540);
        Socket cliente; //canal recibe paquete socket
        /*Se crea un instancia de PaqueteEnvio llamado paqueteRecibido para
        almacenar el paquete enviado por la clase servidor*/
        PaqueteEnvio paqueteRecibido;
        //Se crea bucle infinito para escuchar siempre mensajes, por medio un hilo en segundo plano
        while(true){
            /*Se usa la variable cliente de tipo Socket para aceptar una conexion del servidor
            se usa accept() se usa para esperar y aceptar la conexion del servidor principal*/
            cliente = servidor_cliente.accept(); //escuchar todas las peticiones
            //Se crea un flujo de entrada llamado 'flujoentrada'
            ObjectInputStream flujoentrada = new ObjectInputStream(in: cliente.getInputStream());
            /*Con el objeto paqueteRecibido se guarda el flujo de entrada que es leido con el
            metodo readObject, este flujo contiene el objeto o paquete reenviado por el servidor
            con los datos del mensaje enviado por el primer cliente*/
            paqueteRecibido = (PaqueteEnvio) flujoentrada.readObject();//esta variable tiene la informacion recibida
            /*Al componente campochar, con el metodo append se le agrega el paqueteRecibido al cual se
            recuperan los datos con el metodo get*/
            campochat.append("\n " + paqueteRecibido.getNick()+ ": " + paqueteRecibido.getMensaje());
        }
    }catch(Exception e){
        System.out.println(x: e.getMessage());
    }
}
```

En este hilo se crea un 'ServerSocket' el cual tendrá un puerto de escucha de 540, se crea una instancia de la clase 'PaqueteEnvio', llamada 'paqueteRecibido', luego dentro de un while infinito aceptamos todas las peticiones que ingresen al puerto 540, la petición que recibirá este puerto será de la clase servidor que reenvia el mensaje del cliente inicial, luego creamos un flujo de entrada llamado 'flujoentrada' , con el objeto de 'paqueteRecibido' almacenamos el flujo de entrada, osea recibimos el objeto enviado por la clase servidor, con el método 'readObject ( )', con esto leemos la información que contiene este objeto, para finalizar mostramos en el panel del cliente receptor la información del cliente emisor, para completar la comunicación entre dos clientes que están en la misma red local, con un servidor que cumple la función de intermediario.

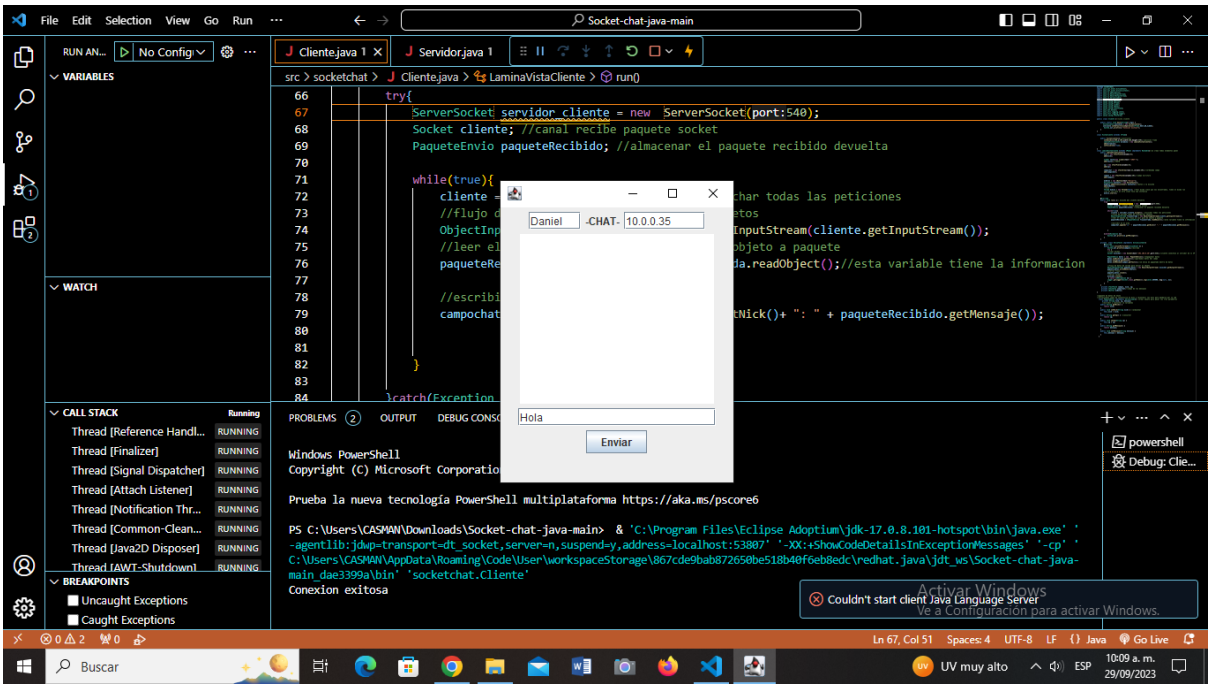
Esta sería la explicación del socket empleado para este análisis, este código fue creado y publicado por el canal de youtube de pildorasinformaticas (pildorasinformaticas, 2016).

“La clase FTP se analizará en el apartado de protocolos FTP”

4. Salida de la interfaz del socket:

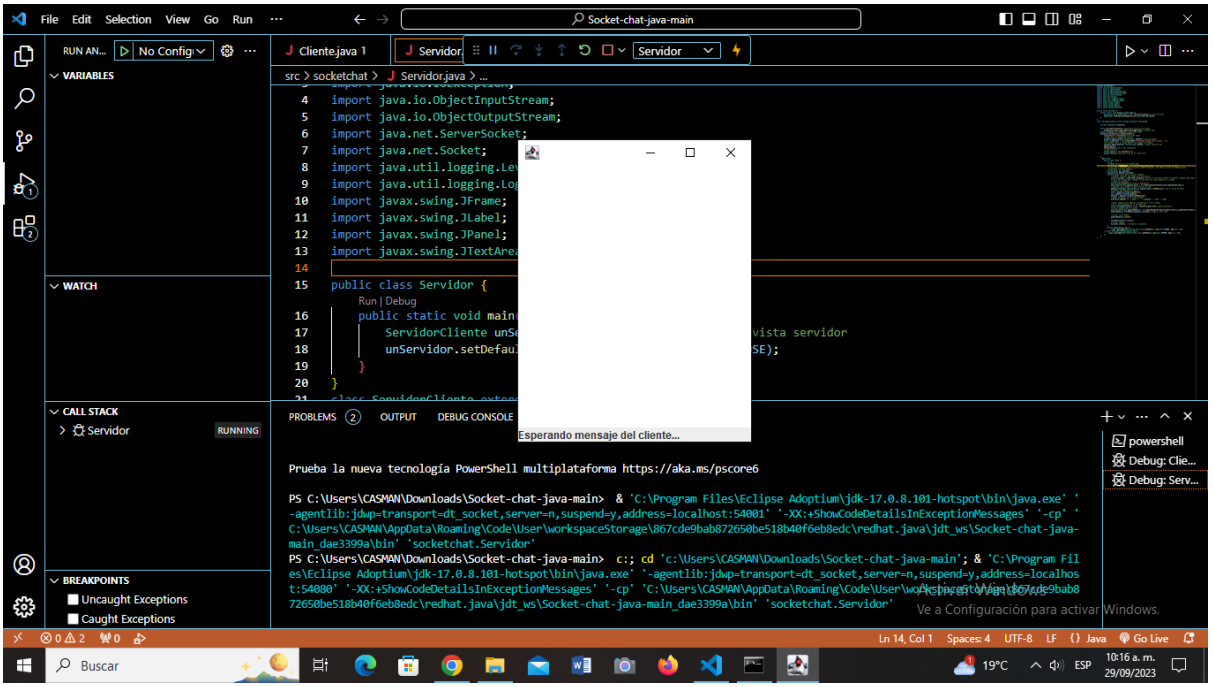
El funcionamiento del socket se aprecia en dos partes de la aplicación, el cliente y el servidor, la interfaz de estas aplicaciones se se verían de la siguiente forma(Usualmente se abre desde netbeans sin embargo se mostrará desde visual studio):

Cliente



El cliente está dividido en tres franjas por así decirlo, en donde la primera franja aparece el contenedor de texto donde se ingresara el nombre de nuestro usuario o nick, en el segundo campo se ingresa la dirección IP del equipo al que queremos enviarle el mensaje, este campo nos permite el envío de los datos por la red, por otro lado en la tercera franja o campo podemos visualizar los datos o los mensajes que enviamos al otro cliente, además de la respectiva respuesta que recibamos del cliente y por último se cuenta con el botón de envío que activa el protocolo TCP/IP para enviar el contenido o paquete de datos a través del servidor al otro computador establecido.

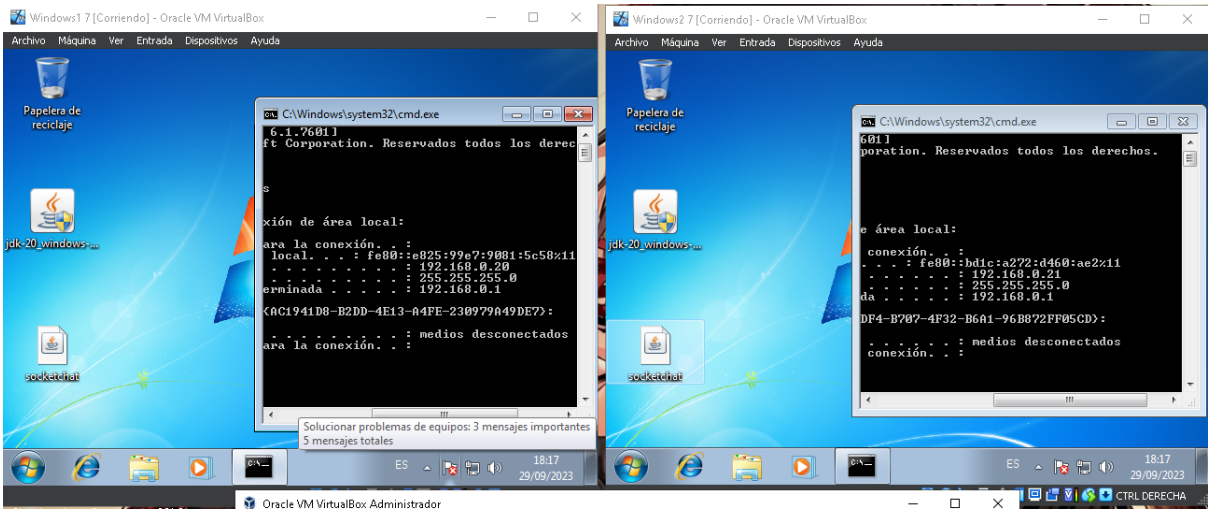
Servidor



En la interfaz del servidor no hay mucho que mostrar solo que aparece el mensaje esperando mensaje del cliente y eso es porque hasta que el cliente no de clic en el botón no va a ocurrir nada. Dentro de esta interfaz funciona la ip que da la direccion de cual es el servidor por el cual atraviesa y cuando se envíe un mensaje, cuando un cliente envíe un mensaje aparecerá el nombre del usuario que lo envió, la ip del computador de destino y el contenido del mensaje, el servidor muestra esta información y la reenvía por la red al cliente receptor por medio de la Ip recibida del cliente emisor.

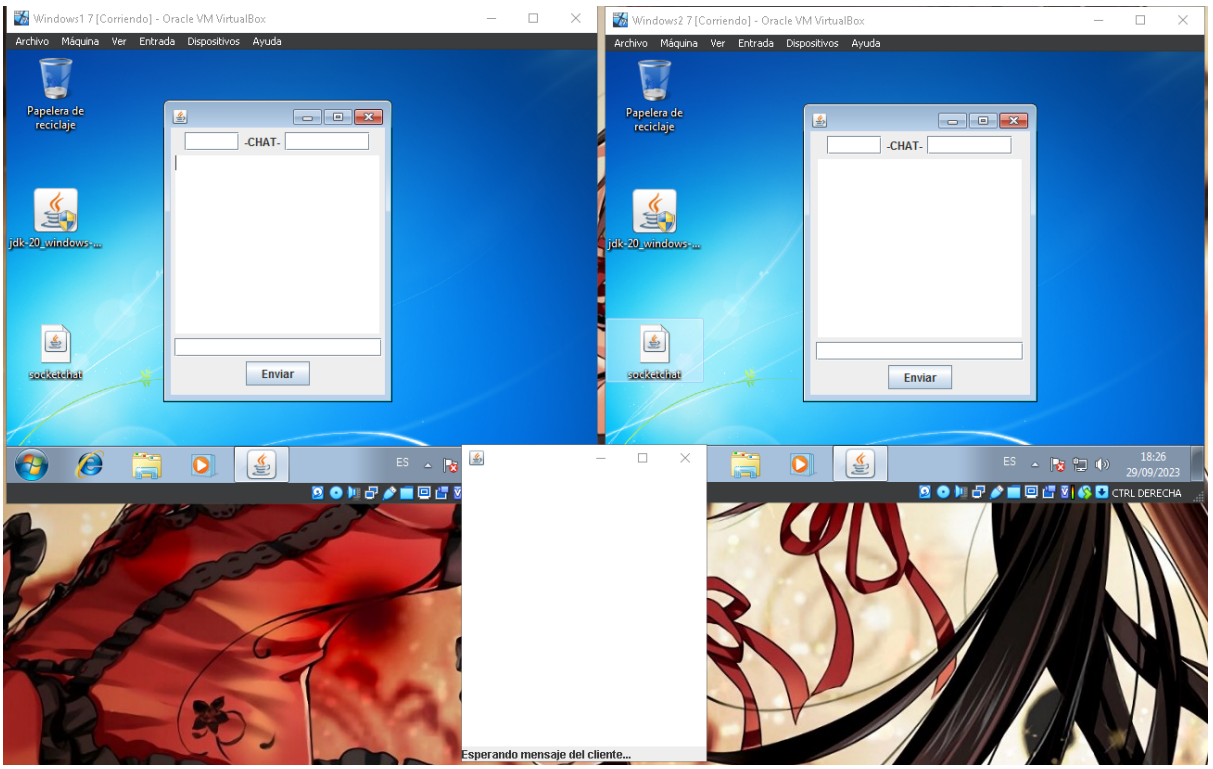
Funcionamiento por medio de la virtualización:

El socket funciona a través de tres computadores ya sean físicas o virtuales, en este ejemplo se usará una computadora física como servidor la cual virtualizara dos sistemas operativos con los dos clientes para hacer la comunicación del socket

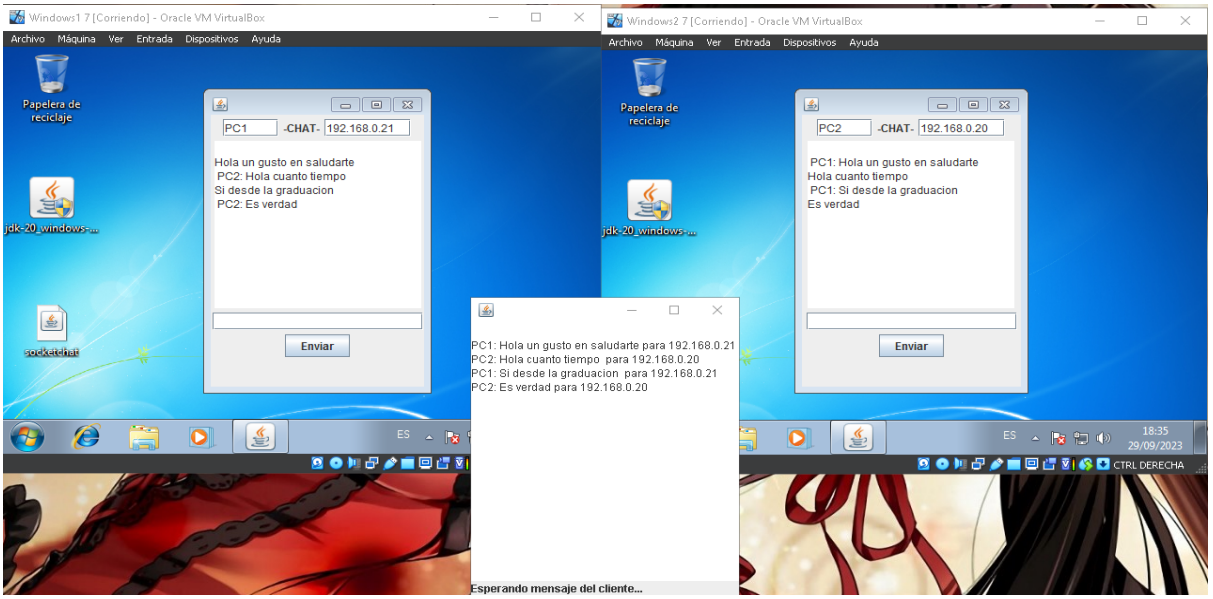


Dentro de la interfaz de cada cliente se debe agregar la dirección IP del servidor como el puerto de escucha de este servidor esté utilizando, en este caso el servidor sera la máquina física, para poder realizar la comunicación por medio de una red local, las dos máquinas virtuales deben estar en una misma red y cada contar con una dirección IP, para poder enviar los mensajes, el PC1 cuenta con la dirección: 192.168.0.20, y el PC2 cuenta con la dirección: 192.168.0.21 y la direccion del servidor es: 192.168.0.17, como se puede ver tanto el servidor como las maquinas estan en la misma red.
En cada máquina virtual se creó un JAR con la clase cliente en donde se apunta con el socket a la IP del servidor y al puerto de escucha que en este caso es 536.

Para hacer la simulación de la comunicación se ejecuta en la máquina física la clase servidor y en las máquinas virtuales el JAR de cliente.



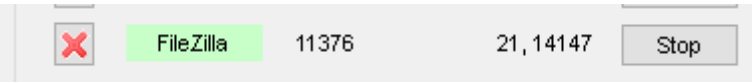
Después de que las interfaces se están ejecutando en las maquinar desde el PC1 envía un mensaje al PC2 por medio de la IP, pero el servidor toma la tarea de intermediario dente los dos clientes este recibe la información la muestra y la reenvía al cliente de destino, luego el cliente receptor lee el mensaje y escribe un mensaje al primer cliente en donde lo vuelve enviar el servidor este lo muestra y lo envía al primer cliente como respuesta a continuación se puede ver un ejemplo de la conversación de las tres máquinas.



5. Uso de protocolos FTP, HTTPS

Protocolo FTP: El protocolo FTP (File Transfer Protocol), es un protocolo de transferencia de archivos en un modelo de cliente y servidor, es utilizada para cargar o descargar archivos desde otro dispositivo, como usa un servidor para la comunicación este usa el puerto 21, además para realizar la conexión con el servidor no solo es necesario utilizar su puerto, ya que el FTP utiliza un paso de autenticación, por medio de un usuario o contraseña.
El servidor en este modelo se encarga de almacenar los ficheros o documentos, en donde los clientes que se conecten a este servidor por medio de su puerto e ingresando las credenciales de Usuario y contraseña, pueda descargar, subir o eliminar archivos del servidor (Faci, 2019).

El protocolo FTP y el socket son medios para la comunicación entre cliente y servidor, son medios de comunicación distintos FTP se usa para la transferencia de archivos entre un cliente y servidor, por otro lado los sockets no solo permiten enviar archivos entre computadoras sino además permiten que las computadoras se hablen entre sí como en un chat de texto.
En este proyecto se añadió el protocolo FTP a parte del socket con el protocolo TCP/IP, con el objetivo de estudiar este protocolo,primero en la clase cliente se crea una clase FTP, en donde se realiza la conexión a un servidor de Xampp creado de manera local en la computadora con Filezilla, en donde se creo un usuario con un nombre y contraseña.



Este servidor tiene un directorio local de la computadora asignado, el código del socket no se toca ya que la lógica de los sockets de java y el FTP, son completamente distintas para ejecutar estos dos protocolos se realizó de forma separada, se creo un boton en la interfaz del chat llamado “Ver archivos FTP”, al ejecutar este botón se conecta con el servidor local de FTP y nos trae los documentos o archivos que este servidor contiene, es una forma de aplicar el FTP para realizar una conexión con un servidor mostrando sus archivos aplicando el modelo cliente servidor, en donde un cliente en este caso el chat solicita información a un servidor, el servidor local y este le responde con la petición de los documentos.

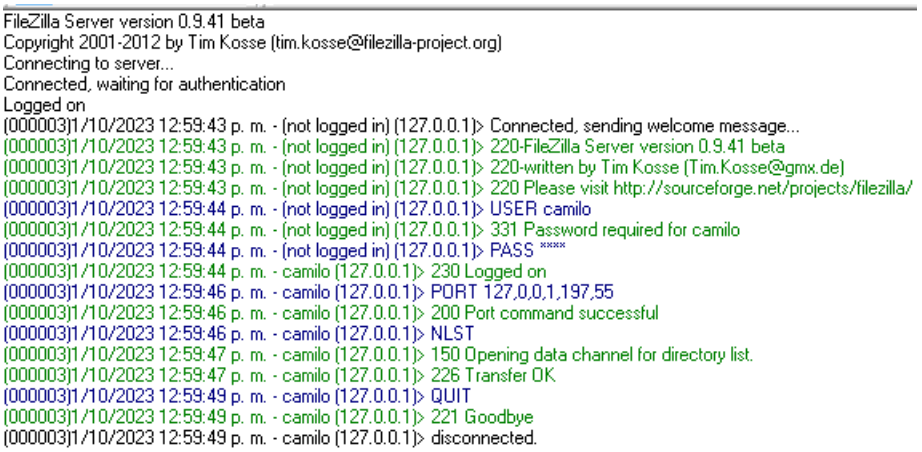
Clase FTP

```java
//clase con el protocolo FTP independiente del funcionamiento del socket de mensajes
/*Esta clase permite que el cliente interactue con un servidor FTP
se puede conectarse aun servidor y listar los archivos de este servidor*/
private class FTP  implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        /*Esta esctructura el try- catch, se usa para manejar excepciones, para
        validar errores o situaciones que sucedan durante el programa, evitando
        que el programa se cierre sin alguna notificación de error*/
        try {
            /*Variables utilizadas para la conexion FTP*/
            /*Almacena la IP o el nombre de dominio del servidor FTP*/
            String server = "127.0.0.1";
            /*Numero de puerto que se usa para la conexion FTP*/
            int port = 21;
            /*Se almacena el nombre de usuario*/
            String username = "camilo"; //lo cree en el servidor del xammp permisos de archivos
            /*Guarda la contraseña del usuario*/
            String password = "Puli";
            /*Se crea una instancia de la clase FTPClient, de la biblioteca Apache
            Commons Net, se usa para establecer y gestionar la conexion FTP, entre el
            cliente y el servidor FTP*/
            FTPClient clienteFTP = new FTPClient();
            //validar conexion
            try {
                /*De la instancia clienteFTP, se usa el metodo connect, en donde se le
                pasa como parametros el server y el port, para conectar el clienteFTP al
                servidor FTP*/
                clienteFTP.connect(hostname: server, port); //creamos la conexion
                /*Se crea la variable respuesta, para guardar el codigo de respuesta
                del servidor FTP, se usa el metodo getReplyCode, quees usado para obtner el codigo
                de respuesta del servidor, estos codigos pueden comunicar si la operacion fue
                exitosa o se produjo un error*/
                int respuesta = clienteFTP.getReplyCode();//codigo de respuesta del servidor
                //si la respuesta del servidor no es valida
                /*Se usa para verificar si el codigo de respuesta es positivo o negativo,
                en est caso con el signo !, realiza la valdiacion si la respuesta es diferente
                de verdarero o True imprime el JOptionPane y no ejecuta lo demas y
                llega al catch*/
                if(!FTPReply.isPositiveCompletion(reply: respuesta)){
                    JOptionPane.showMessageDialog(parentComponent:null, message:"Algo salio mal con la conexion del servidor ");
                }/*Si el codigo si es True imprime este mensaje y continua el codigo */
                 JOptionPane.showMessageDialog(parentComponent:null, message:"La conexion con el servidor fue exitosa");
                 /*Esta variable alamacena si el inicio de sesion fue verdadero o falso,
                 con el metodo login, se inicia sesion al servidor FTP*/
                boolean inicioCorrecto = clienteFTP.login(username, password);//true
                if(inicioCorrecto){//Si el inicio es correcto
                    JOptionPane.showMessageDialog(parentComponent:null, message:"Se inicio sesion en el servidor FTP de Xammp");
                }else{//si no es correcto
                     JOptionPane.showMessageDialog(parentComponent:null, message:"No se pudo hacer la conexion servidor las credenciales no son validas");
                }
            } catch (IOException ex) {
                 JOptionPane.showMessageDialog(parentComponent:null, "Error: " + ex.getMessage());
            }
            // Crear una cadena para almacenar la lista de archivos del servidor FTP
            StringBuilder listaArchivos = new StringBuilder();
            // Se crea un vestor de tipo Sting para almacenar los archivos del servidor
            String[] archivos;
            try {
                /*En el vector se almacena, con el metodo listNames, se obtienen los nombres
                o archivos del directorio del servidor*/
                archivos = clienteFTP.listNames();
                /*Se usa un for-each, se declara la variable archivo, en cada giro del
                ciclo esta variable tendra el valor del elemento actual del arreglo archivos
                el operador : se usa para decir que se esta iterando y que se esta
                asignando cada elemento del arreglo a la variable 'archivo'*/
                for (String archivo : archivos) {
                    /*Se usa la variable listaArchivos para almacenar cada lista de los
                    nombres de la variable archivo, con un salto de linea, en cada giro
                    añade un salto de linea para lista los nombres*/
                    listaArchivos.append(str:archivo).append(str:"\n");
                }
                /*Se imprime la lista de archivos del servidor FTP*/
                JOptionPane.showMessageDialog(parentComponent:null, message:listaArchivos.toString(), title: "Lista de Archivos Servidor FTP",  messageType:JOptionPane.INFORMATION_MESSA
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(parentComponent:null, "Error: " + ex.getMessage());
            }
            //cierra sesion
            clienteFTP.logout();
            //Se desconecta cliente del servidor FTP
            clienteFTP.disconnect();
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(parentComponent:null, "Error: " + ex.getMessage());
        }
    }
}
```

Interfaz con el el protocolo FTP

Al dar click en el botón nos enviará un mensaje comunicando si la conexión fue exitosa o si ocurre algún problema, si se logra conectar con el servidor nos mostrará una nueva ventana con los datos del servidor.

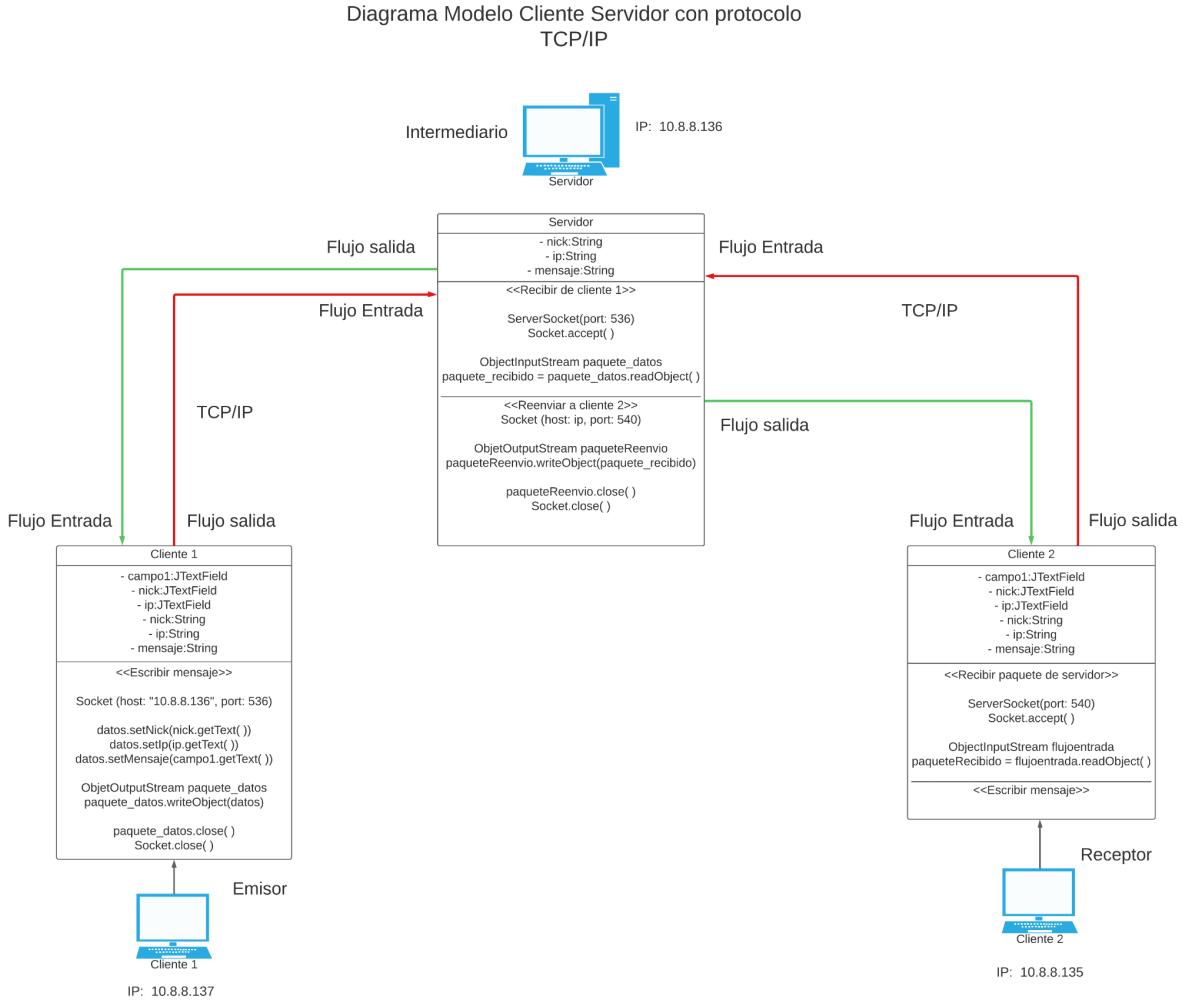
Cada vez que realizamos una petición en la consola del servidor de Filezilla nos muestra la petición realizada por el botón de la interfaz, el ingreso de las credenciales y cuando cerramos la ventana se cierra la sesión y cierra la conexión.



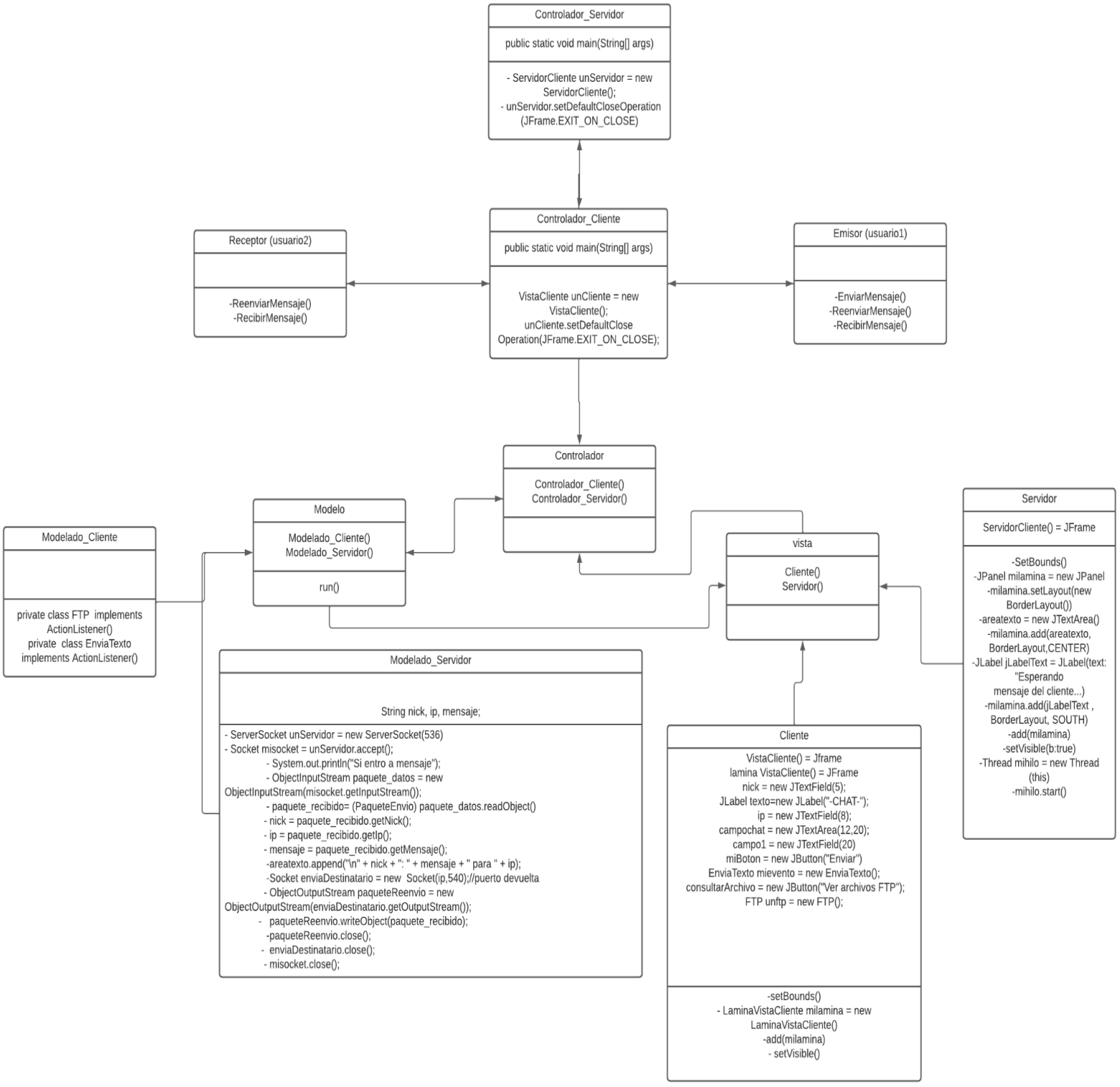
Protocolo HTTP y HTTPS: El protocolo HTTP es utilizado para la transferencia de hipertexto, más conocidas como las páginas web, los hipertextos son un formato de documentos de texto que se pueden acceder a ellos por medio de un hipervínculo. Este protocolo usa el puerto 80 para la conexión, este protocolo no implementa mecanismos de seguridad, este protocolo utiliza el modelo cliente servidor, en donde el servidor almacena documentos de varios clientes, para que los mismos clientes puedan acceder a ellos (Faci, 2019).

El protocolo HTTPS es igual al protocolo HTTP pero con seguridad, este protocolo usa una conexión segura, usa un sistema de cifrado SSL, esto genera que los datos se envíen de una manera segura (Baldikov, 2022).

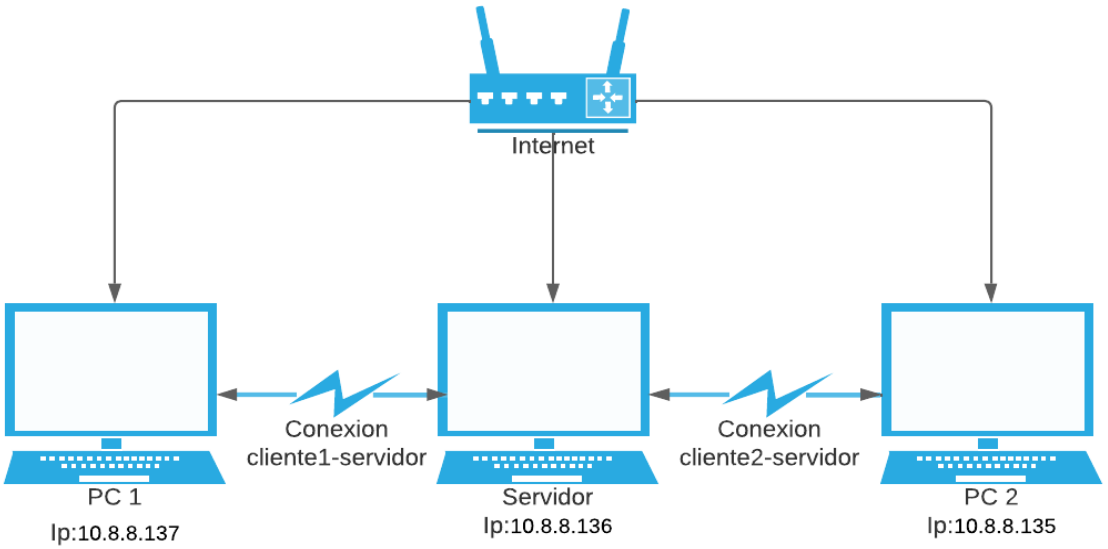
6. Diagrama con el modelo cliente servidor (M-C-S) usando TCP/IP:



7. Diagrama con el modelo vista controlador (M-V-C) socket:



8. Topología del socket:



Bibliografía

Baldikov, N. (15 de Enero de 2022). *¿Qué es el Protocolo HTTPS y para qué sirve? Guía completa*. Obtenido de siteground: https://es.siteground.com/blog/que-es-https-y-para-que-sirve-guia-completa/

Faci, S. (7 de Mayo de 2019). *Programación en red*. Obtenido de psp.codeandcoke: https://psp.codeandcoke.com/apuntes:sockets#ftp

pildorasinformaticas. (5 de Enero de 2016). Curso Java. Sockets I. Vídeo 190 [video]. Youtube. Obtenido de https://www.youtube.com/watch?v=L0Y6hawPB-E&t=12s&ab_channel=pildorasinformaticas

proofpoint. (2023). *¿Qué es el modelo OSI?* Obtenido de proofpoint.com: https://www.proofpoint.com/es/threat-reference/osi-model