Universidad Politécnica Salesiana

Carrera de Computación

Materia:

Sistemas distribuidos

Practica:

Mecanismos de comunicación - Sockets

Docente:

Ing. Cristian Timbi

Integrantes:

Juan Francisco Quizhpi Fajardo

Comunicación por Sockets

Introducción

Los sockets representan un aspecto esencial en la comunicación de redes informáticas debido a que se emplean extensamente en el desarrollo de aplicaciones que requieren el intercambio de datos ente dispositivos conectados a la red. Un socket representa el extremo de una conexión que facilita la comunicación entre dos programas ya sea que se encuentren alojados en la misma máquina o en distintos equipos conectados a través de la red.

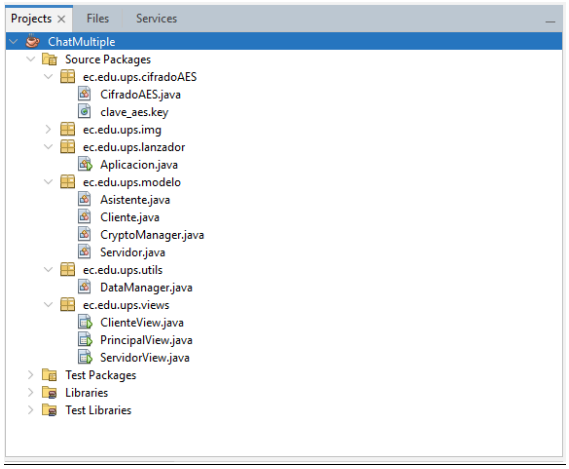
Los mecanismos de comunicación por sockets son implementados a nivel de programación de aplicaciones y se sustentan en protocolos de comunicación como el TCP o UDP. Por un lado, los sockets TCP nos garantizan una comunicación confiable y orientada a la conexión, mientras que los sockets UDP nos proporcionaran una comunicación centrada a ser ágil, pero sin ser orientada a la conexión.

Pasos de la comunicación por Sockets

1. Creación del Socket: El programa crea el socket con el que estable conexión con otro programa. Esta conexión se realiza por medio de la llamada al sistema de red.
2. Conexión: Los sockets TCP realizan una conexión cliente servidor. El servidor estará activamente esperando establecer conexiones mientras los clientes trataran de conectarse usando la dirección y puerto del servidor.
3. Envió y recepción de datos: Establecida la conexión, los programas son capaces de enviar y recibir datos por medio de los sockets.
4. Cierre de la conexión: Cuando concluye la comunicación se cierra la conexión para liberar recursos y terminar la conexión.

Desarrollo del proyecto

Para el desarrollo de la siguiente practica se a optado trabajar con java y el IDE de NetBeans. Este será la estructura de nuestro proyecto.



Paquete cifradoAES

Para la aplicación se a optado por el uso de la encriptación AES. Con esta garantizaremos la confidencialidad de la comunicación de nuestro Chat Múltiple, se usará una clave

compartida. Esta clave será almacenada el un archivo "clave_aes.key" que se encuentra alojado en nuestro proyecto.

Al optar por una clave de este tipo nos aseguramos de que los que tengan esta sean capaces de encriptar y desencriptar los mensajes. Estas es la implementación que se realiza.

```java
/**
 *
 * @author juanf
 */
public class CifradoAES {
    public static byte[] encrypt(String mensaje, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance(transformation: "AES");
        cipher.init(opmode: Cipher.ENCRYPT_MODE, key:secretKey);
        byte[] encryptedBytes = cipher.doFinal(input: mensaje.getBytes(charset:StandardCharsets.UTF_8));
        return encryptedBytes;
    }

    public static String decrypt(byte[] encryptedBytes, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance(transformation: "AES");
        cipher.init(opmode: Cipher.DECRYPT_MODE, key:secretKey);
        byte[] decryptedBytes = cipher.doFinal(input: encryptedBytes);
        return new String(bytes: decryptedBytes, charset:StandardCharsets.UTF_8);
    }

    public static SecretKey getSecretKey() throws FileNotFoundException, IOException {
        byte[] keyBytes;
        try (FileInputStream fis = new FileInputStream(DataManager.getDataPath() + "cifradoAES/clave_aes.key")) {
            keyBytes = fis.readAllBytes();
        }
        return new SecretKeySpec(key:keyBytes, algorithm: "AES");
    }
}
```

El código implementa la encriptación y desencriptación utilizando el algoritmo de cifrado AES. Con la función ` getSecretKey () ` obtenemos la clave compartida que se encuentra en nuestro en nuestro proyecto. La función ` encrypt () ` cifra un mensaje utilizando una clave AES. La función ` decrypt () ` desencripta un mensaje cifrado utilizando una clave AES.

Paquete Lanzador

Aplicacion.java

Se crea una instancia para “PrincipalView “para proceder a lazar la aplicación de Chat Múltiple.

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ec.edu.ups.lanzador;

import ec.edu.ups.views.PrincipalView;

/**
 *
 * @author juanf
 */
public class Aplicacion {
    public static void main(String[] args) {
        PrincipalView principal = new PrincipalView();
        principal.setVisible(b: true);
    }
}
```

Paquete Modelo

Asistente

Creamos la a clase asistente que implementa la interfaz Runnable En la cual tendremos un atributo Socket que representada el socket para la comunicación también "DataInputStream in" el cual utilizamos para leer datos de flujo de entrada del socket entre otros atributos.

```
/**
 *
 * @author juanf
 */
public class Asistente implements Runnable {

    //Declaramos los atributos de la clase asistente
    private Socket sc;
    private DataInputStream in;
    private Servidor server;
    private boolean centinela;

    //constructor de la clase Asistente
    public Asistente(Socket sc, Servidor server) {
        this.sc = sc;
        this.server = server;
        this.centinela = true;
    }
```

El método "run ()" se ejecutará mientras "centinela" sea true. El bucle, utiliza un DataInputStream para leer un mensaje desde el socket (sc. getInputStream ()). Los asistentes enviaran la notificación al servidor y este notificara los demás usuarios los mensajes. Se implemento un mecanismo para notificar la desconexión de usuarios al servidor al recibir el mensaje cerra. Como los mensajes se encuentran encriptados se requiere usar el método de desencriptación AES para verificar que el mensaje es sobre la desconexión de un usuario. Los mensajes seguirán encriptados para el servidor en la demostración será más cómo funciona este mecanismo.

```
@Override
public void run() {
    try {
        String mensaje;
        in = new DataInputStream(in: sc.getInputStream());

        while (centinela == true) {

            //Leectura de mensaje del servidor
            mensaje = in.readUTF();
            //Desciframos el mensaje para saber desconexion
            String mensajeDecifrado = CifradoAES.decrypt(encryptedBytes: Base64.getDecoder().decode(src:mensaje), secretKey: CifradoAES.getSecretKey());
            if (mensajeDecifrado.equalsIgnoreCase(anotherString: "Cerrar")) {

                centinela = false;

                server.desconectarCliente(mensaje:"Cliente Desconectado");
                in.close();
                sc.close();

            } else {
                //Enviamos mensaje al servidor
                server.notificacion(mensaje);
                server.chatEncriptadoServidor(mensaje);
            }

        }

    } catch (IOException ex) {
        Logger.getLogger(name: Asistente.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
    } catch (Exception ex) {
        Logger.getLogger(name: Asistente.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
    }
```

Cliente

La función de la clase de Cliente es la de establecer conexión con el servidor a través de un socket y enviar mensajes. Se declaran las variables para la creación del socket a la para que inicialicemos el socket.

```java
/**
 *
 * @author juanf
 */
public class Cliente {

    //Declaramos atributos de la clase cliente
    private Socket socketCliente;
    private int puerto;
    private DataOutputStream out;
    final String HOST = "localhost";

    //Contructor de la clase cliente
    public Cliente(int puerto) {
        this.puerto = puerto;
        iniciar();
    }

    //Metodo para inicializar el socket
    public void iniciar() {
        try {
            //Se crea el socket
            socketCliente = new Socket(host: HOST, port: puerto);
        } catch (IOException ex) {
            Logger.getLogger(name: Cliente.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        }
    }
```

Creamos el método para enviar los mensajes al servidor, en primer lugar se verifica si el socket se encuentra disponible si lo está usamos los métodos de encriptación AES para que los mensajes que se envíen al servidor se encontraran cifrados.

```java
    //metodo para enviar mensajes al servidor
    public void enviarMensaje(String msj) throws Exception {
        if (socketCliente.isClosed()) {
            System.out.println(x: "Socket sin acceso");
        } else {

            try {
                out = new DataOutputStream(out: socketCliente.getOutputStream());

                byte[] mensajeCifrado = CifradoAES.encrypt(mensaje: msj, secretKey: CifradoAES.getSecretKey());
                out.writeUTF(str: Base64.getEncoder().encodeToString(src: mensajeCifrado));

            } catch (IOException ex) {
                Logger.getLogger(name: Cliente.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
            }
        }

    }
```

Clase servidor

La clase servidor acepta conexiones entrantes de los usuarios, además se instancia la clase servidor con su respectiva vista. Con el método “notificación” recibimos los mensajes para proceder a desencriptarlos y hacerles llegar a los usuarios. Por otro los métodos de desconectar y encriptar asentaran en la vista del servidor los detalles del proceso del chat múltiple.

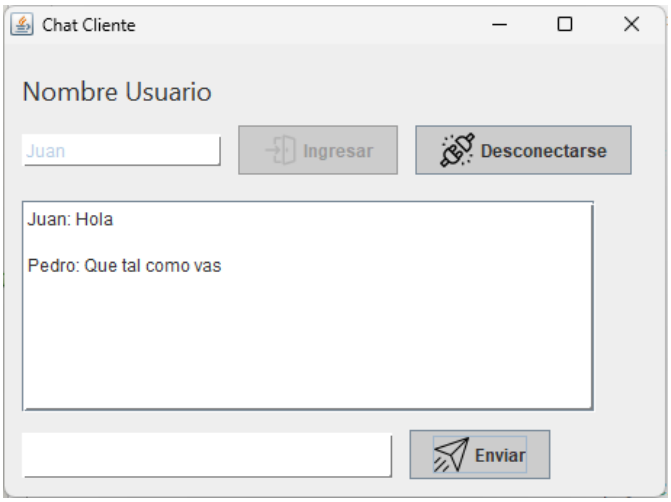
```java
/**
 *
 * @author juanf
 */
public class Servidor extends Observable implements Runnable {

    //Atributos de la clase servidor
    private ServerSocket server;
    private Socket sc;
    private int puerto;
    private ServidorView vistaServidor;

    //
    //
    //Constructor de la clase Servidor
    public Servidor(int puerto) {
        this.puerto = puerto;
    }

    //Metodo para instaciar la clase Servidor con la vista servidor
    public void setVistaServidor(ServidorView vistaServidor) {
        this.vistaServidor = vistaServidor;
    }

    //Metodo para notificar a los observadores
    public void notificacion(String mensaje) throws IOException, Exception {


        //notificamos los cambios a los observadores
        this.setChanged();
        //Decodificar
        System.out.println(x: mensaje);
        String mensajeDecifrado = CifradoAES.decrypt(encryptedBytes: Base64.getDecoder().decode(src:mensaje), secretKey: CifradoAES.getSecretKey());

        this.notifyObservers(mensajeDecifrado+ "\n");
        this.clearChanged();

    }
```

Con el “Método run ()” se inicializa el ServerSocket para que acepte conexiones entrantes. Una vez se recibe una conexión entrante, se notifica a la vista que se ha conectado usuario y crea un objeto Asistente con el manejaremos la comunicación con el usuario además es necesario la creación de un hilo para el usuario que se conecto en ese momento.

```java
    //Metodo para informar la desconeccion de un cliente
    public void desconectarCliente(String mensaje) {
        vistaServidor.notificarConexion(msj:mensaje);
    }

    public void chatEncriptadoServidor(String mensaje){
        vistaServidor.chatEncriptado(msj:mensaje);
    }

    @Override
    public void run() {
        System.out.println(x: "Servidor Activo");
        try {
            server = new ServerSocket(port: puerto);

            while (true) {

                sc = server.accept();
                vistaServidor.notificarConexion(msj:"Cliente Conectado");
                Asistente asistente = new Asistente(sc, server: this);
                Thread t = new Thread(task: asistente);
                t.start();

            }

        } catch (IOException ex) {
            Logger.getLogger(name: Servidor.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
    }
```

Paquete Views

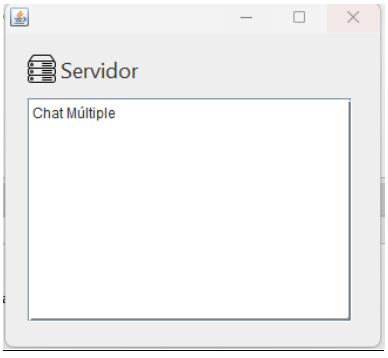
Cliente View



Ventana de Chat de uno de los clientes o usuarios esta permite interactuar con otros enviando mensajes. La acción "btnIngresarActionPerformed ()" garantizara que un usuario ingrese su nombre. Con la acción "btnEnviarActionPerformed" obtenemos el mensaje escrito en el label y lo enviamos al servidor con el uso del método "enviar Mensaje" este será el que encriptará el mensaje que llegará al servidor.

```
private void btnEmpezarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (txtNombre.getText().equalsIgnoreCase(anotherString: "")) {

    } else {
        nombre = txtNombre.getText();
        cliente = new Cliente(puerto: 12345);
        txtMensaje.setEnabled(enabled:true);
        btnEnviar.setEnabled(b: true);
        btnSalir.setEnabled(b: true);
        txtNombre.setEnabled(enabled:false);
        btnEmpezar.setEnabled(b: false);
    }
}

private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    mensaje = txtMensaje.getText();
    try {
        cliente.enviarMensaje(nombre +": " + mensaje + "\n");
    } catch (Exception ex) {
        Logger.getLogger(name: ClienteView.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
    }
    txtMensaje.setText(t: "");
}
```

Servidor View

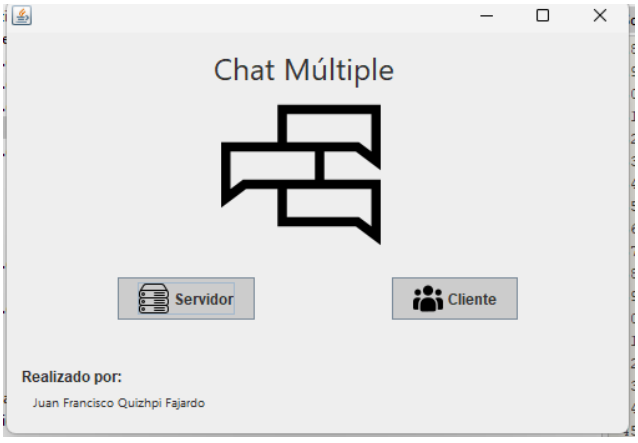


Como ya sabemos esta ventana es implementada en la clase Servidor. Los métodos notificar conexión y chat encriptar lo que se hace es mostrar el proceso de chat entre los usuarios.

```
/**
 *
 * @author juanf
 */
public class ServidorView extends javax.swing.JFrame{

    //implements Observer
    /**
     * Creates new form ServidorView
     */
    public ServidorView() {
        initComponents();
        this.txtHistorial.append("Chat Múltiple"+"\n");
    }

    //notificacin mensaje al servidor
    public void notificarConexion(String msj){
        this.txtHistorial.append(msj+"\n");
    }

    public void chatEncriptado(String msj){
        this.txtHistorial.append("Chat Encriptado: \n"+msj+"\n");
    }
```

Principal View

Esta interfaz gráfica se encargará de iniciar el servidor u simular los usuarios que se conectaran al servidor.



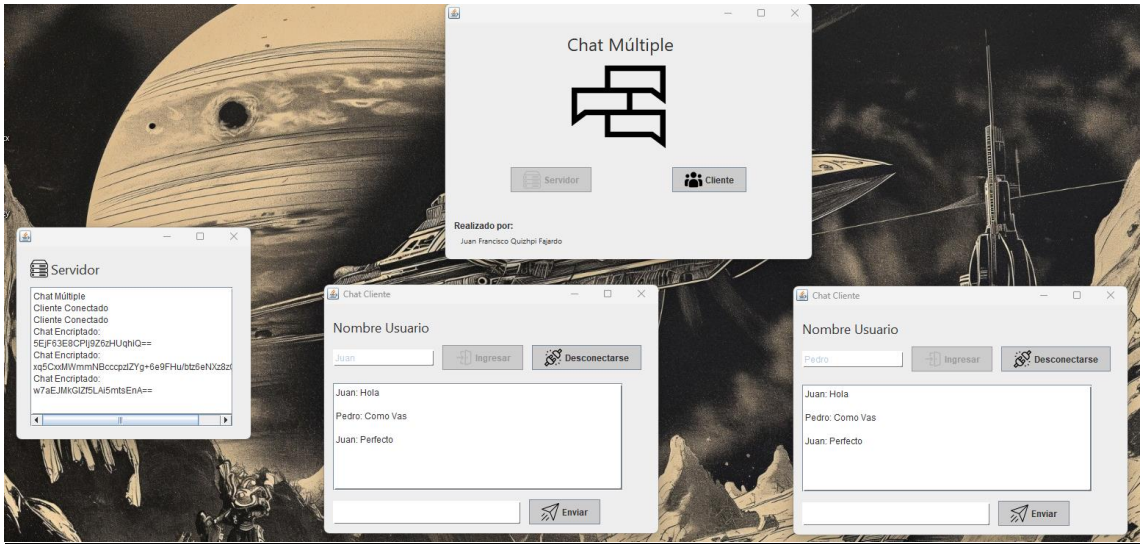
Las acciones de cada botón tendrán un mecanismo controlara que se inicialice únicamente una sola vez el servidor y que se puede lanzar varios clientes si es necesario.

```java
private void btnServidorActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    server = new Servidor(puerto: 12345);
    Thread t = new Thread(task: server);
    t.start();

    ServidorView sv = new ServidorView();
    sv.setVisible(b: true);
    //server.addObserver(sv);
    server.setVistaServidor(vistaServidor: sv);
    btnServidor.setEnabled(b: false);
}

private void btnClienteActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ClienteView cv = new ClienteView();
    server.addObserver(o: cv);
    cv.setVisible(b: true);
}
```

Conclusión

Con el desarrollo de la práctica de Sockets se ha comprendido el uso de sockets y como estos se comunican, para el caso de la encriptación de los mensajes el uso de AES y la clave compartida nos permitió que los mensajes sean confidenciales y que la única forma de desencriptar sea cuándo se tenga la clave compartida.

Demostración funcionamiento



Video: https://youtu.be/BXLO9WQxvWM