Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Distribuidos

Reporte Práctica 02:

“Cliente-Servidor”

Alumno:

Sigala Morales Said 2022630413

Grupo: 7CM1

Maestro: Chadwick Carreto Arellano

Fecha de realización: 28 de febrero de 2025

Fecha de entrega: 10 de marzo de 2025

Antecedente

La arquitectura cliente-servidor es un modelo de comunicación en sistemas distribuidos donde **los clientes realizan solicitudes y los servidores las procesan y responden.** Este paradigma surgió en la década de 1970 con el auge de las redes de computadoras y ha evolucionado con el desarrollo de internet, facilitando la creación de aplicaciones escalables y eficientes (Tanenbaum & Wetherall, 2011).

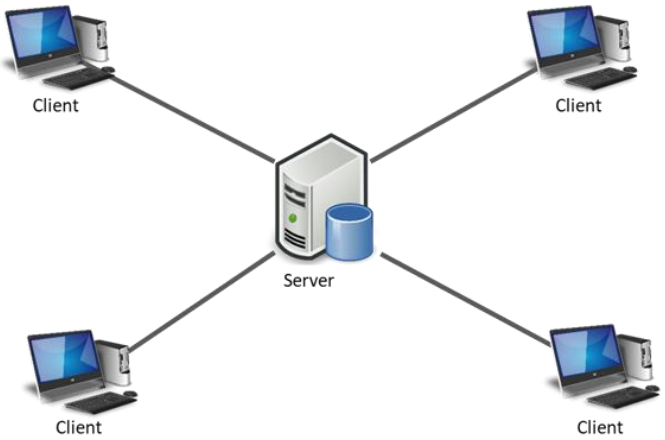


Figura 1: Arquitectura Cliente-Servidor.

Uno de los principales beneficios de esta arquitectura es la **separación de responsabilidades**, lo que permite mejorar la seguridad, el rendimiento y la administración de los datos. Los servidores centralizan los recursos y servicios, mientras que los clientes acceden a ellos sin necesidad de procesar grandes volúmenes de información localmente (Stallings, 2020).

La importancia del modelo cliente-servidor radica en su **aplicabilidad en diversos entornos**, desde bases de datos hasta aplicaciones web y sistemas en la nube. Su flexibilidad ha permitido la creación de servicios como correo electrónico, redes sociales y aplicaciones empresariales, consolidándose como un estándar en el desarrollo de software moderno (Kurose & Ross, 2021).

Planteamiento del problema

Se busca desarrollar un sistema interactivo en el que un usuario pueda participar en un **juego en línea para adivinar un número.** El juego debe proporcionar al usuario la posibilidad de recibir **retroalimentación en tiempo real**, indicando si el número correcto es mayor o menor que su conjetura. Además, se debe establecer un mecanismo para determinar cuándo un

jugador ha ganado y, en consecuencia, **notificar para finalizar** la partida de manera ordenada.

También es fundamental que el sistema **maneje correctamente la desconexión** para evitar errores o bloqueos en la ejecución del programa.

Propuesta solución

Se propone **implementar una arquitectura cliente-servidor** en la que un servidor centralizado administre el juego y el cliente se conecte para participar.

El servidor tendrá la responsabilidad de generar un número aleatorio que los jugadores deberán adivinar, recibir los intentos del participante y responder con pistas que los guíen hacia la respuesta correcta.

El cliente será quien se conecta al servidor y podrá interactuar con él enviando su posible número. A través de un mecanismo de comunicación bidireccional, **recibirá respuesta que indicarán si su número es mayor o menor al objetivo.** Una vez que un jugador adivine correctamente, será notificado de que gano y va proceder a su desconexión.

Materiales y métodos empleados

Hardware

- Procesador: AMD Ryzen 5 7535HS.
- Tarjeta gráfica: NVIDIA RTX 2050.
- Memoria RAM: 20 GB DDR5 a 4800 Mhz.

Software

- Lenguaje: Java 20.
- Entorno de desarrollo: Visual Studio Code (VSCode).
- Compilador y ejecución: Terminal integrada de VSCode.

Bibliotecas utilizadas

- java.net.*: Para la comunicación entre cliente y servidor mediante sockets (ServerSocket, Socket).
- java.io.*: Para la lectura y escritura de datos mediante BufferedReader, PrintWriter, y InputStreamReader.
- java.util.*: Para la generación de números aleatorios (Random) y el manejo de colecciones (Set, HashSet).

Métodos del Servidor

- main(): Inicia el servidor y espera conexiones de clientes.
- broadcast(String message): Envía un mensaje a todos los jugadores conectados.
- isGameOver(): Verifica si el juego ha finalizado.
- setGameOver(boolean over): Cambia el estado del juego cuando un jugador acierta el número.

- getTargetNumber(): Devuelve el número aleatorio generado para adivinar.

Métodos del Cliente

- main(): Establece la conexión con el servidor y gestiona la entrada del usuario.
- readThread.run(): Hilo encargado de recibir y mostrar mensajes del servidor en tiempo real.
- consoleReader.readLine(): Captura la entrada del usuario para enviar su intento al servidor.
- serverWriter.println(input): Envía el intento del usuario al servidor.

Métodos de Gestión de Clientes en el Servidor (ClientHandler)

- sendMessage(String message): Envía un mensaje a un jugador específico.
- run(): Maneja la interacción del jugador con el servidor, procesando sus intentos.
- getSocket(): Devuelve el socket del cliente para identificar su conexión.

Flujo del Juego

- Inicio del Servidor: Se genera un número aleatorio entre 1 y 100 y se habilita la recepción de conexiones.
- Conexión de los Jugador: El cliente ingresa su nombre y comienza a enviar intentos.
- Interacción Cliente-Servidor:
- Si el número ingresado es menor, el servidor responde "El número es mayor."
- Si es mayor, responde "El número es menor."
- Si es correcto, se notifica al jugador.

- Cierre del Juego: Cuando un jugador acierta, el servidor informa y finaliza la partida.

Desarrollo de la solución

Para implementar el juego "Adivina el Número" en un entorno distribuido, se utilizó una arquitectura cliente-servidor basada en sockets TCP en Java. La solución consta de dos componentes principales:

- **Servidor (GuessNumberServer.java):** Responsable de gestionar la lógica del juego, aceptar múltiples conexiones de clientes y coordinar la partida.
- **Cliente (GuessNumberClient.java)**: Permite al jugador conectarse, enviar intentos y recibir retroalimentación en tiempo real.

1. Inicio del Servidor

El servidor se encarga de generar un número aleatorio entre 1 y 100 que los jugadores deben adivinar y mantiene una lista de jugadores activos y permite múltiples conexiones concurrentes.

```
private static Set<ClientHandler> clientHandlers = new HashSet<>();
    private static final int PORT = 5000;

    private static int targetNumber;

    private static AtomicBoolean gameOver = new AtomicBoolean(false);
```

- ***PORT:*** Define el puerto en el que el servidor escucha conexiones.
- ***targetNumber:*** Almacena el número secreto a adivinar.
- ***gameOver:*** Variable atómica que indica si el juego ha terminado.
- **clientHandlers:** Lista de jugadores conectados.

1.1 Generación del Número Aleatorio

Cuando el servidor inicia, genera el número aleatorio que los jugadores intentarán adivinar.

```
targetNumber = new Random().nextInt(100) + 1;
        System.out.println("Servidor iniciado en el puerto " + PORT);
        System.out.println("Número a adivinar (para pruebas): " + targetNumber);
```

Este número es secreto para los jugadores, pero se imprime en consola para corroborar el funcionamiento del programa.

1.2 Aceptación de Conexiones

El servidor espera conexiones de clientes de manera indefinida y crea un hilo (Thread) para cada jugador que se conecta

```
try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            // Espera y acepta solo un cliente
            Socket socket = serverSocket.accept();
            System.out.println("Cliente conectado: " + socket);
```

- ***serverSocket.accept()***: Espera y acepta nuevas conexiones.

- Se ejecuta en un hilo separado para manejar múltiples conexiones concurrentemente.

1.3 Evaluación de Intentos

Cada vez que un jugador ingresa un número, el servidor evalúa si es mayor, menor o correcto.

```java
@Override
    public void run() {
        String input;
        try {
            while ((input = reader.readLine()) != null) {
                // Si el juego ya terminó, se le notifica al jugador
                if (GuessNumberServer.isGameOver()) {
                    writer.println("El juego ya terminó.");
                    break;
                }

                int guess;
                try {
                    guess = Integer.parseInt(input.trim());
                } catch(NumberFormatException e) {
                    writer.println("Por favor, ingresa un número válido.");
                    continue;
                }

                if (guess < GuessNumberServer.getTargetNumber()) {
                    writer.println("El número es mayor.");
                } else if (guess > GuessNumberServer.getTargetNumber()) {
                    writer.println("El número es menor.");
                } else {

                    writer.println("¡Correcto! Has adivinado el número.");
                    GuessNumberServer.broadcast("El jugador " + playerName + " ha ganado el juego adivinando el número " + guess + "!");
                    GuessNumberServer.setGameOver(true);
                    break;
                }
            }
        } catch(IOException e) {
            System.out.println("Error en la conexión con el jugador " + playerName + ": " + e.getMessage());
        } finally {
            try {
                socket.close();
            } catch(IOException e) {
                System.out.println("Error al cerrar el socket: " + e.getMessage());
            }
            GuessNumberServer.removeClient(this);
        }
    }
```

- Se convierte la entrada a un número entero ***(Integer.parseInt())***.
- Se compara con el número objetivo y se envía la respuesta adecuada.
- Si el jugador acierta, se anuncia a todos los jugadores y el juego finaliza.

1.4 Notificación al jugador

El servidor envía un mensaje al jugador.

```
if (guess < targetNumber) {
                    writer.println("El número es mayor.");
                } else if (guess > targetNumber) {
                    writer.println("El número es menor.");
                } else {
                    writer.println("¡Correcto! Has adivinado el número.");
                    System.out.println("El jugador " + playerName + " ha ganado adivinando el número " + guess + "!");
                    gameOver = true;
                }
            }
```

2. Desarrollo del Cliente

El cliente se conecta al servidor y gestiona la comunicación con el usuario.

```
private static final String SERVER_HOST = "localhost";
    private static final int SERVER_PORT = 5000;
```

- Se define el **host** (localhost) y el **puerto** (5000) del servidor.

2.1 Conexión y Lectura de Mensajes

El cliente se conecta al servidor y comienza a leer los mensajes entrantes en un hilo separado.

```java
try (Socket socket = new Socket(SERVER_HOST, SERVER_PORT);
             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
             BufferedReader serverReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter serverWriter = new PrintWriter(socket.getOutputStream(), true)) {

            Thread readThread = new Thread(() -> {
                String response;
                try {
                    while ((response = serverReader.readLine()) != null) {
                        System.out.println(response);
                    }
                } catch(IOException e) {
                    System.out.println("Error al leer del servidor: " + e.getMessage());
                }
            });
            readThread.start();

            // Lee la entrada del usuario y la envía al servidor
            String input;
            while ((input = consoleReader.readLine()) != null) {
                serverWriter.println(input);
            }
        } catch(IOException e) {
            System.out.println("Error de conexión: " + e.getMessage());
        }
```

- Se abren los streams de entrada/***salida (BufferedReader, PrintWriter).***
- Se inicia un hilo para leer mensajes del servidor en segundo plano.

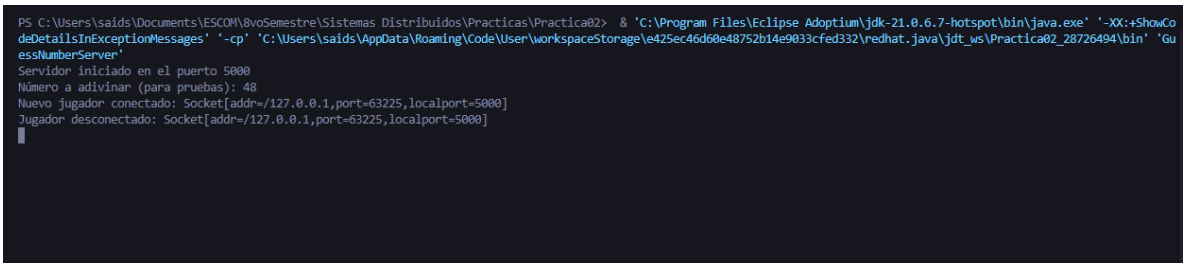
3.2 Envío de Intentos del Usuario

```java
String input;
            while ((input = consoleReader.readLine()) != null) {
                serverWriter.println(input);
            }
```

El cliente lee la entrada del usuario y la envía al servidor hasta que el juego termine.

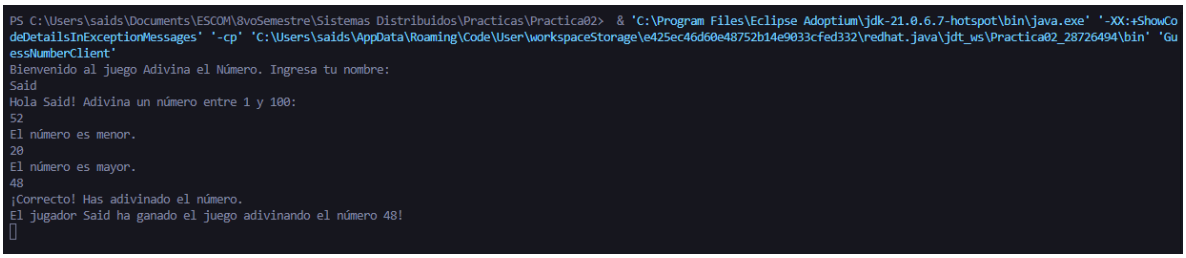
Resultados

Servidor:



Primero, el servidor se inicia correctamente en el puerto 5000, lo que indica que está listo para recibir conexiones de clientes. Luego, genera un número aleatorio secreto, en este caso 48, el cual solo se muestra en la consola del servidor para fines de prueba. Posteriormente, un jugador se conecta desde la dirección 127.0.0.1, lo que indica que la conexión proviene de la misma máquina. Finalmente, tras la desconexión del cliente, el servidor registra la salida del jugador con el mensaje "Jugador desconectado", lo que confirma que la sesión del usuario se ha cerrado correctamente.

Usuario:



El usuario introduce su nombre "Said" y el servidor le da la bienvenida, indicándole que debe adivinar un número entre 1 y 100. Luego, el jugador ingresa 52, a lo que el servidor responde que el número es menor. Posteriormente, ingresa 20, y el servidor responde que el número es mayor. Finalmente, al ingresar 48, el servidor confirma que es el número correcto y anuncia que el jugador Said ha ganado el juego, notificando a todos los clientes conectados sobre el resultado. Esto indica que el sistema está funcionando correctamente, ya que evalúa los intentos y finaliza el juego al identificar un ganador.

Conclusiones

El desarrollo del juego "Adivina el Número" basado en una arquitectura cliente-servidor demuestra la eficacia del uso de sockets TCP en Java para la comunicación en red. La estructura del servidor garantiza una gestión eficiente de las conexiones, asegurando que el juego finalice correctamente cuando un jugador acierta el número notificando a los jugadores de lo ocurrido en la partida

El correcto funcionamiento del sistema, evidenciado en la ejecución, muestra que la interacción entre cliente y servidor es fluida, con respuestas inmediatas a los intentos del usuario. Esta solución puede ser escalada y mejorada en el futuro, por ejemplo, agregando una interfaz gráfica o implementando nuevas reglas de juego.

Referencias

- Kurose, J. F., & Ross, K. W. (2021). Computer networking: A top-down approach (8th ed.). Pearson.
- Stallings, W. (2020). Computer networking with Internet protocols and technology (7th ed.). Pearson.
- Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer networks (5th ed.). Pearson.