INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE

SISTEMAS DISTRIBUIDOS

PRACTICA No. 2

MODELO CLIENTE - SERVIDOR

ALUMNO

GÓMEZ GALVAN DIEGO YAEL

GRUPO

7CM1

PROFESORA

CARRETO ARELLANO CHADWICK

FECHA DE ENTREGA

3 DE MARZO DE 2025

INDICE

ANTESCEDENTES

El modelo Cliente – Servidor es una arquitectura de sistemas de software y redes de computadoras que permite la comunicación entre un cliente, que realiza solicitudes, y un

servidor, que las procesa y responde con la información o servicio requerido. Aunque este modelo puede admitir múltiples clientes conectados a un servidor, también se utiliza en configuraciones donde un solo cliente interactúa con un único servidor, como ocurren en ciertos sistemas bancarios, cajeros automáticos (ATM) y aplicaciones de escritorio que dependen de un servidor central.

Antes de la aparición del modelo Cliente – Servidor, los sistemas de computación eran centralizados y basados en mainfraimes, donde una única computadora central procesaba todas las tareas y almacenaba los datos. Los usuarios accedían a través de terminales sin capacidad de procesamiento independiente, lo que generaba una alta dependencia del sistema central y limitaba la escalabilidad y eficiencia del procesamiento.

Años después, aproximadamente en los 80s, con el avance de las redes de computadoras y la reducción de costos en hardware, surgió la necesidad de distribuir la carga de trabajo. Esto llevo a la adopción del modelo Cliente – Servidor. Esto permitió que el cliente podía realizar parte del procesamiento localmente, mientras que el servidor se encargaba de gestionar datos y recursos compartidos. Este modelo mejoro significativamente el rendimiento y la seguridad, permitiendo una comunicación mas eficiente en entornos donde solo un cliente necesita interactuar con un servidor dedicado.

Durante la década de 1990, con la expansión del internet, el modelo Cliente – Servidor se convirtió en el estándar para aplicaciones web y bases de datos en línea. Aunque en estos sistemas la arquitectura se diseñó para soportar múltiples clientes, también se mantuvieron aplicaciones Cliente – Servidor dedicadas, en las que un único cliente establecía conexión directa con un servidor sin compartirlo con otros clientes. Este tipo de implementación se utilizó en sistemas de monitoreo, aplicaciones industriales y ciertos dispositivos de comunicación remota.

En la actualidad, aunque han surgido nuevas arquitecturas como los microservicios y la computación en la nube, el modelo Cliente – Servidor sigue siendo ampliamente utilizado, tanto en aplicaciones con múltiples clientes como en configuraciones específicas de un solo cliente y servidor. Este tipo de implementación es común en sistemas donde la seguridad, la estabilidad y la administración centralizada son prioritarias, como en cajeros automáticos, sistemas de control de acceso y servidores de autenticación dedicados.

A pesar de los avances en la descentralización de sistemas con arquitecturas como Peer-to-Peer (P2P) o la computación distribuida, el modelo Cliente – Servidor en su configuración de un solo cliente y servidor sigue siendo una solución robusta para aplicaciones que requieren una conexión estable, controlada y sin interferencia de múltiples usuarios. Su relevancia radica en la capacidad de garantizar una comunicación confiable entre cliente y servidor sin la complejidad adicional de administrar múltiples conexiones simultaneas

PLANTEAMIENTO DEL PROBLEMA

En el desarrollo de sistemas distribuidos, el modelo Cliente – Servidor es una de las arquitecturas mas utilizadas para la comunicación entre dispositivos, permitiendo la distribución de procesos y la centralización de datos. Sin embargo, su implementación puede variar según los requisitos de la aplicación, existiendo casos en los que se requiere que un único cliente se conecte a un solo servidor en lugar de permitir múltiples conexiones simultaneas.

Bajo esta premisa, surge la necesidad de desarrollar un programa que implemente el modelo Cliente – Servidor en una configuración de un solo cliente y servidor, con el objetivo de analizar como se lleva a cabo la comunicación entre ambos. Esta implementación permitirá comprender aspectos clave del modelo, como la transmisión de datos, la gestión de la conexión y la respuesta a solicitudes del cliente.

Dado que muchas aplicaciones en la actualidad dependen de este modelo, es fundamental estudiar su funcionamiento en un entorno controlado, asegurando que el servidor pueda manejar solicitudes y enviar respuestas sin la complejidad de múltiples conexiones simultaneas. El problema radica en la necesidad de diseñar e implementar un sistema que permita simular la interacción entre un cliente y un servidor de manera eficiente, clara y estructurada.

PROPUESTA DE SOLUCIÓN

Para abordar este problema, se propone el desarrollo de un sistema basado en el modelo Cliente – Servidor, implementado en Java y utilizando sockets TCP para la comunicación. Se diseñará un simulador de cajero automático (ATM) donde un cliente podrá conectarse a un servidor para realizar operaciones bancarias básicas.

El servidor se encargará de administrar cuentas bancarias, validar la autenticación del usuario y procesar solicitudes como consulta de saldo, depósitos y retiros. El cliente actuará como la interfaz a través de la cual el usuario podrá interactuar con su cuenta.

Este sistema permitirá demostrar el funcionamiento de un modelo Cliente – Servidor de conexión exclusiva, es decir, donde solo un cliente puede conectarse al servidor a la vez. A través de esta implementación, se analizará el manejo de conexiones, la gestión de solicitudes y el flujo de información entre ambos componentes.

MATERIALES Y METODOS

Para la implementación de este sistema, se hará uso de los siguientes recursos:

- ✓ Lenguaje de programación: Java
- ✓ Entorno de desarrollo: NetBeans IDE 21
- ✓ Librerías utilizadas:
  - Java.net: Liberia implementada para la comunicación mediante sockets TCP
  - Java.io: Librería implementada para la manipulación de flujos de entrada y salida de datos
  - Java.util: Librería implementada para la gestión de estructuras de datos como HashMap
- ✓ Protocolo de comunicación: TCP (Transmission Control Protocol), que garantiza una conexión estable entre el cliente y servidor
- ✓ Sistema operativo: Windows 11

DESARROLLO DE SOLUCIÓN

Para implementar el modelo Cliente – Servidor en un entorno de conexión exclusiva, se desarrollo un sistema que simula el funcionamiento de un cajero automático (ATM), donde un cliente se conecta a un servidor bancarios, ingresa sus credenciales y realiza operaciones bancarias básicas como consulta de saldo, deposito y retiro de dinero.

La implementación del sistema está compuesta por tres módulos principales: Servidor, Cliente, Cuenta Bancaria. Cada uno de estos módulos tiene funciones especificas dentro del modelo Cliente – Servidor, permitiendo la interacción entre el usuario y la base de datos del servidor.

Clase Servidor

Es servidor es el componente principal del sistema, ya que gestiona las cuentas bancarias y responde las solicitudes del cliente. Para lograr esto, implemente una clase que utiliza un ServerSocket en el puerto 3000, permitiendo la espera de conexiones entrantes.

Cuando un cliente se conecta, el servidor solicita sus credenciales (numero de cuenta y NIP) y verifica si los datos ingresados corresponden a una cuenta valida. En caso de autenticación exitosa, el servidor habilita la comunicación, permitiendo que el cliente realice operaciones como consulta de salgo, deposito y retiro de dinero.

El servidor gestiona las cuentas bancarias mediante un HashMap, donde cada cuenta esta asociada a un numero de cuenta único. Una vez que el cliente se autentica, el servidor procesa cada solicitud y envía la respuesta correspondiente. Además, maneja la desconexión del cliente de manera controlada, asegurando que el sistema continue funcionando correctamente después de la salida del usuario.

En el siguiente fragmento de código, el servidor inicializa la conexión y maneja la autenticación del usuario.

```
ServerSocket ss = new ServerSocket(3000);
System.out.println("Servidor bancario iniciado en el puerto 3000...");
while (true) {
    Socket cl = ss.accept();
    System.out.println("Cliente conectado.");
    try{
        BufferedReader ent = new BufferedReader(new InputStreamReader(cl.getInputStream()));
        PrintWriter sal = new PrintWriter(cl.getOutputStream(), true);
        //Recibiendo numero de cuenta
        numCuenta = ent.readLine();
        //Recibiendo nip
        nip = Integer.parseInt(ent.readLine());
        //Validando numero de cuenta y nip
        if(cuentas.containsKey(numCuenta)){
            CuentaBancaria cuenta = cuentas.get(numCuenta);
            if(cuenta.autenticar(nip)){
                sal.println("Bienvenido "+cuenta.getTitular());
                sal.flush();
```

EL fragmento de código que se ejecuta si las credenciales ingresadas por el cliente son correctas, es el siguiente:

```
while(true){
    operacion = ent.readLine();
    switch(operacion){
        case "C": sal.println("Saldo actual: $"+cuenta.consultarSaldo());
                  sal.flush();
                  break;
        case "D": montoD = Double.parseDouble(ent.readLine());
                  cuenta.depositar(montoD);
                  sal.println("Deposito exitoso. Nuevo saldo: $"+cuenta.consultarSaldo());
                  sal.flush();
                  break;
        case "R": montoR = Double.parseDouble(ent.readLine());
                  if(montoR>cuenta.consultarSaldo()){
                      sal.println("La cuenta no cuenta con los fondos necesarios.\nSaldo: $"+cuenta.consultarSaldo());
                  sal.flush();
                  }else{
                    cuenta.retirar(montoR);
                    sal.println("Retiro exitoso. Nuevo saldo: $"+cuenta.consultarSaldo());
                    sal.flush();
                  }
                  break;
        case "S": sal.println("Sesión finalizada.");
                  sal.flush();
                  break;
    }
    if(operacion.equals("S")){
        break;
    }
}
```

Clase ATM

El cliente es el módulo encargado de enviar solicitudes al servidor y recibir respuestas. Para lograr la comunicación, el cliente establece una conexión con el servidor mediante un Socket y enviar la información necesaria para la autenticación del usuario.

Después de que el servidor valida las credenciales, el cliente muestra un menú interactivo con opciones para consultar saldo, realizar depósitos y retirar dinero. Dependiendo de la selección del usuario, el cliente envía una solicitud especifica al servidor, recibe la respuesta correspondiente y la muestra en pantalla.

Además, el cliente maneja la desconexión segura, permitiendo que el usuario finalice la sesión correctamente sin interrumpir el funcionamiento del sistema.

A continuación, se muestra el fragmento de código donde el cliente establece la conexión y gestiona la autenticación:

```
Socket cl = new Socket("localhost",3000);
BufferedReader ent = new BufferedReader(new InputStreamReader(cl.getInputStream()));
PrintWriter sal = new PrintWriter(cl.getOutputStream(), true);
Scanner scan = new Scanner(System.in);
//Solicitando numero de cuenta
System.out.print("Ingrese su numero de cuenta: ");
String numCuenta = scan.nextLine();
//Solicitando NIP
System.out.print("Ingrese su NIP: ");
nip = scan.nextInt();
//Enviar info al servidor
sal.println(numCuenta);
sal.println(nip);
sal.flush();
//Recibiendo respueta de servidor
String autenticacion = ent.readLine();
System.out.println("\n"+autenticacion);
```

En el siguiente fragmento de código se muestra el menú que se muestra en caso de que las credenciales de autenticación sean correctas:

```
while(true){
    if(opc == 4){
        break;
    }
    System.out.println("Seleccione la opcion deseada:");
    System.out.println("1. Consultar saldo");
    System.out.println("2. Depositar dinero");
    System.out.println("3. Retirar dinero");
    System.out.println("4. Salir");
    opc = scan.nextInt();

    switch(opc){
        case 1: sal.println("C");
                sal.flush();
                System.out.print("\n");
                break;
        case 2: System.out.print("\nIngrese el monto a depositar: $");
                montoD = scan.nextInt();
                sal.println("D");
                sal.println(montoD);
                sal.flush();
                break;
        case 3: System.out.print("Ingrese el monto a retirar: $");
                montoR = scan.nextInt();
                sal.println("R");
                sal.println(montoR);
                sal.flush();
                break;
        case 4: sal.println("S");
                sal.flush();
                break;
    }
    System.out.println(ent.readLine());
}
```

Clase CuentaBancaria

Esta clase permite gestionar las cuentas bancarias, a través del almacenamiento de los datos de cada usuario y el uso de diversos métodos para realizar las operaciones financieras de manera segura.

Cada cuenta bancaria tiene los siguientes atributos:

- ✓ Número de cuenta
- ✓ Nombre del titular
- ✓ Saldo disponible
- ✓ NIP de acceso

Los métodos implementados en dicha clase son los siguientes:

```
public CuentaBancaria(String numeroCuenta, String titular, double saldo, int nip){
    this.numeroCuenta = numeroCuenta;
    this.titular = titular;
    this.saldo = saldo;
    this.nip = nip;
}
```

Este método es el constructor que se encarga de inicializar los datos de la cuenta

```
public String getTitular(){
    return titular;
}
```

Este método se encarga de devolver el nombre del titular de la cuenta

```
public double consultarSaldo(){
    return saldo;
}
```

Este método retorna el saldo actual de la cuenta

```
public void depositar(double monto){
    if(monto > 0){
        saldo += monto;
        System.out.println("Deposito exitoso. Nuevo saldo $"+saldo);
    }
}
```

Este método aumenta el saldo de la cuenta si el monto ingresado es valido

```java
public void retirar(double monto){
    if(monto < saldo){
        saldo -= monto;
        System.out.println("Retiro exitoso. Nuevo saldo $"+saldo);
    }else{
        System.out.println("La cantidad debe ser menor al saldo total de la cuenta.");
    }
}
```

Este método disminuye el saldo de la cuenta si hay suficientes fondos disponibles

```java
public boolean autenticar(int claveIngresada) {
    return nip==claveIngresada;
}
```

Este método valida que el NIP ingresado sea el correcto para la cuenta

RESULTADOS

```
Servidor bancario iniciado en el puerto 3000...
Cliente conectado.
Cliente desconectado.
```

Al ejecutar el código del servidor, este se inicializa y queda en espera de conexiones de clientes en el puerto 3000. Cuando un cliente se conecta al servidor, este lo notifica por medio de un mensaje en pantalla, al igual que cuando un cliente se desconecta.

```
Ingrese su numero de cuenta: 1234567890
Ingrese su NIP: 1235

NIP incorrecto.
```

Por parte del cliente, cuando este se ejecuta, solicita las credenciales de acceso para su autenticación. Si estas son incorrectas, el servidor de lo notifica al cliente y este lo muestra en pantalla el mensaje y se termina su sesión.

```
Ingrese su numero de cuenta: 1234567890
Ingrese su NIP: 1234

Bienvenido Juan Perez
Seleccione la opcion deseada:
1. Consultar saldo
2. Depositar dinero
3. Retirar dinero
4. Salir
```

Si las credenciales de autenticación son correctas, el sistema te permite el acceso al menú para poder realizar cualquiera de las operaciones que ofrece sobre la cuenta

```
1

Saldo actual: $5000.0
Seleccione la opcion deseada:
1. Consultar saldo
2. Depositar dinero
3. Retirar dinero
4. Salir
2

Ingrese el monto a depositar: $1500
Deposito exitoso. Nuevo saldo: $6500.0
Seleccione la opcion deseada:
1. Consultar saldo
2. Depositar dinero
3. Retirar dinero
4. Salir
3
Ingrese el monto a retirar: $100
Retiro exitoso. Nuevo saldo: $6400.0
Seleccione la opcion deseada:
1. Consultar saldo
2. Depositar dinero
3. Retirar dinero
4. Salir
4
Sesi◆n finalizada.
```

Si se ingresa la opción 1, nos muestra el saldo actual de la cuenta que nos entregó el servidor al realizar dicha solicitud. Para el caso de la opción 2, nos solicita la cantidad a depositar a la cuenta, para enviar dicha información al servidor y realizar la modificación del saldo disponible. Para el caso de la opción 3, nos solicita la cantidad a retirar, donde el servidor recibe dicha información y valida que el saldo de la cuenta sea menor al saldo de la cuenta. En caso de que así sea, se modifica el saldo de la cuenta y nos vuelve a mostrar el menú.

CONCLUSIONES

Después de realizar esta practica sobre el modelo Cliente – Servidor, me queda claro lo esencial que es este esquema en el desarrollo de sistemas distribuidos. Aunque en teoría parece un concepto bastante sencillo, su implementación me permitió entender como fluye la comunicación entre un cliente y un servidor, como se manejan las solicitudes y respuestas, y la importancia de una conexión estable para que todo funcione correctamente.

Uno de los aspectos mas interesantes fue como el servidor actúa como un centro de control, gestionando cuentas bancarias y verificando que las transacciones se realicen de forma segura. Mientras tanto, el cliente simplemente se encarga de enviar solicitudes y mostrar la información al usuario, lo que refuerza la idea de que cada parte tiene un rol bien definido dentro del sistema.

Otro aprendizaje clave fue la importancia de manejar bien las conexiones y la autenticación de los usuarios. Vi que un mal manejo de la entrada y salida de datos puede generar errores o incluso interrumpir la comunicación entre el modelo. Además, comprender el funcionamiento de los sockets TCP fue fundamental para que la comunicación fuera fluida y sin problemas.

Por último, considero que esta practica no solo me permitió ver en acción el modelo Cliente – Servidor, sino que también nos ayudo a reforzar conceptos como gestión de conexiones, autenticación, control de operaciones y estructura de software distribuido. Sin duda, este modelo sigue siendo una base solida para muchas aplicaciones modernas y su correcta implementación es clave para garantizar sistemas eficientes y confiables.