



CENTRO DE CIENCIAS BÁSICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
METAHEURÍSTICAS I
7° "A"

ACTIVIDAD 2.02: ARQUITECTURA CLIENTE/SERVIDOR

Profesor: Francisco Javier Luna Rosas

Alumnos:

Almeida Ortega Andrea Melissa
Espinoza Sánchez Joel Alejandro
Flores Fernández Óscar Alonso
Gómez Garza Dariana
González Arenas Fernando Francisco
Orocio García Hiram Efraín

Fecha de Entrega: Aguascalientes, Ags., 17 de septiembre de 2021

Actividad 2.02: Arquitectura Cliente/Servidor

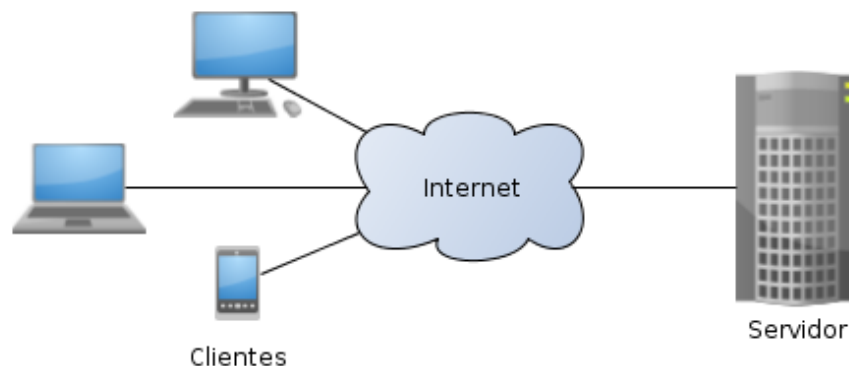
Antecedentes

La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

En esta arquitectura “la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema” (Berson, 1996).

Algunos ejemplos de aplicaciones que usen el modelo cliente-servidor son el Correo electrónico y un Servidor de impresión. La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.

Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.





La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un solo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

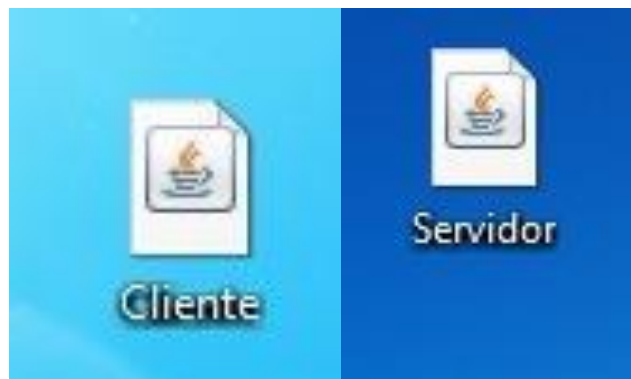
La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad, los archivos que son de uso público y los que son de uso restringido, los archivos que son de sólo lectura y los que, por el contrario, pueden ser modificados, etc. Este tipo de red puede utilizarse conjuntamente en caso de que se esté utilizando en una red mixta.

Funcionamiento de la arquitectura

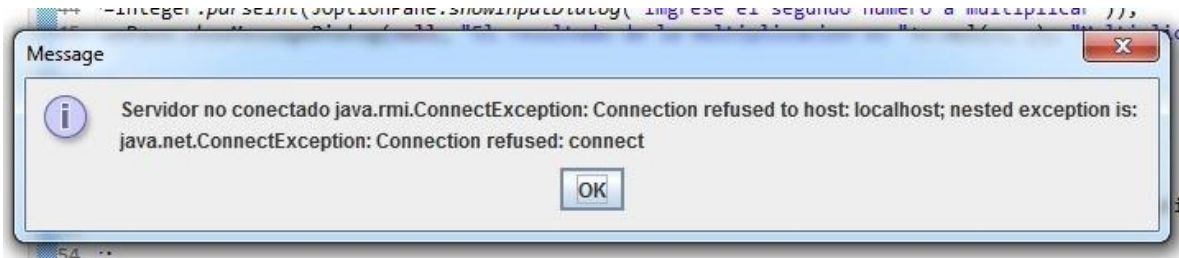
El programa tiene dos ejecutables .jar como se muestra a continuación:

 Cliente.jar	16/09/2021 11:02 p. m.	Executable Jar File
 Servidor.jar	16/09/2021 11:02 p. m.	Executable Jar File

El equipo optó por realizar la arquitectura en lenguaje Java:



Para la ejecución de la arquitectura será necesario primero iniciar el procedimiento desde el lado del servidor con su respectivo programa, ya que si se quiere ejecutar el archivo cliente sin ejecutar antes el servidor, pasa lo siguiente:



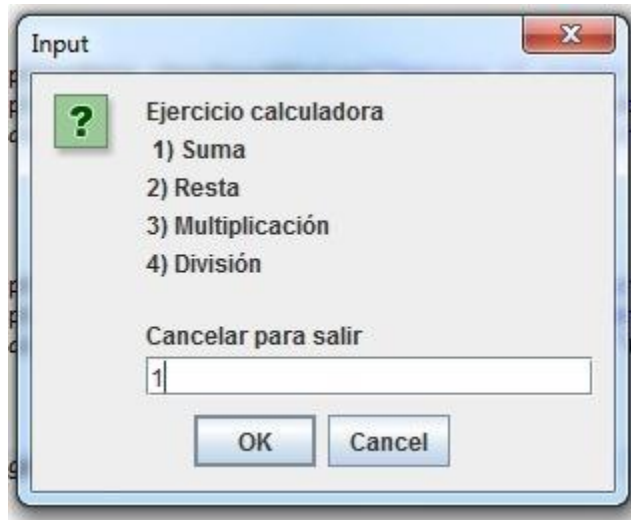
Al ejecutar primero el servidor, debe verse así:



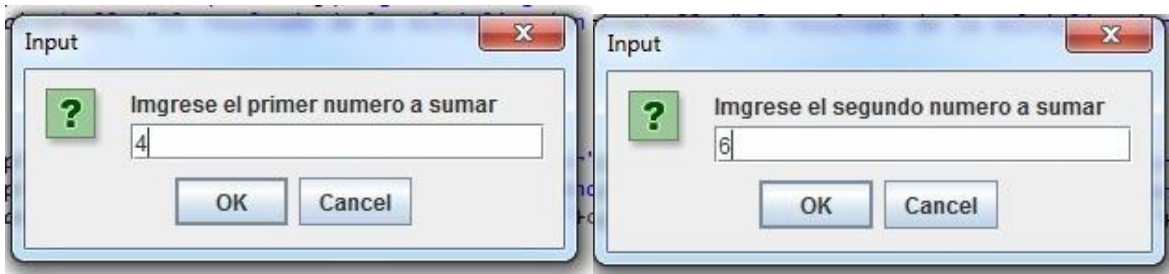
Finalmente se podrá abrir el cliente, que para probar el funcionamiento correcto del servidor, se probó rápidamente una calculadora:



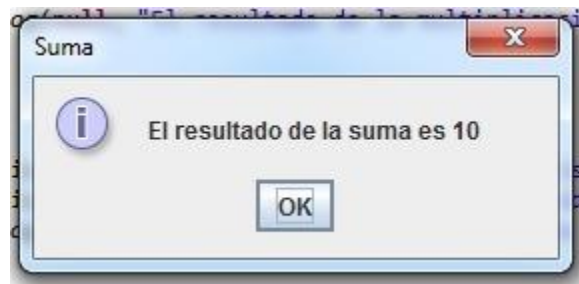
Donde su funcionamiento es lo de menos para la arquitectura cliente – servidor y todo este proceso está hecho desde el servidor:



Así se puede probar el funcionamiento de la calculadora ejecutado desde el servidor:



Y los resultados los dará desde el lado del servidor.



El canal de transmisión se podrá realizar bajo una red de área local dentro de las computadoras del equipo.

Conclusiones

Andrea Melissa Almeida Ortega: Como se puede ver, es de suma importancia la implementación de una arquitectura de cliente servidor cuando hablamos de proyectos grandes, ya que ayuda a que las tareas sean repartidas para poder saciar las peticiones de los clientes de forma eficiente.

Joel Alejandro Espinoza Sánchez: La implementación y uso de una arquitectura cliente – servidor será muy importante para el proyecto de la materia. Con esta arquitectura se puede conectar un conjunto de computadoras para la comunicación de ellas y obtener ventajas sobre información entre las computadoras.

Óscar Alonso Flores Fernández: Poder trabajar con una arquitectura cliente - servidor nos permite comprender el como hacer uso de diversas fuentes de recursos. En este caso el programa es sencillo y a una escala muy pequeña, pero escalar poco a poco con el fin de realmente mejorar la eficiencia en futuros proyectos de mayor tamaño será imperativo para ser capaces de cubrir las demandas de cada tarea llevada a cabo

Dariana Gómez Garza: Para realizar esta práctica se tuvo que buscar un poco más de información sobre redes ya que teníamos puro conocimiento teórico, nos dimos cuenta que esto nos serviría para agilizar nuestro trabajo que estamos realizando para el proyecto de esta materia.

También visualizamos aspectos importantes y que no habíamos tomado en cuenta antes y que son fundamentales para la práctica

Fernando Francisco González Arenas: Usar un servidor permite utilidades en las que podemos comunicar las computadoras fácilmente. En este trabajo podemos observar cómo comunicar nuestras computadoras a partir de un programa en Java y tener nuestras computadoras conectadas.

Hiram Efraín Orocio García: Un servidor es útil al momento de hacer uso de recursos, datos, servicios entre otras cosas así que aprender a implementar un así sea en una aplicación básica como el uso de una calculadora, en un futuro lo

podríamos implementar en problemas mayores donde el compartir estos recursos, serán de gran ayuda.

Referencias bibliográficas

1. Berson, A. (1996). *Client/Server Architecture*. Los Angeles: McGraw Hill.
2. Proper, H. (2017). *Architectural Coordination of Enterprise Transformation*. London: Springer.

Anexos

Anexo 1: Código de cliente.java en Java:

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
import javax.swing.JOptionPane;

public class cliente {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner sc = new Scanner(System.in);
        try
        {
            Registry miRegistro=
LocateRegistry.getRegistry("localhost",1099);
            calculadora c= (calculadora)
Naming.lookup("//localhost/calculadora");

            while(true)
            {
                String
menu=JOptionPane.showInputDialog("Ejercicio calculadora \n "+
                "1) Suma\n"+
                "2) Resta\n"+
                "3) Multiplicación\n"+
                "4) División\n\n"+
                "Cancelar para salir");
                switch(menu)
                {
                    case "1":
                    {
                        int
x=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el primer
numero a sumar"));
                        int
y=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el segundo
numero a sumar"));
                        JOptionPane.showMessageDialog(null, "El
resultado de la suma es "+c.sum(x, y),
"Suma",JOptionPane.INFORMATION_MESSAGE);
                    }
                    break;
                }
            }
        }
    }
}
```

```

        case "2":
        {
            int
x=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el primer
numero a restar"));
            int
y=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el segundo
numero a resar"));
            JOptionPane.showMessageDialog(null, "El
resultado de la resta es "+c.res(x, y),
"Resta",JOptionPane.INFORMATION_MESSAGE);
        }
        break;
        case "3":
        {
            int
x=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el primer
numero a multiplicar"));
            int
y=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el segundo
numero a multiplicar"));
            JOptionPane.showMessageDialog(null, "El
resultado de la multiplicacion es "+c.mul(x, y),
"Multiplicación",JOptionPane.INFORMATION_MESSAGE);
        }
        break;
        case "4":
        {
            int
x=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el
divisor"));
            int
y=Integer.parseInt(JOptionPane.showInputDialog("Imgrese el
dividendo"));
            JOptionPane.showMessageDialog(null, "El
resultado de la division es "+c.div(x, y),
"División",JOptionPane.INFORMATION_MESSAGE);
        }
        break;
        default:

JOptionPane.showMessageDialog(null, "opcion invalida");
        break;

    }
}

```

```
        }catch(Exception e) {
            JOptionPane.showMessageDialog(null, "Servidor no
conectado "+e);
        }
    }
}
```

Anexo 2: Código de servidor.java en Java:

```
import java.rmi.registry.Registry;
import javax.swing.JOptionPane;

public class servidor {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
            Registry r =
java.rmi.registry.LocateRegistry.createRegistry(1099);
            r.rebind("calculadora", new rmi());
            JOptionPane.showMessageDialog(null, "Servidor
conectado");
        }catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Servidor no
conectado "+e);
        }
    }
}
```

Anexo 3: Código de rmi.java en Java:

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class rmi extends UnicastRemoteObject implements
calculadora {
    public rmi() throws RemoteException{
        int a,b;
    }

    @Override
    public int div(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a/b;
    }

    @Override
    public int mul(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a*b;
    }

    @Override
    public int res(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a-b;
    }

    @Override
    public int sum(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a+b;
    }
}
```

Anexo 4: Código de calculadora.java en Java:

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface calculadora extends Remote{  
    public int div(int a, int b) throws RemoteException;  
    public int mul(int a, int b) throws RemoteException;  
    public int res(int a, int b) throws RemoteException;  
    public int sum(int a, int b) throws RemoteException;  
  
}
```