PRACTICA 3

Irvyn xicale cabrera - 201963582

Cambios previos de la practica anterior

Realice algunas correcciones al código anterior como lo es:

- Variables para almacenar las ip de los demás equipos

```
public InetAddress[] ipPc = new InetAddress[1];// ip de las pc de los
compañeros

    public VerArchivos() {
        try {
            // Agregar direcciones IP a tu arreglo
            ipPc[0] = InetAddress.getByName("192.168.100.95");
            //ipPc[0] = InetAddress.getByName("192.168.100.14");

            // descomentar segun el numero de usuarios
            /*
            ipPc[1] = InetAddress.getByName("192.168.137.54");
            ipPc[2] = InetAddress.getByName("192.168.137.101");
            ipPc[3] = InetAddress.getByName("192.168.137.10");
             */
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
        this.archivo = new ArrayList<>();
        cargarArchivo(); // Cargar la lista de archivos
    }
```

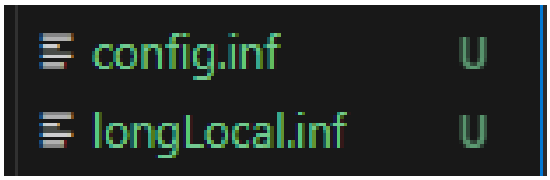
- El socket nos enviara los mensajes a la clase middleware

```
public void run() {
        try {
            while (true) {
                // recibimos el mensaje en el servidor
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
                socket.receive(receivePacket);

                String message = new String(receivePacket.getData(), 0,
receivePacket.getLength());
                System.out.println();
                middleware.procesarMensaje(message, receivePacket);
            }
        } catch (IOException e) {
            e.printStackTrace();
```

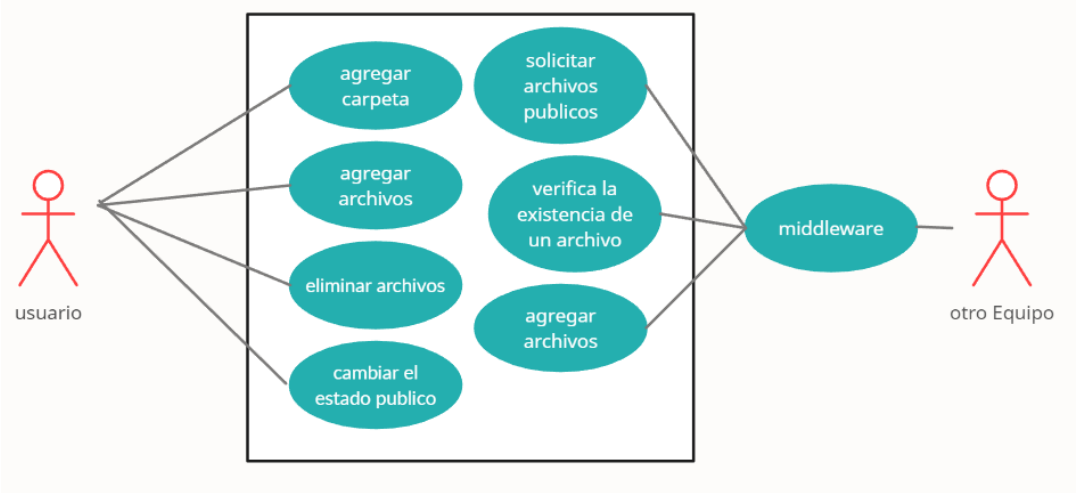
```
        }
    }
```

- Separación del config.inf en longLocal.inf y config.inf debido a una confusión estaban en el mismo archivo



PLANTEACION

Decidimos no confiar en nadie y optamos por crear un archivo global en cada servidor. Aunque esta elección inicialmente genera el envío de paquetes pesados para compartir los archivos con cada servidor, posteriormente, con la creación de las listas globales, el tamaño de estos paquetes disminuye de manera significativa. Esto se debe a que la actualización de los archivos en cada servidor se hace de forma incremental, lo que significa que en lugar de enviar todos los archivos nuevamente cada vez que se realiza una actualización o cambio, solo se transmiten los archivos que han sido modificados o agregados desde la última actualización. Este enfoque descentralizado y autosuficiente garantiza la independencia de cada servidor y mejora la eficiencia del sistema en su conjunto.

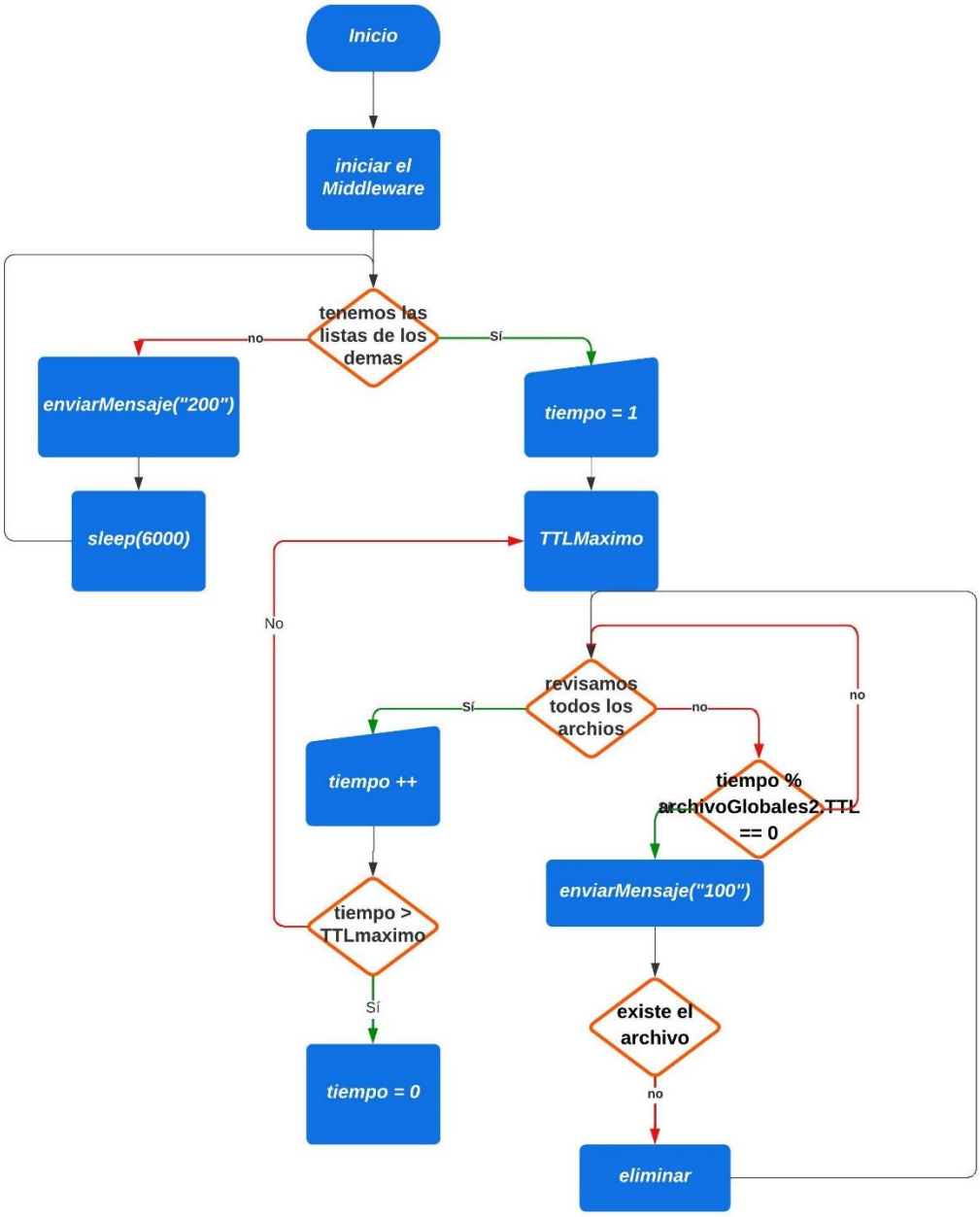


Decidimos establecer un estándar claro para el intercambio de mensajes a través de nuestro middleware con el fin de facilitar la comunicación y comprensión entre los diferentes servidores y equipos. Para lograr esto, definimos códigos específicos que indican la naturaleza de la solicitud y su contenido. Estos códigos se componen de números enteros y se estructuran de la siguiente manera:

- Código 100, nombre.extension: Este código se utiliza para verificar la existencia de un archivo en el servidor local. Cuando se recibe una solicitud con este código, el servidor verifica si el archivo especificado por "nombre.extension" existe en su lista de archivos.

- Código 101, nombre.extension: Si el archivo existe en el servidor local o se encuentra en la lista global y otro equipo lo tiene, se devuelve una respuesta autoritativa con este código. Esto indica que el servidor tiene conocimiento del archivo y está autorizado para proporcionar información sobre él.
- Código 102: En caso de que el archivo no exista en ningún servidor, se envía este código como respuesta negativa para indicar que el archivo solicitado no está disponible en la red.
- Código 200: Este código se utiliza para solicitar la lista local de archivos de otro equipo. Cuando un servidor recibe una solicitud con este código, envía su lista de archivos locales como respuesta.
- Código 201, nombre.extension, nombre.extension, ...: La lista de archivos locales se estructura utilizando este código, seguido de los nombres de archivo y extensiones separados por comas. Esto permite a otros servidores conocer los archivos disponibles en el equipo solicitante.
- Código 300, nombre.extension: Si deseamos agregar un nuevo archivo a la lista global, utilizamos este código junto con el nombre y la extensión del archivo. De esta manera, podemos compartir nuevos archivos con otros servidores y equipos en la red.

Al adoptar esta estructura de códigos y mensajes, hemos simplificado y estandarizado nuestras comunicaciones, lo que facilita la implementación y el entendimiento del middleware en todos los sistemas involucrados. Esto nos permite avanzar de manera eficiente en el desarrollo y la expansión de nuestra red de archivos distribuidos.



Codigos

Actualizarmiddle.java

La clase actualizarMiddle es un hilo que forma parte de nuestra clase middleware en un sistema de directorio distribuido. Su función principal es mantener actualizados los archivos en función de su tiempo de vida (TTL)

- Inicialización: El hilo actualizarMiddle se inicia con un objeto UDP como parámetro, lo que le permite comunicarse con nuestro servidor.
- Cálculo del TTL Máximo: En cada ciclo del bucle, se calcula el TTL máximo entre todos los archivos en la lista. Esto es importante porque nos aseguramos de que el tiempo no supere el TTL máximo para evitar errores causados por números demasiado altos.
- Incremento de Tiempo: El tiempo se incrementa en 1 milisegundo en cada iteración. Esto se hace para mantener un seguimiento preciso del tiempo y garantizar que se cumpla el TTL de los archivos correctamente.
- Verificación del Tiempo y TTL: Para cada archivo en la lista, se verifica si su TTL es diferente de cero. Si el TTL es mayor que cero, se verifica si el tiempo actual es un múltiplo exacto del TTL del archivo. Esto se hace verificando si el cociente entre el tiempo y el TTL es un número entero. Si es así, se envía un mensaje al servidor correspondiente para actualizar el archivo.
- Espera de 1 Milisegundo: Después de cada ciclo del bucle, se introduce una pausa de 1 milisegundo para controlar el incremento de tiempo y evitar un consumo excesivo de recursos.

```
package DNS;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;

class ActualizarMiddle extends Thread{
    private ArrayList<ArchivoGlobales> archivoGlobales;
    private UDP servidor;
    private int tiempo = 0;
    private int ttlMaximo = 0;
    public int estadoPeticion = 0; // 0 = ninguna respuesta, 1 = respuesta
afirmativa, 2 = respuesta negativa


    // Constructor que acepta UDP y ArrayList<ArchivoGlobales>
    public ActualizarMiddle(UDP serv, ArrayList<ArchivoGlobales> archivos) {
        servidor = serv;
        archivoGlobales = archivos;
    }

    @Override
    public void run() {
        while (true) {
            ttlMaximo = encontrarTTLMaximo();
```

```
            // Incrementa el tiempo en 1 milisegundo en cada iteración
            tiempo = tiempo + 1;

            // Si el tiempo supera el TTL máximo, reinicia el tiempo para
evitar errores por numeros altos
            if (tiempo > ttlMaximo + 1) {
                tiempo = 0;
            }

            Iterator<ArchivoGlobales> iterator = archivoGlobales.iterator();
            while (iterator.hasNext()) {
                ArchivoGlobales archivoGlobales2 = iterator.next();
                if (archivoGlobales2.TTL != 0) {
                    // Verifica si el cociente entre tiempo y TTL es un
número entero ejmplo 1000/1000 = 1 1500/1000 = 1.5
                    if (tiempo % archivoGlobales2.TTL == 0) {
                        System.out.println("verificando " +
archivoGlobales2.nombre + "." + archivoGlobales2.extension);
                        estadoPeticion = 0;
                        servidor.enviarMensaje("100," +
archivoGlobales2.nombre + "." + archivoGlobales2.extension,
archivoGlobales2.IP, 5000);

                        // Esperamos hasta recibir la respuesta o hasta que
pase un tiempo máximo
                        long tiempoInicial = System.currentTimeMillis();
                        long tiempoMaximoEspera = 8000; // Tiempo máximo de
espera en milisegundos (30 segundos)

                        while (estadoPeticion == 0 &&
System.currentTimeMillis() - tiempoInicial < tiempoMaximoEspera) {
                            try {
                                Thread.sleep(10); // Pausa de 10
milisegundos (puedes ajustar el valor según tus necesidades)
                            } catch (InterruptedException e) {
                                // Manejo de la excepción, si es necesario
                            }
                        }

                        // Verificamos el estado y eliminamos si es
necesario
                        if (estadoPeticion == 1) {
```

```
                            System.out.println("archivo: " +
archivoGlobales2.nombre + "." + archivoGlobales2.extension + "   ---
>   verificado");
                        } else if (estadoPeticion == 2) {
                            System.out.println("archivo: " +
archivoGlobales2.nombre + "." + archivoGlobales2.extension + "   --->   fue
eliminado");
                            iterator.remove(); // Eliminamos el elemento
usando el iterador
                            System.out.println("Se eliminó el archivo
global: " + archivoGlobales2.nombre + "." + archivoGlobales2.extension);
                        }
                    }
                }
            }



            try {
                Thread.sleep(1); // Espera 1 milisegundo antes de la
siguiente iteración
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }


    public void vincularArchivos(ArrayList<ArchivoGlobales> original){
        archivoGlobales = original;
    }

    private int encontrarTTLMaximo() {
        int maximo = 0;
        for (ArchivoGlobales archivo : archivoGlobales) {
            if (archivo.TTL > maximo) {
                maximo = archivo.TTL;
            }
        }
        return maximo;
    }
}
```

Este hilo es el cuerpo de nuestro middleware ya que será el encargado de verificar y actualizar nuestra lista verificando la integridad de los archivos en los demás servidores, pero para que funcione necesitaremos hacer unas actividades previas que se explicaran acontinuacion.

Middleware.java

Este middleware actúa como un intermediario entre los servidores, facilitando la búsqueda, obtención y actualización de listas de archivos globales, además de proporcionar la capacidad de buscar archivos específicos y gestionar su disponibilidad en la red. A través de este middleware puedo coordinar y compartir información entre los demás servidores

- archivoGlobal: Es una lista que almacena objetos ArchivoGlobales, que representan archivos disponibles globalmente en la red.
- servidor: Una instancia de la clase UDP que se utiliza para la comunicación mediante UDP (User Datagram Protocol).
- archivosLocales: Una instancia de la clase VerArchivos que parece representar archivos locales en la máquina.
- listaObtenida: Un array de booleanos que se utiliza para controlar si se han obtenido las listas de archivos de otros servidores.
- actualizarTTL: Una instancia de la clase ActualizarMiddle que se utiliza para actualizar los tiempos de vida (TTL) de los archivos.
- archivoLong: Una cadena que representa la ruta de un archivo de registro.
- Constructor Middleware: El constructor de la clase Middleware toma una instancia de la clase UDP como parámetro y realiza varias inicializaciones, como vincular el servidor UDP y cargar información desde el archivo de registro.
- Método run (Ejecución Principal):
- El método run es un método que se ejecuta cuando se inicia un hilo (Thread) basado en esta clase Middleware.
- En este método, se implementa un ciclo while que espera a que se obtengan todas las listas de archivos de otros servidores. Se espera cada 6 segundos antes de volver a consultar. Se envían solicitudes (200) a otros servidores para obtener sus listas de archivos. Una vez que se han obtenido todas las listas, se inicia un proceso de actualización por TTL.
- Método procesarMensaje: Este método se utiliza para procesar mensajes recibidos por el middleware. Los mensajes pueden contener códigos y datos específicos. Aquí se realizan las siguientes acciones:
    - Se obtiene la dirección IP y el puerto del remitente del mensaje.
    - Se divide el mensaje en elementos separados por comas.
    - Se interpreta el primer elemento del mensaje como un código numérico.
    - Según el código, se realiza una acción específica, como buscar archivos, enviar listas locales o agregar archivos a la lista global.
- Método enviarMensaje: Este método se utiliza para enviar mensajes a través del servidor UDP.
- Método vincularArchivos: Este método se utiliza para vincular los archivos locales a la instancia de Middleware.

- Método archivoExiste: Este método verifica si un archivo específico existe en la lista global o en otros servidores mediante una comunicación de red. En función de la respuesta, se actualiza la lista global o se elimina el archivo si no se encuentra.
- Método agregarArchivoGlobal: Se utiliza para agregar un archivo a la lista global.
- Método eliminarArchivoGlobal: Se utiliza para eliminar un archivo de la lista global.
- Método nuevoArchivoLocal: Se utiliza para agregar un nuevo archivo local y propagarlo a otros servidores si se han obtenido todas las listas.

```
package DNS;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Middleware extends Thread{
    private ArrayList<ArchivoGlobales> archivoGlobal = new
ArrayList<>(); // un array donde guargamos cada archivo
    private UDP servidor;
    private VerArchivos archivosLocales;
    public boolean[] listaObtenida = new boolean[1]; // controla si
obtuvimos las listas (se inicializa en false)
    private ActualizarMiddle actualizarTTL;
    private String archivoLong = System.getProperty("user.dir") +
"\\DNS\\longGlobal.inf";;

    Middleware(UDP serv) {
        servidor = serv;
        actualizarTTL = new ActualizarMiddle(servidor, archivoGlobal);
        actualizarTTL.vincularArchivos(archivoGlobal);
        cargarLong();
    }

    @Override
    public void run() {
        // obtenemos las listas de los demas usuarios
        System.out.println("iniciando middleware");
```

```
        while (listaObtenida[0] == false /*|| listaObtenida[1] == false || listaObtenida[2] == false || listaObtenida[3] == false*/) {
            try {
                sleep(6000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            for (int i = 0; i < listaObtenida.length; i++) {
                if (!listaObtenida[i]) {
                    System.out.println("pidiendo listas de los demás equipos");
                    // Solicitamos la lista de la otra pc
                    servidor.enviarMensaje("200", archivosLocales.ipPc[i], 5000);

                    // Esperamos hasta recibir la respuesta o hasta que pase un tiempo máximo
                    long tiempoInicial = System.currentTimeMillis();
                    long tiempoMaximoEspera = 8000; // Tiempo máximo de espera en milisegundos (30 segundos)

                    while (!listaObtenida[i] && System.currentTimeMillis() - tiempoInicial < tiempoMaximoEspera) {
                        // Aquí puedes agregar una espera corta (por ejemplo, sleep) para evitar un bucle de CPU al esperar.
                        // Pero ten en cuenta que podría aumentar el tiempo total de espera.
                    }
                }
            }
        }

        System.out.println("lista global obtenida");
        actualizarTTL.start();
    }

    public void procesarMensaje(String message, DatagramPacket receivePacket){
        InetAddress clientAddress = receivePacket.getAddress();
        int clientPort = receivePacket.getPort();
        System.out.println("Mensaje recibido en el servidor desde " + clientAddress + ", " + clientPort + ":" + message);
        String[] elementos = message.split(",");
        int primerNumero = 0;
```

```
        List<String> elementosRestantes = new ArrayList<String>();

        if (elementos.length > 0) {
            try {
                primerNumero = Integer.parseInt(elementos[0].trim());
// Intenta convertir el primer elemento en un entero

                // Si no hay errores en la conversión, el primer
elemento es un número
                System.out.println("codigo: " + primerNumero);

                for (int i = 1; i < elementos.length; i++) {
                    String[] elementos2 = elementos[i].split("\\.");
                    elementosRestantes.add(elementos2[0]); // Agrega
el nombre
                    elementosRestantes.add(elementos2[1]); // Agrega
la extensión
                }

                System.out.println("Elementos restantes: " +
elementosRestantes);
            } catch (NumberFormatException e) {
                // Si ocurre una excepción, el primer elemento no es
un número
                System.out.println("Primer elemento no es un número.
Valor del primer elemento: " + elementos[0].trim());

                // Puedes manejar esta excepción de acuerdo a tus
necesidades
            }
        } else {
            try {
                primerNumero = Integer.parseInt(message);
                System.out.println("codigo: " + primerNumero);

            } catch (NumberFormatException e) {
                System.out.println("Primer elemento no es un número.
Valor del primer elemento: " + message);
            }
        }

        switch (primerNumero) {
            case 100: // codigo de busqueda de un archivo
                String nombreArchivo = elementosRestantes.get(0);
                String extensionArchivo = elementosRestantes.get(1);
```

```
                if (archivosLocales.archivoExiste(nombreArchivo,
extensionArchivo) == true) {
                    String mensaje = "101," + nombreArchivo + "." +
extensionArchivo;
                    enviarMensaje(mensaje, clientAddress, clientPort);
// respuesta autoritativa de mi servidor
                }else{
                    if (archivoExiste(nombreArchivo,extensionArchivo)
== true){
                        String mensaje = "101," + nombreArchivo + "."
+ extensionArchivo; // respuesta autoritativa de otra maquina
                        enviarMensaje(mensaje, clientAddress,
clientPort);
                    }else{
                        enviarMensaje("102", clientAddress,
clientPort);// Nack el archivo no fue encontrado
                    }
                }
                break;
            case 101:
                actualizarTTL.estadoPeticion = 1;
                break;
            case 102:
                actualizarTTL.estadoPeticion = 2;
                break;
            case 200: // codigo para enviar la lista local
                enviarMensaje(("201," +
archivosLocales.obtenerArchivosPublicados()), clientAddress,
clientPort);
                break;

            case 201: // codigo para recibir la lista
                for (int i = 0; i < elementosRestantes.size(); i = i +
2) {
                    System.out.println("agregando " +
elementosRestantes.get(i) + "." + elementosRestantes.get(i+1));
                    agregarArchivoGlobal(elementosRestantes.get(i),
elementosRestantes.get(i+1), clientAddress);
                }
                for (int i = 0; i < archivosLocales.ipPc.length; i++)
{
                    if (archivosLocales.ipPc[i].equals(clientAddress))
{
                        listaObtenida[i] = true;
                    }
```

```
                }
                break;

            case 300: // codigo para agregar archivo a la lista global
                agregarArchivoGlobal(elementosRestantes.get(0),
elementosRestantes.get(1), clientAddress);
            default:
                break;
        }
    }

    public void enviarMensaje(String message, InetAddress
destinationAddress, int destinationPort){
        servidor.enviarMensaje(message, destinationAddress,
destinationPort);
    }

    public void cargarLong() {
        try (BufferedReader reader = new BufferedReader(new
FileReader(archivoLong))) {
            String line;
            while ((line = reader.readLine()) != null) {
                // Split de la línea usando comas
                String[] parts = line.split(",");

                // Verificamos que haya al menos 5 partes en la línea
antes de intentar crear un objeto ArchivoGlobales
                if (parts.length >= 5) {
                    // Parseamos la IP de la cadena de texto
                    InetAddress ip = InetAddress.getByName(parts[2]);

                    // Parseamos el TTL de la cadena de texto
                    int ttl = Integer.parseInt(parts[3]);

                    // Creamos un objeto ArchivoGlobales y lo
agregamos a la lista archivoGlobal
                    archivoGlobal.add(new ArchivoGlobales(parts[0],
parts[1], ip, ttl));
                    System.out.println("Agregando " + parts[0] + "." +
parts[1] + " - IP: " + ip + ", TTL: " + ttl);
                }
            }
        } catch (FileNotFoundException e) {
            System.err.println("El archivo de registro no existe. Se
creará uno nuevo.");
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void vincularArchivos(VerArchivos vincular){
        archivosLocales = vincular;
    }

    public boolean archivoExiste(String nombreArchivo, String
extensionArchivo) {
    boolean res = false;
    for (ArchivoGlobales archivo : archivoGlobal) {
        if (archivo.nombre.equals(nombreArchivo) &&
archivo.extension.equals(extensionArchivo)) {
            // Aquí se envía un mensaje y se espera una respuesta
            try (Socket socket = new Socket(archivo.IP, 5000)) {
                ObjectOutputStream outStream = new
ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream inStream = new
ObjectInputStream(socket.getInputStream());

                // Enviar un mensaje al servidor
                outStream.writeObject("100" + "," + nombreArchivo +
"." + extensionArchivo);

                // Esperar la respuesta del servidor
                String respuesta = (String) inStream.readObject();

                // Procesar la respuesta
                if (respuesta.equals("101" + "," + nombreArchivo + "."
+ extensionArchivo)) {
                    res = true;
                }else{
                    eliminarArchivoGlobal(nombreArchivo,
extensionArchivo);
                }
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
    return res;
    }

```

```
    public void agregarArchivoGlobal(String nombre, String extension, InetAddress ip) {
        // ArchivoGlobales archivo = new ArchivoGlobales(nombre, extension, ip, ttl);
        ArchivoGlobales archivo = new ArchivoGlobales(nombre, extension, ip, 5000);
        archivoGlobal.add(archivo);
    }

    public void eliminarArchivoGlobal(String nombre, String extension) {
        ArchivoGlobales archivoEliminar = null;

        for (ArchivoGlobales archivo : archivoGlobal) {
            if (archivo.nombre.equals(nombre) && archivo.extension.equals(extension)) {
                archivoEliminar = archivo;
                break; // Encontramos el archivo, salimos del bucle
            }
        }

        if (archivoEliminar != null) {
            archivoGlobal.remove(archivoEliminar);
        }
    }

    public void nuevoArchivoLocal(String nombre, String extension){
        for (int i = 0; i < archivosLocales.ipPc.length; i++) {
            if (listaObtenida[0] == true /*|| listaObtenida[1] == false || listaObtenida[2] == false || listaObtenida[3] == false*/) {
                enviarMensaje("300," + nombre + "." + extension, archivosLocales.ipPc[i], 5000);
            }
        }
    }
}
```