Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería

de la Computación

Aplicaciones para Comunicaciones en Red

Practica 01 "Servicio de transferencia de archivos"

Profesor: Axel Ernesto Moreno Cervantes

Grupo: 6CM1

Integrantes:

Ruvalcaba Flores Martha Catalina

Sandoval Hernández Eduardo

Fecha de entrega: 24/10/2022

Objetivo

El estudiante implementará una aplicación para el envío de múltiples archivos a través de la red haciendo uso de sockets de flujo.

Planteamiento del problema

En muchas organizaciones se requiere transferir grandes volúmenes de archivos a través de la red (archivos de audio, video, fotografías, archivos ejecutables, etc.) de una manera confiable, es decir, garantizar la entrega de los datos, que estos lleguen en orden y sin duplicados.

Introducción Teórica

El envío de archivos a través de la red es una característica importante para la gran mayoría de las aplicaciones que hoy día se utilizan (blogs, redes sociales, mensajería instantánea, Declaración de impuestos, educación en línea, etc.), sin embargo, no todas las aplicaciones disponibles permiten el envío de archivos de gran tamaño (p.e. El correo electrónico no permite enviar archivos de más de 10 o 20 MB). Esto hace necesario el desarrollo de aplicaciones que permitan transferir archivos sin importar el tamaño de éstos.

Los servicios de transferencia de archivos son servicios que permiten el intercambio de archivos de datos entre sistemas de cómputo. Dicho de otra forma, es el proceso de copiar y/o mover archivos de un ordenador a otro a través de una red, lo que permite compartir, transferir y/o transmitir estos archivos entre diferentes usuarios u ordenadores de manera local o remota [6].

TCP (Protocolo de Control de Transmisión, en inglés, “Transmission Control Protocol), es un protocolo de red diseñado por Vint Cerf y Bob Kahn el cual permite a dos hosts la posibilidad de conectarse para intercambiar flujos de datos. Este protocolo garantiza que los datos y paquetes se entreguen en el mismo orden en el que fueron enviados. También asegura que estos paquetes se entreguen de manera confiable y sin fallas [4].

Los Sockets son herramientas que ayudan a los procesos a comunicarse entre sí, los sockets de flujo proporcionan flujos de datos bidireccionales, fiables, secuenciados y sin duplicidad de datos sin límites de registro [5].

Desarrollo de la practica

En esta práctica se implementó un servicio de transferencia de archivos para que el cliente de la aplicación pueda enviar/recibir uno o más archivos de cualquier tamaño/tipo hacia/desde el servidor, además de la eliminación de archivos/carpetas en la ruta del servidor.

A partir de los programas “CEnvia” y “SRecibe” proporcionados por el profesor se realizaron las siguientes modificaciones:

1.- La aplicación cliente debe mostrar al usuario un menú que permita realizar las siguientes acciones: subir archivos/carpetas a la carpeta remota en el servidor, descargar archivos/carpetas hacia la carpeta local del usuario en el cliente y dar la posibilidad de eliminar archivos/carpetas de manera local/remota.

2.- El programa CEnvia implementa un cliente de flujo bloqueante que realiza el envío de un archivo elegido previamente a través de una caja de diálogo (JFileChooser) y éste es enviado utilizando flujos orientados a byte. Se modificó el código para que, en lugar de enviar un archivo, este programa permitiera enviar cualquier cantidad de archivos (secuencialmente) o carpetas.

3.- El programa SRecibe implementa un servidor de flujo bloqueante que recibe un archivo por cada conexión aceptada y éste es recibido usando flujos orientados a byte. Se modificó el código para que, en lugar de recibir un

solo archivo, éste sea capaz de recibir/enviar uno o más archivos o carpetas. Cada archivo se envía de manera individual.

4.- Se realizaron pruebas intentando enviar distintos tipos de archivo (imágenes, texto, ejecutables), así mismo se intentó enviar archivos de distintos tamaños (menos de 100KB, más de 100KB y menos de 10MB, más de 10MB y menos de 200MB, más de 200MB y hasta 2GB).

Estructura del programa

El código del programa consta de 4 clases las cuales son: Archivo, el cual servirá para manejar los archivos y cuenta con atributos de apoyo como lo son su nombre, tamaño, ruta y el archivo en sí, además de funciones para el manejo de estos atributos; Cliente, el cual maneja toda la lógica del lado del cliente como las funciones para comunicarse con el servidor para el envío y recepción de archivos/carpetas; Servidor, el cual maneja toda la lógica del lado del servidor y al igual que la clase Cliente contiene funciones para la comunicación con el cliente para el envío y recepción de archivos/carpetas; ClienteGUI, la cual sirve para la interfaz gráfica del cliente la cual le ayudará a seleccionar la acción que desee realizar y los archivos/carpetas que desee ya sea enviar, recibir o eliminar.

Simulación

En las siguientes figuras se muestra la interfaz del cliente junto a las funciones que puede realizar.

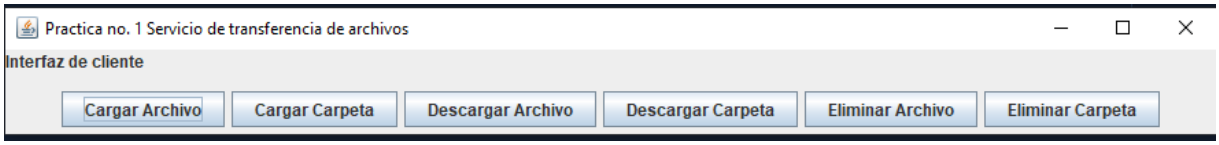

Figura 1. Interfaz principal del programa.

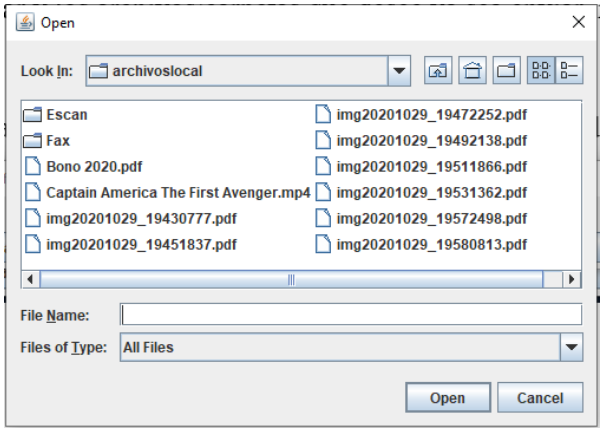

Figura 2. JFileChooser para la selección de archivos a enviar (solo admite selección de archivos).

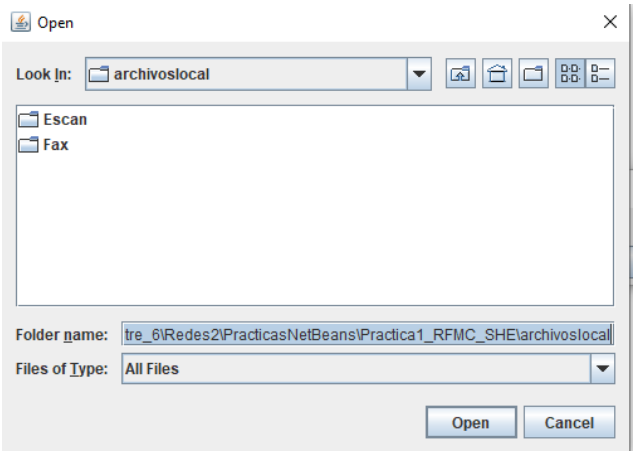

Figura 3. JFileChooser para la selección de carpetas a enviar (solo permite la selección de carpetas).

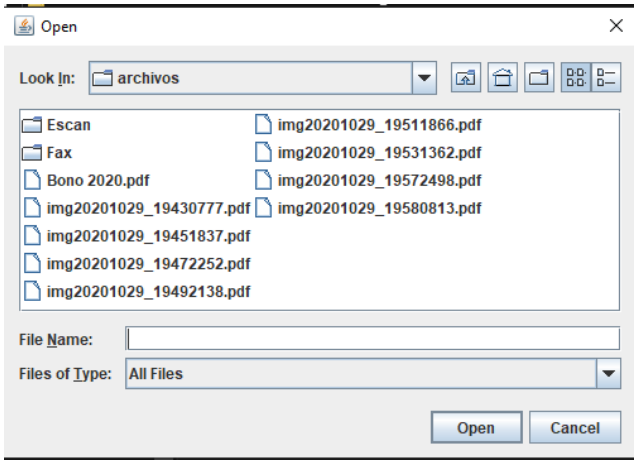

Figura 4. JFileChooser para la selección de archivos a descargar (servidor a cliente).

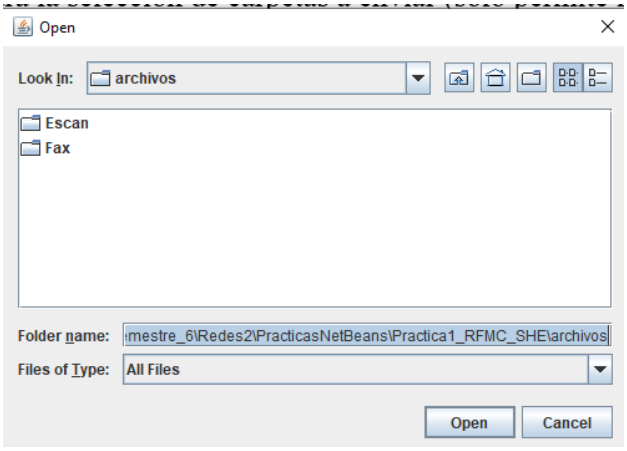

Figura 5. JFileChooser para la selección de carpetas a descargar (servidor a cliente).

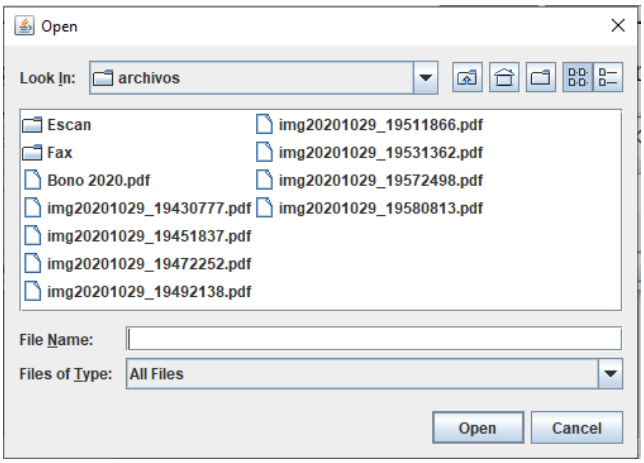

Figura 6. JFileChooser para la selección de archivos a eliminar (del servidor).

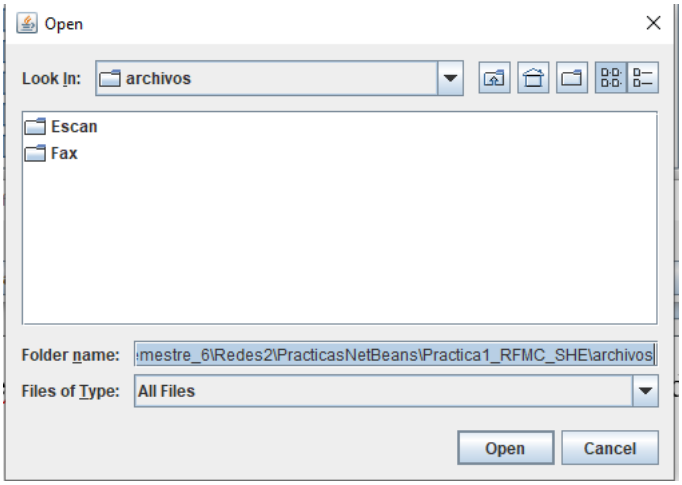

Figura 7. JFileChooser para la selección de carpetas a eliminar (del servidor).

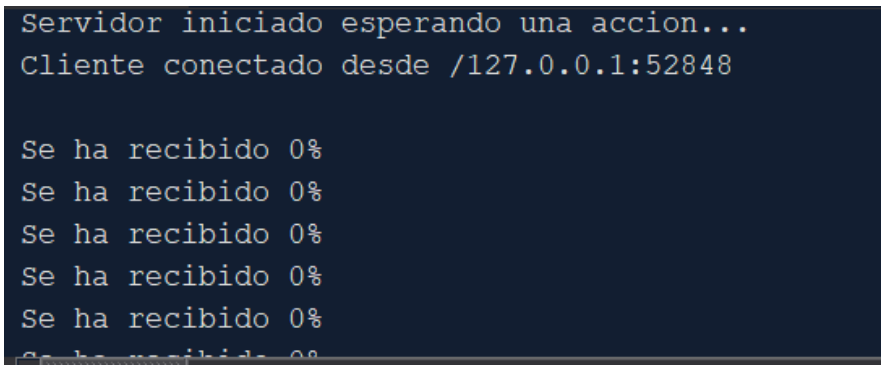

Figura 8. Servidor aceptando una conexión y recibiendo un archivo.

Figura 9. Cliente enviando un archivo.

Preguntas sobre el resultado final de la practica

¿Qué tipo de archivos se enviaron más rápido?

R: Archivos ligeros de cualquier formato (pdf, txt, etc.)

¿Cuál fue el número máximo de archivos que fue posible enviar a la vez?

R: Se realizó la prueba enviando 350 archivos con tamaños entre los 1 y 10 MB y estos pudieron enviarse sin problema, en el caso de las carpetas, con 64 carpetas la aplicación tuvo una ralentización considerable.

¿Cuál fue el tamaño de archivo más grande que se pudo transferir? ¿por qué?

R: Se realizó la prueba con un archivo de 2.2 GB, sin embargo, esta fue muy lenta tardando alrededor de 20 min. Se estima que el límite estaría fijado por la variable long que almacena el tamaño del archivo y la cual su máximo se sitúa en los $2^{63} - 1$ bits [1].

¿Qué es el orden de red?

R: Se sabe que en algunas aplicaciones que requieren de una comunicación confiable TCP son: el FTP, Telnet y HTTP, en dichas aplicaciones, el orden en que se envían y reciben los datos es crítico. Entonces TCP garantiza que los datos recibidos en un extremo y en el otro serán íntegros y si esto no llegase a ocurrir, entonces se reportaría un error.

¿Por qué razón es importante utilizar el orden de red al enviar los datos a través de un socket?

R: Es importante porque a través de un socket solo podemos enviar y recibir bytes. En Java, esta provee clases que permiten enmascarar este flujo de bytes como si fueran objetos, entonces permite leer y escribir objetos a través de la red. Entonces, en un esquema de cliente – servidor que utiliza TCP, cada una de las partes obtiene un socket, en el cual a través de este se podrá enviar y recibir datos hasta que se dé por finalizada la comunicación. Entonces, el socket es uno de los dos extremos en una comunicación de dos procesos a través de la red. De esta forma el servidor y el cliente se comunican a través del socket. Cabe mencionar que el socket direcciona unívocamente un proceso en toda la red, ya que incluye la dirección del host (ip) y la dirección del proceso (puerto).

Si deseáramos enviar archivos de tamaño muy grande, ¿qué cambios sería necesario hacer con respecto a los tipos de datos usados para medir el tamaño de los archivos, así como para leer bloques de datos del archivo?

R: Cambiar el tamaño de las variables asociadas al tamaño del archivo a enviar, así como el número de bytes que se envían por paquete.

Conclusiones Individuales

Ruvalcaba Flores Martha Catalina

A pesar de ser la primera práctica, esta tuvo un nivel de dificultad intermedio, puesto que la lógica de envío de archivos mediante el uso de sockets de flujo fue un reto para el equipo, sin embargo, la comunicación entre el servidor y el cliente fue relativamente sencillo. Para realizar la práctica se utilizó como base el programa proporcionado por el profesor, a partir de este se realizaron las modificaciones respectivas con el fin de cumplir con los objetivos de dichas prácticas. Al principio fue complicado entender el funcionamiento del socket, pero con el apoyo del profesor y la consulta del libro de "Java a Fondo" logré comprender mejor sobre estos temas, por ejemplo, el funcionamiento del TCP, el cual es un protocolo orientado a la conexión, el cual permite conectar dos aplicaciones de manera confiable. Así mismo, la definición de conceptos como los puertos, dirección ip, socket, las aplicaciones de un cliente y de un servidor, la serialización de objetos, el envío y recibo de bytes, entre otros temas vistos en clase. Se puede concluir que la practica fue un éxito, puesto que es funcional y cumple con el envío y descarga de archivos y carpetas desde un directorio local o remoto, la practica puede mejorar con respecto a su código, ya que no es tan eficiente, debido a que no se aprovecharon al máximo los recursos.

Sandoval Hernández Eduardo

El desarrollo de esta práctica resultó un tanto complicada debido a que se tenía poco conocimiento de los sockets de flujo y por tanto en cierto punto de la práctica fue cuestión de prueba y error, lo más complicado fue diseñar la lógica detrás de las funciones requeridas para el envío, recepción y eliminación de carpetas puesto que se tuvieron que diseñar estas funciones de manera recursiva, además de encontrarnos con problemas como la necesidad de vaciar los directorios antes de tratar de eliminarlos, al final se pudo lograr un programa con un resultado satisfactorio y eficaz, es posible mejorar algunas partes de este mismo para que sea más eficiente en un futuro ya que aún tiene algunas carencias principalmente la velocidad de transferencia e incluso la propia interfaz de cliente que puede no ser tan amigable como algunos servicios en la nube como iCloud o Google Drive. Lo que mejor dejó en claro esta práctica es la importancia que tienen los servicios de comunicación en cuanto al envío y recepción de archivos puesto que día a día utilizamos estos servicios ya sea para nuestra vida personal como académica y laboral.

Bibliografía

[1] Oracle (s.f.). Primitive Data Types. [En línea]. Disponible en https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html#:~:text=long%3A%20The%20long%20data%20type,value%20of%20264-1.

[2] P. A. Sznajdleder, *Java a fondo: estudio del lenguaje y desarrollo de aplicaciones*. 2000.

[3] IBM (2021). Network byte order. [En línea]. Disponible en https://www.ibm.com/docs/en/zos/2.4.0?topic=hosts-network-byte-order

[4] Mozilla (2022). TCP. [En línea]. Disponible en https://developer.mozilla.org/es/docs/Glossary/TCP#:~:text=TCP%20(Protocolo%20de%20Control%20de,orden%20en%20que%20se%20enviaron.

[5] IBM (2022). Tipos de Socket. [En línea]. Disponible en https://www.ibm.com/docs/es/aix/7.3?topic=protocols-socket-types

[6] IBM (2022). ¿Qué es transferencia gestionada de archivos?. [En línea]. Disponible en https://www.ibm.com/mx-es/topics/file-transfer

Anexo 1: Código

Archivo.java

```java
package FlujoArchivos;

/**
 * @author Ruvalcaba Flores Martha Catalina
 * @author Sandoval Hernández Eduardo
 */

import java.io.File;

public class Archivo {
    private String name;
    private long size;
    private String path;
    private File file;

     public Archivo(String name, long size, String path, File file) {
        this.name = name;
        this.size = size;
        this.path = path;
        this.file = file;
    }

    public String getName() { return name; }
    public long getSize()   { return size; }
    public String getPath() { return path; }
    public File getFile()   { return file; }

    public void setName(String name)    { this.name = name; }
    public void setSize(long size)      { this.size = size; }
    public void setPath(String path)    { this.path = path; }
    public void setFile(File file)      { this.file = file; }

}
```

Cliente.java

```java
package FlujoArchivos;
```

```java
/**
 * @author Ruvalcaba Flores Martha Catalina
 * @author Sandoval Hernández Eduardo
 */

import java.io.*;
import java.net.*;
import java.util.*;

public class Cliente {
    //Se especifica host y puerto del servidor
    private String host;
    private int puerto;
    /*|-------------------------------------------------------------|
        Banderas para que el servidor identifique la accion a realizar
    |-------------------------------------------------------------|*/
    private final int banderaEnviar = 0; //Servidor recibe
    private final int banderaRecibir = 1;//Servidor envia
    private final int banderaEliminar = 2;//Servidor elimina archivo
    private final int banderaListar = 3;//Servidor envia informacion para actualizar el panel
    private final int banderaEliminarCarpeta = 4;//Servidor elimina carpeta
    /*|-------------------------------------------------------------*/
    private Socket cliente;
    private DataInputStream dis;
    private DataOutputStream dos;

    public Cliente(String host, int puerto){
        this.host = host;
        this.puerto = puerto;
    }

    public void enviarArchivo(ArrayList <Archivo> listArchivos, String pathDestino) {
        try {
            for(Archivo a : listArchivos) {
                File f = a.getFile();
                String nombre = a.getName();
                long tam = a.getSize();
                cliente = new Socket(this.host, this.puerto);
                dis = new DataInputStream(new FileInputStream(f.getAbsoluteFile()));
                dos = new DataOutputStream(cliente.getOutputStream());

                dos.writeInt(banderaEnviar);
                dos.flush();
                dos.writeUTF(nombre);
                dos.flush();
```

```
            dos.writeLong(tam);
            dos.flush();
            dos.writeUTF(pathDestino);
            dos.flush();

            long enviados = 0;
            int porcentaje, l = 0;

            while(enviados < tam){
                byte[] b = new byte[1500];
                l = dis.read(b);
                //System.out.println("enviados: "+l);
                dos.write(b,0,l);
                dos.flush();
                enviados = enviados + l;
                porcentaje = (int)((enviados*100)/tam);
                System.out.print("\rPorcentaje:" + porcentaje + "%");
            }

            System.out.println("Archivo Enviado");
            dos.close();
            dis.close();
            cliente.close();
        }

        listArchivos.clear();

    } catch(Exception e){
        e.printStackTrace();
    }
}

/*
Dado una carpeta, su direccion de destino y los archivos de esta, se envia al servidor.
*/
public void enviarCarpeta(File carpeta, String pathDestino, ArrayList<Archivo> listArchivos) {
    if(pathDestino.equals(""))
        pathDestino = carpeta.getName();
    else
        pathDestino = pathDestino + "\\" + carpeta.getName();

    for(File archivo : carpeta.listFiles()) {
        if(archivo.isDirectory())
            enviarCarpeta(archivo, pathDestino, listArchivos);
        else{
            listArchivos.add(new Archivo(archivo.getName(), archivo.length(), pathDestino, archivo));
```

```
            }
        }
        enviarArchivo(listArchivos, pathDestino);
    }

    /*
    Recibe un archivo dado este y su ruta de destino
    */
    public void recibirArchivo(Archivo file, String rutaDestino) {
        try {
            cliente = new Socket(this.host, this.puerto);

            dos = new DataOutputStream(cliente.getOutputStream());

            //Escribe los metadatos del archivo que el servidor debe enviar
            dos.writeInt(banderaRecibir);
            dos.flush();
            dos.writeUTF(file.getName());
            dos.flush();
            dos.writeLong(file.getSize());
            dos.flush();
            dos.writeUTF(file.getPath());
            dos.flush();

            dis = new DataInputStream(cliente.getInputStream());
            String nombre = dis.readUTF();
            long tam = dis.readLong();
            String ruta = dis.readUTF();

            DataOutputStream archivo = new DataOutputStream(new FileOutputStream(rutaDestino + "/" + nombre));

            System.out.println(rutaDestino);

            long recibidos = 0;
            int porcentaje, l = 0;

            while(recibidos < tam) {
                byte[] b = new byte[1500];
                l = dis.read(b);
                System.out.println("leidos: "+l);
                archivo.write(b,0,l);
                archivo.flush();
                recibidos = recibidos + l;
                porcentaje = (int)((recibidos*100)/tam);
                System.out.println("Recibido el " + porcentaje + "% del archivo");
            }
```

```
            System.out.println("Archivo recibido...");
            dis.close();
            dos.close();
            archivo.close();

        }catch(Exception e){
            e.printStackTrace();
        }
    }

    /*
    Dado una carpeta y su direccion de destino, se recibe dicha carpeta en el directorio local
    para ello comprueba que existan las rutas si no las crea y recibe los archivos uno por uno en
    la ruta especificada.
    */
    public void recibirCarpeta(File carpeta, String pathDestino) throws IOException{
        try {
            File temp = new File(pathDestino);
            if(!temp.exists()){         //Si no existe el directorio
                try{
                    if(temp.mkdirs()){  //Si pudo crear el directorio
                        temp.setWritable(true);
                        System.out.println("Se ha creado la ruta");
                    }
                    else                //Si no pudo crear el directorio
                        System.out.println("No se ha podido crear la ruta");
                }catch(SecurityException se){
                    se.printStackTrace();
                }
            }

            for(File archivo : carpeta.listFiles()) {   //Por cada archivo en la carpeta a recibir
                if(archivo.isDirectory())               //Si es una subcarpeta
                    recibirCarpeta(archivo, pathDestino + "\\" + archivo.getName() + "\\");//Llamada recursiva
                else{                                   //Si no es una carpeta
                    Archivo a = new Archivo(archivo.getName(), archivo.length(), archivo.getAbsolutePath(),
archivo);
                    recibirArchivo(a, pathDestino);     //Recibe el archivo
                }
                //Archivo a = new Archivo(archivo.getName(), archivo.length(), archivo.getAbsolutePath(), archivo);
                //System.out.println("XDDDDDDDDDDDDDD2" + pathDestino);
                //recibirArchivo(a, pathDestino);
            }
        }catch(IOException e){
            e.printStackTrace();
```

```
        }
    }

    /*
    Dado un arraylist de archivos, se envian sus metadatos con la bandera de eliminar para que
    el servidor elimine dichos archivos.
    */
    public void eliminarArchivo(ArrayList <Archivo> listArchivos) {//, String path
        try {
            for(Archivo a : listArchivos) { //Por cada archivo a eliminar
                File f = a.getFile();
                String nombre = a.getName();
                long tam = a.getSize();
                cliente = new Socket(this.host, this.puerto);
                dis = new DataInputStream(new FileInputStream(f.getAbsoluteFile()));
                dos = new DataOutputStream(cliente.getOutputStream());

                //Se escriben los metadatos
                dos.writeInt(banderaEliminar);
                dos.flush();
                dos.writeUTF(nombre);
                dos.flush();
                dos.writeLong(tam);
                dos.flush();
                dos.writeUTF(f.getAbsolutePath());
                dos.flush();

                dos.close();
                dis.close();
                cliente.close();
            }

            listArchivos.clear();

        } catch(Exception e){
            e.printStackTrace();
        }
    }

    /*
    Dada una carpeta, envia sus metadatos para con la bandera de eliminar carpeta para que el servidor
    elimine dicha carpeta.
    */
    public void eliminarCarpeta(File carpeta){
        try {
            File temp = new File("temp.txt");
```

```
            String nombre = carpeta.getName();
            long tam = carpeta.length();
            cliente = new Socket(this.host, this.puerto);
            dis = new DataInputStream(new FileInputStream(temp.getAbsoluteFile()));//***DUDA***
            dos = new DataOutputStream(cliente.getOutputStream());

            //Escribe los metadatos
            dos.writeInt(banderaEliminarCarpeta);
            dos.flush();
            dos.writeUTF(nombre);
            dos.flush();
            dos.writeLong(tam);
            dos.flush();
            dos.writeUTF(carpeta.getAbsolutePath());
            dos.flush();

            //dis = new DataInputStream(cliente.getInputStream());
            dos.close();
            dis.close();
            cliente.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    public Archivo actualizarPantalla(String path) {
        try {
            //System.out.println("Actualizar pantalla");
            File empty = new File("");
            cliente = new Socket(this.host, this.puerto);

            dos = new DataOutputStream(cliente.getOutputStream());

            //Escribe los metadatos del archivo que el servidor debe enviar
            dos.writeInt(banderaListar);
            dos.flush();
            dos.writeUTF("");
            dos.flush();
            dos.writeLong(0);
            dos.flush();
            dos.writeUTF(path);
            dos.flush();
            //System.out.println("Se escribieron los metadatos");

            dis = new DataInputStream(cliente.getInputStream());
            String nombre = dis.readUTF();
```

```
            long tam = dis.readLong();
            String ruta = dis.readUTF();
            //System.out.println("Se recibieron los metadatos");

            Archivo datos = new Archivo(nombre, tam, ruta, empty);
            //System.out.println("Cliente pide metadatos");
            dis.close();
            dos.close();
            return datos;

        }catch(Exception e){
            e.printStackTrace();
        }
        return null;
    }
}
```

Servidor.java

```
package FlujoArchivos;

/**
 * @author Ruvalcaba Flores Martha Catalina
 * @author Sandoval Hernández Eduardo
 */

import java.io.*;
import java.net.*;

public class Servidor {
    private int puerto;
    private ServerSocket servidor;
    private Socket cliente;
    /*|---------------------------------------------------------------|
        Banderas para que el servidor identifique la accion a realizar
    |---------------------------------------------------------------|*/
    private final int banderaRecibir = 0;   //Servidor recibe
    private final int banderaEnviar = 1;    //Servidor envia
    private final int banderaEliminar = 2;  //Servidor elimina archivo
    private final int banderaListar = 3;    //Servidor lista los archivos en ruta
    private final int banderaEliminarCarpeta = 4;   //Servidor elimina carpeta
    /*|---------------------------------------------------------------*/
    //Constructor del servidor
    public Servidor(int puerto) {
        this.puerto = puerto;
```

```
    }

    //Cuando se conecta un cliente al servidor
    public void conectar() {
        try {
            servidor = new ServerSocket(puerto);    //Se inicializa el socket con el numero de puerto ya
especificado
            servidor.setReuseAddress(true);         //Se establece la posibilidad de reutilizar direcciones
            System.out.println("Servidor iniciado esperando una accion...");
            //Ciclo for para aceptar clientes
            for(;;) {
                cliente = servidor.accept();    //Se inicializa el socket de cliente cuando el socket servidor
acepta una conexion
                System.out.println("Cliente conectado desde "+cliente.getInetAddress()+":"+cliente.getPort());
                DataInputStream dis = new DataInputStream(cliente.getInputStream());    //Se crea el DIS
                /*Se leen los metadatos*/
                int bandera = dis.readInt();    //Bandera
                String nombre = dis.readUTF();  //Nombre del archivo o carpeta
                long tam = dis.readLong();      //Tamano del archivo o carpeta
                String ruta = dis.readUTF();    //Ruta del archivo o carpeta
                /*|-------------------------------------------------------------*/
                /*Variables para identificar la ruta del servidor*/
                File temp = new File("");
                String absolute_path = temp.getAbsolutePath();
                String folder_name = "archivos";
                String ruta_archivos = absolute_path + "\\" + folder_name + "\\";
                /*|-------------------------------------------------------------*/

                switch(bandera){
                    case banderaRecibir:
                        nombre = comprobarRuta(nombre, ruta);
                        nombre = ruta_archivos + nombre;
                        recibirArchivo(nombre, tam, dis);
                        break;

                    case banderaEnviar:
                        enviarArchivo(nombre, tam, ruta);
                        break;

                    case banderaEliminar:
                        eliminarArchivo(ruta);
                        break;
                    case banderaEliminarCarpeta:
                        eliminarCarpeta(ruta + "\\");
                        break;
```

```
                case banderaListar:
                    actualizarPantalla(ruta);
                    break;

                default:
                    break;
            }

            cliente.close();
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}

/*
Esta funcion se encarga de comprobar/crear los directorios para cuando se reciben carpetas
*/
public String comprobarRuta(String nombre, String ruta) {
    if(!ruta.equals("")) {
        File temp = new File("");
        String absolute_path = temp.getAbsolutePath();
        String folder_name = "archivos";
        String ruta_archivos = absolute_path + "\\" + folder_name + "\\";
        File carpeta = new File(ruta_archivos + "\\" + ruta + "\\");
        System.out.println(carpeta.getAbsolutePath());

        if(!carpeta.exists()){
            try{
                if(carpeta.mkdirs()){
                    carpeta.setWritable(true);
                    System.out.println("Se ha creado la ruta");
                }
                else
                    System.out.println("No se ha podido crear la ruta");
            }catch(SecurityException se){
                se.printStackTrace();
            }
        }
        else{
            System.out.println("Ya existe la ruta");
        }
        nombre = ruta + "\\" + nombre;
    }
    return nombre;
}
```

```
public void recibirArchivo(String nombre, long tam, DataInputStream dis) {
    try{
        int porcentaje, l = 0;
        long recibidos = 0;
        DataOutputStream dos = new DataOutputStream(new FileOutputStream(nombre));
        while(recibidos < tam) {
            byte[] b = new byte[1500];
            l = dis.read(b);
            //System.out.println("leidos: "+l);
            recibidos += l;
            dos.write(b, 0, l);
            dos.flush();
            porcentaje = (int)((recibidos*100)/tam);
            System.out.print("\rSe ha recibido " + porcentaje + "%");
        }
            dos.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}

public void enviarArchivo(String nombre, long tam, String path) {
    try{
        DataOutputStream dos = new DataOutputStream(cliente.getOutputStream());
        DataInputStream dis = new DataInputStream(new FileInputStream(path));

        //Escribe los metadatos del archivo a enviar
        dos.writeUTF(nombre);
        dos.flush();
        dos.writeLong(tam);
        dos.flush();
        dos.writeUTF(path);
        dos.flush();

        int porcentaje, l = 0;
        long enviados = 0;

        while(enviados < tam){
            byte[] b = new byte[1500];
            l = dis.read(b);
            System.out.println("enviados: "+l);
            dos.write(b, 0, l);
            dos.flush();
            enviados = enviados + l;
            porcentaje = (int)((enviados*100)/tam);
```

```
                System.out.print("\rEnviado el " + porcentaje + "% del archivo");
            }
            System.out.println("\nArchivo Enviado...");
            dis.close();
            dos.close();

        }catch(Exception e){
            e.printStackTrace();
        }
    }

    /*
    Elimina un archivo dada la ruta de este
    */
    public void eliminarArchivo(String nombre){
        try{
            File temp = new File(nombre);
            System.out.println(temp.getAbsolutePath());
            if(temp.exists()){                      //Comprueba la existencia del archivo
                try{
                    if(temp.delete())               //Si ha sido eliminado correctamente
                        System.out.println("El archivo de ha eliminado satisfactoriamente");
                    else                            //Si no
                        System.out.println("No se ha podido eliminar el archivo");
                }catch(SecurityException se){
                    se.printStackTrace();
                }
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    /*
    Elimina una carpeta dada su ruta, para ello elimina todos los archivos que puedan existir dentro
    de esta y de las posibles subcarpetas y una vez que se encuentra vacia elimina el directorio
    */
    public void eliminarCarpeta(String ruta){
        try{
            File carpeta = new File(ruta);
            //System.out.println(carpeta.getAbsolutePath());
            //carpeta.delete();
            if (carpeta.exists()){                                          //Si el directorio existe
                try{
                    for(File archivo : carpeta.listFiles()){                //Por cada archivo dentro del
directorio
```

```
                    if(archivo.isDirectory())                                   //Comprueba si es una subcarpeta
                        eliminarCarpeta(ruta + "\\" + archivo.getName());   //Llamada recursiva con la direccion
de la subcarpeta
                    archivo.delete();                                           //Elimina el archivo o subcarpeta
vacia
                    //eliminarArchivo(archivo.getName());                     //Elimina el archivo o subcarpeta
vacia
                }
                carpeta.delete();//Elimina la carpeta vacia en la ruta
            }catch(SecurityException se){
                se.printStackTrace();
            }
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}

public void actualizarPantalla(String path){
    try{
        File aux = new File(path);
        File temp = new File("temp.txt");
        String nombre = aux.getName();
        long tam;
        if(aux.isDirectory())
            tam = 0;
        else
            tam = aux.length();

        DataOutputStream dos = new DataOutputStream(cliente.getOutputStream());
        DataInputStream dis = new DataInputStream(new FileInputStream("temp.txt"));

        //Escribe los metadatos del archivo a enviar
        dos.writeUTF(nombre);
        dos.flush();
        dos.writeLong(tam);
        dos.flush();
        dos.writeUTF(path);
        dos.flush();
        //System.out.println("Servidor envia metadatos");
        dis.close();
        dos.close();

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

```
        }
    }

    public static void main(String[] args) {
        int PUERTO = 9000;
        //long serialVersionUID = 2L;
        Servidor Server = new Servidor(PUERTO);
        Server.conectar();
    }
}
```

ClienteGUI.java

```
package FlujoArchivos;

/**
 * @author Ruvalcaba Flores Martha Catalina
 * @author Sandoval Hernández Eduardo
 */

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.table.*;
import java.io.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.tree.DefaultMutableTreeNode;

public class ClienteGUI extends JFrame  implements ActionListener  {
    private Container container;
    private JPanel mainPanel;
    private JLabel cliente;
    private JPanel panelBotones;
    private JButton btnCargarArchivo, btnCargarCarpeta, btnDescargarArchivo, btnDescargarCarpeta, btnEliminar,
btnEliminarCarpeta, btnActualizar;
    private JFileChooser file;
    private ArrayList <Archivo> listArchivosParaCargar;
    private Cliente clienteActual;
    private final int PUERTO = 9000;
    private final String HOST = "127.0.0.1";
    private static final long serialVersionUID = 1L;

    private File temp = new File("");
```

```
    private String absolute_path = temp.getAbsolutePath();
    private String folder_name = "archivos";
    private String local_folder_name = "archivoslocal";
    private String ruta_archivos = absolute_path + "\\" + folder_name + "\\";
    private String ruta_archivos_local = absolute_path + "\\" + local_folder_name + "\\";

    public ClienteGUI() {
        container = getContentPane();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));

        setTitle("Practica no. 1 Servicio de transferencia de archivos");
        //setSize(1000,600);
        setSize(900,100);
        //setBounds(300, 100, 450, 600);
        setResizable(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);



        mainPanel = new JPanel();
        panelBotones = new JPanel();
        cliente = new JLabel("Interfaz de cliente");
        btnCargarArchivo = new JButton("Cargar Archivo");
        btnCargarCarpeta = new JButton("Cargar Carpeta");
        btnDescargarArchivo = new JButton("Descargar Archivo");
        btnDescargarCarpeta = new JButton("Descargar Carpeta");
        btnEliminar = new JButton("Eliminar Archivo");
        btnEliminarCarpeta = new JButton("Eliminar Carpeta");
        listArchivosParaCargar = new ArrayList<>();

        panelBotones.add(btnCargarArchivo);
        panelBotones.add(btnCargarCarpeta);
        panelBotones.add(btnDescargarArchivo);
        panelBotones.add(btnDescargarCarpeta);
        panelBotones.add(btnEliminar);
        panelBotones.add(btnEliminarCarpeta);
        container.add(panelBotones);

        mainPanel.setLayout(new BorderLayout());
        mainPanel.add(cliente, BorderLayout.NORTH);
        mainPanel.add(panelBotones, BorderLayout.SOUTH);
        add(mainPanel);

        btnCargarArchivo.addActionListener(this);
        btnCargarCarpeta.addActionListener(this);
        btnDescargarArchivo.addActionListener(this);
        btnDescargarCarpeta.addActionListener(this);
```

```
        btnEliminar.addActionListener(this);
        btnEliminarCarpeta.addActionListener(this);

        setVisible(true);

    }

    public DefaultMutableTreeNode arbolDeRuta(File file, DefaultMutableTreeNode Padre, int pos){
        int pos2 = pos;
        for(File archivo : file.listFiles()){
            clienteActual = new Cliente(HOST, PUERTO);
            Archivo tempFile = clienteActual.actualizarPantalla(archivo.getAbsolutePath());
            DefaultMutableTreeNode tempNode = new DefaultMutableTreeNode(archivo.getName());
            if(tempFile.getSize() == 0){
                //modelo.insertNodeInto(tempNode,Padre,pos2);



            }
                else
                    Padre.insert(tempNode,pos2);
            pos2++;

        }
        return Padre;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton boton = (JButton) e.getSource();
//        File temp = new File("");
//        String absolute_path = temp.getAbsolutePath();
//        String folder_name = "archivos";
//        String local_folder_name = "archivoslocal";
//        String ruta_archivos = absolute_path + "\\" + folder_name + "\\";
//        String ruta_archivos_local = absolute_path + "\\" + local_folder_name + "\\";

        if(boton == btnCargarArchivo){
            file = new JFileChooser(ruta_archivos_local);
            //file = new JFileChooser();
            file.setFileSelectionMode(JFileChooser.FILES_ONLY);
            file.requestFocus();
            if (!file.isMultiSelectionEnabled()) {
                file.setMultiSelectionEnabled(true);
            }
            int returnVal = file.showOpenDialog(ClienteGUI.this);
            if(returnVal == JFileChooser.APPROVE_OPTION) {
                File[] selectedFiles = file.getSelectedFiles();
```

```
            for(File archivos : selectedFiles){
                File f = archivos;
                listArchivosParaCargar.add(new Archivo(f.getName(), f.length(), f.getAbsolutePath(), f));
                //modelo.addRow(new Object[] { f.getName() });
            }
            clienteActual = new Cliente(HOST, PUERTO);
            clienteActual.enviarArchivo(listArchivosParaCargar, "");
            //modelo.setRowCount(0);
        }

    } else if (boton == btnCargarCarpeta){
        JFileChooser jf = new JFileChooser(ruta_archivos_local);
        //JFileChooser jf = new JFileChooser();
        jf.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        jf.requestFocus();
        if (!jf.isMultiSelectionEnabled()) {
            jf.setMultiSelectionEnabled(true);
        }
        int returnVal = jf.showOpenDialog(ClienteGUI.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File[] selectedFiles = jf.getSelectedFiles();
            for(File archivos : selectedFiles){
                //modelo.addRow(new Object[]{"Carpeta: " + archivos.getName()});
                clienteActual = new Cliente(HOST, PUERTO);
                clienteActual.enviarCarpeta(archivos, "", listArchivosParaCargar);
                //modelo.setRowCount(0);
            }
        }
    } else if (boton == btnDescargarArchivo){



        JFileChooser jf = new JFileChooser(ruta_archivos);
        jf.setFileSelectionMode(JFileChooser.FILES_ONLY);
        jf.requestFocus();
        if (!jf.isMultiSelectionEnabled()) {
            jf.setMultiSelectionEnabled(true);
        }
        int returnVal = jf.showOpenDialog(ClienteGUI.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            JFileChooser jf2 = new JFileChooser(ruta_archivos_local);
            jf2.requestFocus();
            jf2.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            int r2 = jf2.showOpenDialog(ClienteGUI.this);
            if(r2 == JFileChooser.APPROVE_OPTION) {
                File[] selectedFiles = jf.getSelectedFiles();
                for(File archivos : selectedFiles){
```

```
                        File f2 = jf2.getSelectedFile();
                        Archivo a = new Archivo(archivos.getName(), archivos.length(), archivos.getAbsolutePath(),
archivos);
                        clienteActual = new Cliente(HOST, PUERTO);
                        clienteActual.recibirArchivo(a, f2.getAbsolutePath());
                    }
                }
            }
        } else if (boton == btnDescargarCarpeta){
            JFileChooser jf = new JFileChooser(ruta_archivos);
            jf.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            jf.requestFocus();
            if (!jf.isMultiSelectionEnabled()) {
                jf.setMultiSelectionEnabled(true);
            }
            int returnVal = jf.showOpenDialog(ClienteGUI.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                JFileChooser jf2 = new JFileChooser(ruta_archivos_local);
                jf2.requestFocus();
                jf2.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
                int r2 = jf2.showOpenDialog(ClienteGUI.this);
                if(r2 == JFileChooser.APPROVE_OPTION) {
                    File[] selectedFiles = jf.getSelectedFiles();
                    for(File archivos : selectedFiles){
                        File f2 = jf2.getSelectedFile();
                        clienteActual = new Cliente(HOST, PUERTO);
                        try {
                            clienteActual.recibirCarpeta(archivos, f2.getAbsolutePath() + "\\" +
archivos.getName());
                        } catch (IOException ex) {
                            Logger.getLogger(ClienteGUI.class.getName()).log(Level.SEVERE, null, ex);
                        }
                    }
                }
            }
        } else if (boton == btnEliminar){
            file = new JFileChooser(ruta_archivos);
            file.setFileSelectionMode(JFileChooser.FILES_ONLY);
            file.requestFocus();
            if (!file.isMultiSelectionEnabled()) {
                file.setMultiSelectionEnabled(true);
            }
            int returnVal = file.showOpenDialog(ClienteGUI.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File[] selectedFiles = file.getSelectedFiles();
                for(File archivos : selectedFiles){
```

```
                    File f = archivos;
                    listArchivosParaCargar.add(new Archivo(f.getName(), f.length(), f.getAbsolutePath(), f));
                }
                clienteActual = new Cliente(HOST, PUERTO);
                clienteActual.eliminarArchivo(listArchivosParaCargar);
            }
        } else if (boton == btnEliminarCarpeta){
            file = new JFileChooser(ruta_archivos);
            file.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            file.requestFocus();
            if (!file.isMultiSelectionEnabled()) {
                file.setMultiSelectionEnabled(true);
            }
            int returnVal = file.showOpenDialog(ClienteGUI.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File[] selectedFiles = file.getSelectedFiles();
                for(File archivos : selectedFiles){
                    clienteActual = new Cliente(HOST, PUERTO);
                    clienteActual.eliminarCarpeta(archivos);
                }
            }
        }

    }

    public static void main(String[] args) {
        new ClienteGUI();
    }

}
```