INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

----------**APLICACIONES EN COMUNICACIONES EN RED**----------

PRÁCTICA 1:

Servicio de transferencia de archivo

Alumno:

Meza Vargas Brandon David

Grupo:
3CM16

Profesor:

Moreno Cervantes Axel Ernesto

índice

índice de Ilustraciones

Introducción

Los sockets permiten conectar dos programas en red para que se puedan intercambiar datos. Los sockets están basados en una arquitectura cliente/servidor en donde el servidor esta continuamente a la escucha y a la espera de que alguien se quiera comunicar a él.

Los sockets de flujo son un servicio orientado a la conexión, donde los datos se transfieren sin encuadrarlas en registros o bloques, estos están basados en el protocolo TCP brindando comunicaciones fiables.

En Java las comunicaciones TCP se realizan utilizando la clásica abstracción de socket.

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.

En la presente práctica se realizará una aplicación que permita el envío y descarga de archivos de un servidor a un cliente y viceversa empleando los sockets de flujo en el lenguaje de programación Java.

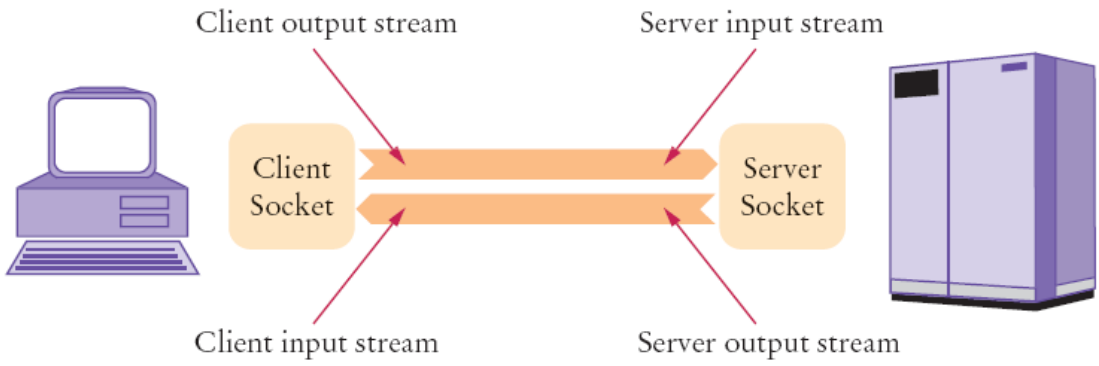

Ilustración 1. Sockets de flujo

Desarrollo

Para el desarrollo de esta práctica fue necesario implementar dos partes, una de parte del cliente que se conectará al servidor para transferir archivos así como recibirlos y la del servidor que se mantiene a la escucha del cliente para saber cuando mandar o recibir archivos.

Cliente

Función chooser

Esta función se encarga de crear la carpeta donde se almacenarán los archivos del cliente, así como de verificar si lo que quiere hacer el cliente es mandar archivos al servidor o descargar archivos del mismo servidor, esto se determina por medio del parámetro que recibe mandado desde la interfaz del usuario.

```
public void chooser(Boolean op) throws FileNotFoundException, IOException{
    //*Creating client folder
    File f = new File("");
    String path = f.getAbsolutePath();
    String folder = "ClientFiles";
    String files_path = path + "\\" + folder + "\\";
    System.out.println("Client folder path: " + files_path);
    File f2 = new File(files_path);
    f2.mkdirs();
    f2.setWritable(true);

    //*If op is true means we want to send a file to the server
    if(op){
        System.out.println("Connection stablished");
        JFileChooser jf = new JFileChooser();
        if(!op) jf.setCurrentDirectory(new File(serverPath));
        jf.setMultiSelectionEnabled(true);
        jf.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

        int r = jf.showOpenDialog(null);
        jf.setRequestFocusEnabled(true);

        if(r == JFileChooser.APPROVE_OPTION){
            File[] files = jf.getSelectedFiles();
            sendFiles(files, "", false);
        }
    }else{
        Socket cop = new Socket(dir, pto+1000);
        DataOutputStream dcop = new DataOutputStream(cop.getOutputStream());
        //*Sending op
        dcop.writeInt(0);
        dcop.flush();
        dcop.close();
        cop.close();
```

```
        //*Reading if download is cancelled
        cop = new Socket(dir, pto+1000);
        DataInputStream dis = new DataInputStream(cop.getInputStream());
        Integer isCancelled = dis.readInt();
        dis.close();
        cop.close();

        //*Verifying if download is cancelled or not
        if(isCancelled == 1){
          return;
        }else{
            for(;;){
                client = new Socket(dir, pto+1000);
                dis = new DataInputStream(client.getInputStream());
                Integer fi = dis.readInt();
                //*If data has been downloaded we exit
                if(fi == 1){
                    break;
                }

                dis.close();
                client.close();

                //*Reading file to download info
                client  = new Socket(dir, pto);
                dis = new DataInputStream(client.getInputStream());
                String name = dis.readUTF();
                long size = dis.readLong();
                boolean directory = dis.readBoolean();
                downloadFiles(files_path, dis, size, name, directory);
            }

        }
    }
}
```

Ilustración 2. Función chooser del cliente.

Función sendFiles

Esta función se encarga de mandar los archivos al servidor, se hace un ciclo que recorre todos los archivos seleccionados por el usuario para mandar al servidor, por cada archivo verificamos si es una carpeta o un simple archivo, en caso de ser una carpeta se manda a llamar la misma función con recursión para crear la carpeta en el servidor y mandar los archivos dentro de ella.

En caso de ser un archivo se abre un socket que se conecta al servidor para indicar al servidor que se esta enviando un archivo y se abre uno mas para indicar los datos del archivo que se esta enviado como su tamaño, nombre y si es una carpeta. Finalmente con un ciclo while vamos mandando los bytes del archivo hasta mandarlo por completo.

```java
public void sendFiles(File [] currentFiles, String directoryName, boolean directory){
    try {
        //*Indicating the port and socket direction
        int pto = 8000;
        String dir = "127.0.0.1";

        //*We go trough the files to verify if they're folder or file
        for(File fileToSend : currentFiles){
            if(fileToSend.isDirectory()){
                directoryName += directory ? "/" : "";
                sendFiles(fileToSend.listFiles(), directoryName+fileToSend.getName(), true);
            }else{
                Socket cl = new Socket(dir, pto);

                DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
                String currentName = directory ? directoryName + "/" + fileToSend.getName() : fileToSend.getName();
                String currentPath = fileToSend.getAbsolutePath();
                long currentSize = fileToSend.length();

                System.out.println("Sending file " + currentName + " with size " + currentSize + " bytes");

                DataInputStream dis = new DataInputStream(new FileInputStream(currentPath));

               //*Sending op
                Socket cop = new Socket(dir, pto+1000); //*pto 8000+1000
                DataOutputStream dcop = new DataOutputStream(cop.getOutputStream());
                dcop.writeInt(1);
                dcop.flush();
                dcop.close();
                cop.close();

                //*File info
                dos.writeUTF(currentName);
                dos.flush();
                dos.writeLong(currentSize);
                dos.flush();
                dos.writeBoolean(directory);
                dos.flush();

                //*Sending file proccess
                long currentSent = 0;
                Integer l = 0, currentPercentage = 0;

                while(currentSent  < currentSize){
                    byte[] b = new byte[1500];
                    l = dis.read(b);
                    System.out.println("Sent " + l);
                    dos.write(b,0,l);
                    dos.flush();
                    currentSent = currentSent + l;
                    currentPercentage = (int)((currentSent*100) / currentSize);
                    System.out.println(ANSI_GREEN + "\rSending data: " + currentPercentage + "%" + ANSI_RESET);

                } //*While end

                System.out.println(ANSI_PURPLE + currentName + " sent");
                dos.close();
                dis.close();
                cl.close();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 3. Función sendFiles del cliente.

Función downloadFiles

Por último tenemos la función downloadFiles(), en esta vamos descargando los archivos del servidor que el usuario selecciono para descargar, este función se ejecuta en un ciclo infinito hasta que el servidor indique que ya se enviaron todos los datos.

El proceso es similar a la función de enviar datos del cliente, la diferencia es que aquí vamos leyendo en lugar de escribir los bytes de los archivos mandados por el servidor.

```
public void downloadFiles(String dest, DataInputStream dis, long size, String name, Boolean directory){
        try {
            //*Verifying if the file is a directory
             File newFile = new File(dest+name);
             if(directory){
                 File directoryDir = newFile.getParentFile();
                 while(directoryDir.getPath().contains(dest)){
                     directoryDir.mkdirs();
                     directoryDir.setWritable(true);
                     directoryDir = directoryDir.getParentFile();
                 }
                 newFile.createNewFile();
             }

             DataOutputStream dos = new DataOutputStream(new FileOutputStream(newFile));
             long receivedData = 0;
             int l = 0, percentage = 0;

             //*Downloading file proccess
             System.out.println("Downloading " + name + " " + size);

             while(receivedData < size){
                 byte[] b = new byte[1500];
                 l = dis.read(b);
                 System.out.println("Read: " + l);
                 dos.write(b,0,l);
                 dos.flush();
                 receivedData = receivedData + l;
                 percentage = (int)((receivedData*100) / size);
                 System.out.println(ANSI_GREEN + "Downloading data; " + percentage + "%" + ANSI_RESET);
             }//*While end

             System.err.println("Data downloaded" + name);
             dis.close();
             dos.close();
             client.close();
         } catch (Exception e) {
                 e.printStackTrace();
           }
```

Ilustración 4. Función downloadFiles();

Servidor

Por parte del servidor las funciones de enviar y recibir datos son muy similares a las del cliente con la diferencia que en estas se aceptan conexiones de socket por parte del cliente, por esto se limitara a colocar el código de cada función.

Función connect

En esta función se hace un ciclo infinito donde un socket siempre estará abierto para esperar datos por parte del cliente, se usaron dos, uno para recibir datos por el puerto 8000 y otr por el puerto 9000 para evitar confusiones y saturaciones al enviar datos por los sockets.

En esta función recibimos una operación que se envía por parte del cliente cuando este indica lo que quiere hacer, en este caso si se manda un 1 el servidor se prepara para recibir archivos, de otro modo manda datos al cliente.

```
public void connect(){
    try {
        Integer op;

        System.err.println("Server connected!");
        System.out.println("Server waiting for files..");

        //*Creting the server folder
        File f = new File("");
        String path = f.getAbsolutePath();
        String folder = "Serverfiles";
        files_path = path + "\\" + folder + "\\";
        System.out.println("Server folder path: " + files_path);
        File f2 = new File(files_path);
        f2.mkdirs();
        f2.setWritable(true);

        //*Opening first serverSocket
        s = new ServerSocket(pto);
        s.setReuseAddress(true);

        //*Opening a second serverSocket for connections
        sop = new ServerSocket(pto+1000);
        sop.setReuseAddress(true);

        //*The server waits for a client connection
        for(;;){
            //*Recieving the option (send or receive files)
            client = sop.accept();
            DataInputStream dis = new DataInputStream(client.getInputStream());
            op = dis.readInt();
            dis.close();
            client.close();

              System.out.println("Operacion" + op);

            System.out.println(ANSI_BLUE + "Client connected from " + client.getInetAddress()+":" + client.getPort() + ANSI_

            if(op == 1){
                //*Accepting connection
                Socket client;
                client = s.accept();
                //*Receiving the file to recieve info
                dis = new DataInputStream(client.getInputStream());
                String name = dis.readUTF();
                long size = dis.readLong();
                boolean directory = dis.readBoolean();
                //*Function that receives every from the client
                recieveFiles(files_path, dis, size, name, directory);
            }else{
                chooser();
                  System.out.println("Finished");
                //*Indicating when data transfer is completed
                client = sop.accept();
                DataOutputStream dos = new DataOutputStream(client.getOutputStream());
                dos.writeInt(1);
                dos.close();
                client.close();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 5. Función connect del servidor()

Función recieveFiles

```
public void recieveFiles(String dest, DataInputStream dis, long size, String name, Boolean directory){
    try {
        for(;;){
            //*Creating a file with the destination and file name
            File newFile = new File(dest+name);
            //*We use recursion to recieve files whic are folders
            if(directory){
                File directoryDir = newFile.getParentFile();
                while(directoryDir.getPath().contains(dest)){
                    directoryDir.mkdirs();
                    directoryDir.setWritable(true);
                    directoryDir = directoryDir.getParentFile();
                }
                newFile.createNewFile();
            }
            DataOutputStream dos = new DataOutputStream(new FileOutputStream(newFile));
            long receivedData = 0;
            int l = 0, percentage = 0;
            System.err.println("Receiving " + name + " " + size);

            //*Recieving file
            while(receivedData < size){
                byte[] b = new byte[1500];
                l = dis.read(b);
                System.out.println("Read: " + l);
                dos.write(b,0,l);
                dos.flush();
                receivedData = receivedData + l;
                percentage = (int)((receivedData*100) / size);
                System.out.println(ANSI_GREEN + "\rReceiving data: " + percentage + "%" + ANSI_RESET);
            }//*While end

            System.out.println(ANSI_PURPLE +  name + " sent" + ANSI_RESET);
            //*Closing connections
            dos.close();
            dis.close();
            client.close();
            break;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 6. Función recieveFiles

Función sendFiles

```
public void sendFiles(File [] currentFiles, String directoryName, boolean directory){
    try {
        for(File fileToSend : currentFiles){
            if(fileToSend.isDirectory()){
                directoryName += directory ? "/" : "";
                sendFiles(fileToSend.listFiles(), directoryName+fileToSend.getName(), true);

            }else{
                String currentName = directory ? directoryName + "/" + fileToSend.getName() : fileToSend.getName();
                String currentPath = fileToSend.getAbsolutePath();
                long currentSize = fileToSend.length();

                System.out.println("Sending file " + currentName + " with size " + currentSize + " bytes");

                DataInputStream dis = new DataInputStream(new FileInputStream(currentPath));

                //*Sending if the data transfer is completed (in this case isn't)
                client = sop.accept();
                DataOutputStream dos = new DataOutputStream(client.getOutputStream());
                dos.writeInt(0);
                dos.close();
                client.close();

                //*File info
                Socket client = s.accept();
                dos = new DataOutputStream(client.getOutputStream());
                dos.writeUTF(currentName);
                dos.flush();
                dos.writeLong(currentSize);
                dos.flush();
                dos.writeBoolean(directory);
                dos.flush();

                //*Sending file proccess
                long currentSent = 0;
                int l = 0, currentPercentage = 0;
                System.out.println("Sending...");
                while(currentSent  < currentSize){
                    byte[] b = new byte[1500];
                    l = dis.read(b);
                    System.out.println("Sent " + l);
                    dos.write(b,0,l);
                    dos.flush();
                    currentSent = currentSent + l;
                    currentPercentage = (int)((currentSent*100) / currentSize);
                    System.out.println(ANSI_GREEN + "\rSending data; " + currentPercentage + "%" + ANSI_RESET);

                } //*While end

                System.err.println(currentName + " sent");
                dis.close();
                dos.close();
                client.close();
            }
        }

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

}
```

Ilustración 7. Función sendFiles del servidor

Función chooser

Esta función se manda a llamar cuando vamos a mandar archivos solicitados al cliente, esta se encarga de abrir el seleccionador de archivos dentro de la carpeta del servidor para que el usuario pueda seleccionar que archivos quiere descargar.

```
*/
public void chooser() throws FileNotFoundException, IOException{

//*Opengin file chooser
    JFileChooser jf = new JFileChooser();
    jf.setCurrentDirectory(new File(files_path));
    jf.setMultiSelectionEnabled(true);
    jf.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

    int r = jf.showOpenDialog(null);
    jf.setRequestFocusEnabled(true);

    if(r == JFileChooser.APPROVE_OPTION){
        Socket cho = sop.accept();
        DataOutputStream chod = new DataOutputStream(cho.getOutputStream());
        File[] files = jf.getSelectedFiles();
        chod.writeInt(0);
        chod.flush();
        cho.close();
        sendFiles(files, "", false);
     } else if(r == JFileChooser.CANCEL_OPTION){
     Socket cho = sop.accept();
        DataOutputStream chod = new DataOutputStream(cho.getOutputStream());
        chod.writeInt(1);
        chod.flush();
        cho.close();
        System.out.println("Download cancelled");
     }
}
```

Ilustración 8. Función chooser del servidor.

Pruebas de funcionamiento

En la siguiente ilustración podemos ver la interfaz del usuario.

Ilustración 9. Interfaz usuario.

Si listamos los archivos podemos ver que no hay nada ni en el cliente ni en el servidor (ver ilustración 10).

Ilustración 10. Listado de archivos.

Primeramente vamos a mandar archivos al servidor, estos serán carpetas con datos a dentro.

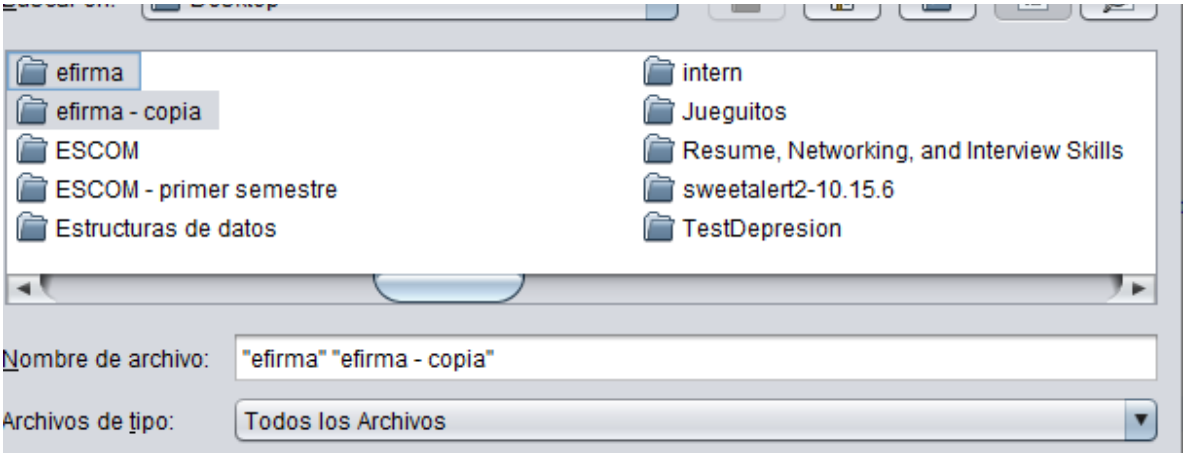

Ilustración 11. Mandando archivos al servidor

Una vez mandado podemos listar los archivos para verificar que se enviaron correctamente como se muestra en la siguiente figura.

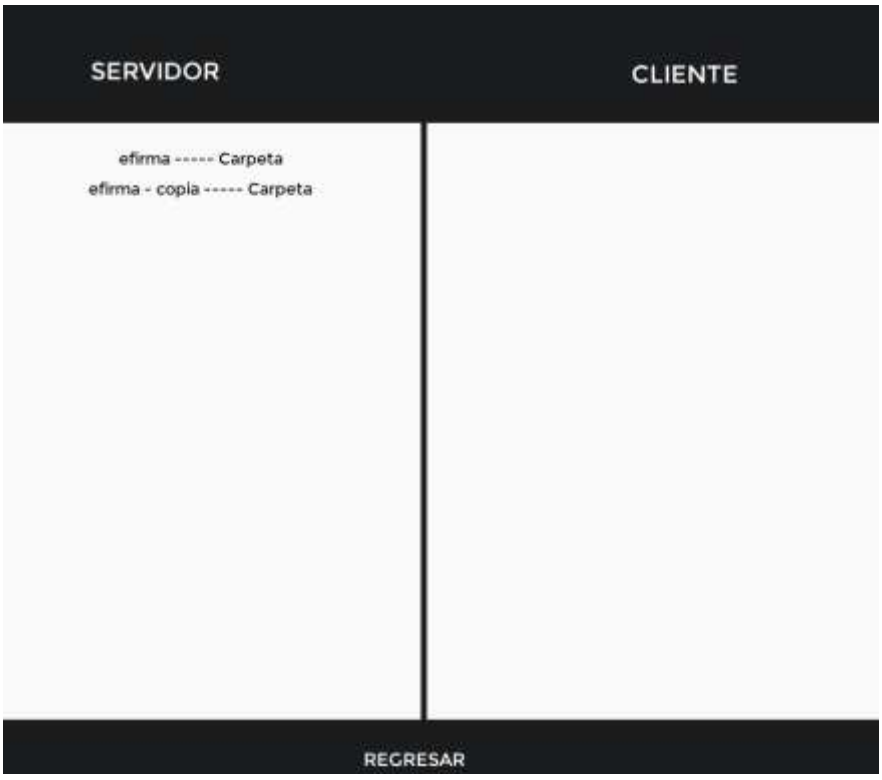

Ilustración 12. Listado de archivos enviados al servidor.

Ahora procedamos a descargar esos archivos descargados al cliente.

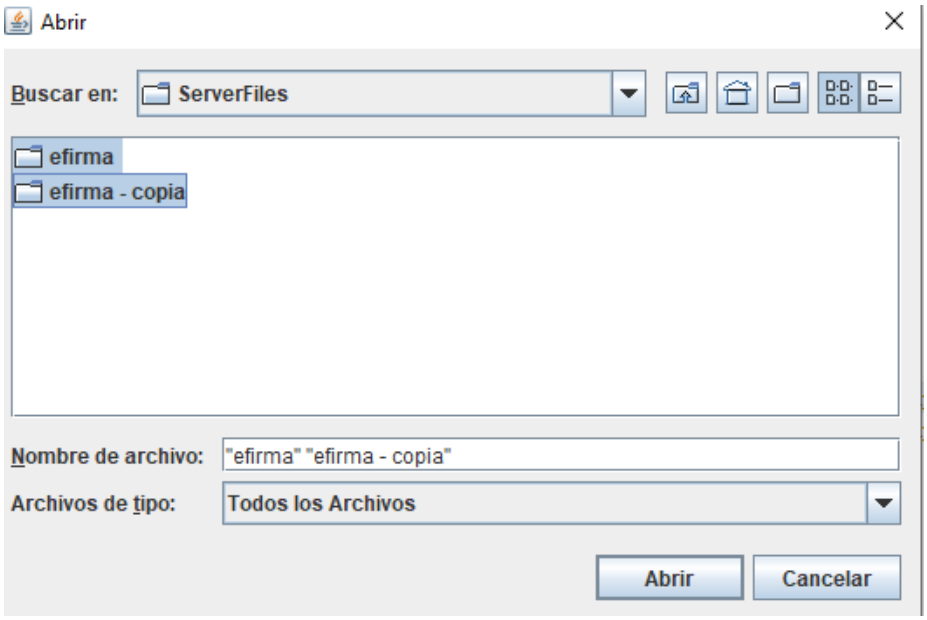

Ilustración 13. Descargando archivos.

Al ser descargados podemos listarlos de igual forma para verificar que se descargaron correctamente.

Ilustración 14. Listado de archivos descargados por el cliente.

Ahora eliminemos estos archivos, primeramente al seleccionar la parte de eliminar archivos del servidor se nos abre la ruta del servidor y podemos seleccionar lo que queremos eliminar como se ve en la ilustración.

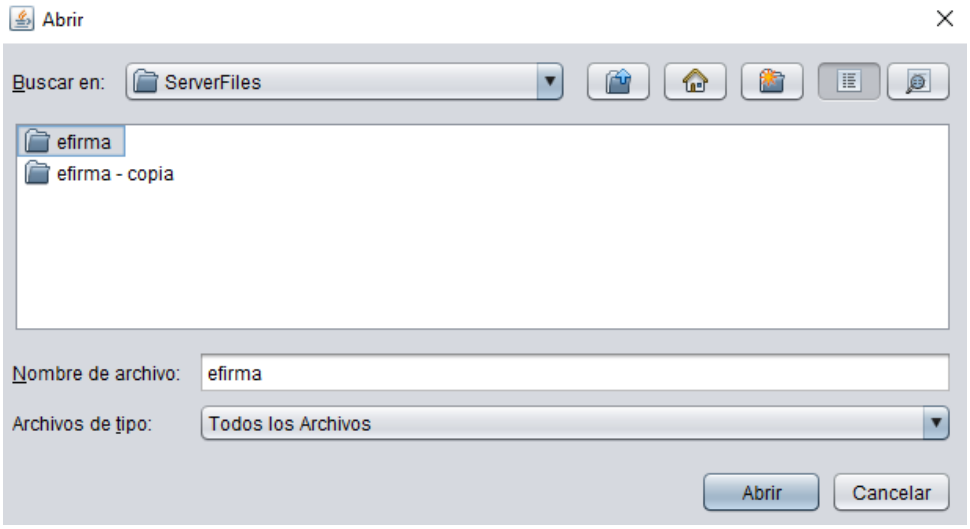

Ilustración 15. Selección de archivo para eliminar.

Al eliminarlo podemos listar de nuevo y veremos que ya no está.

Ilustración 16. Listando archivos.

Haremos lo mismo en el cliente pero eliminando un archivo distinto.

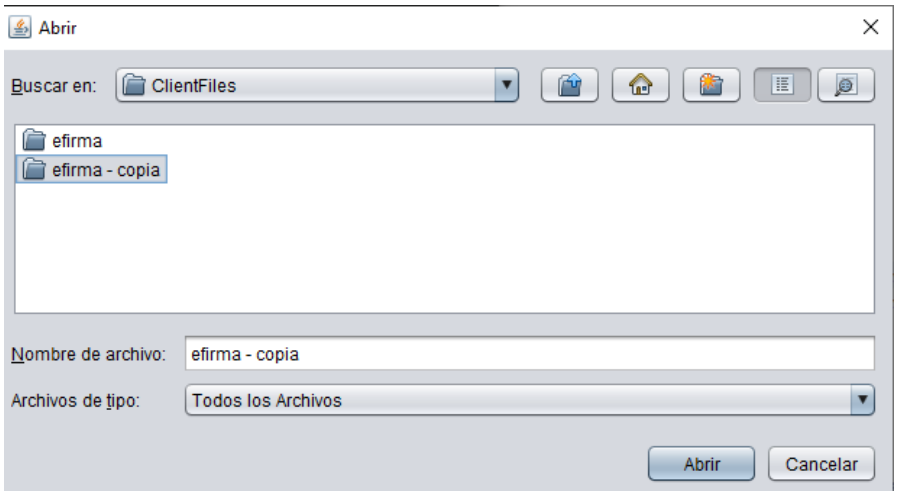

Ilustración 17. Archivo para eliminar del cliente.

Si listamos los archivos de nuevo vemos que ya no aparece

Ilustración 18. Listando archivos de nuevo

Se hicieron pruebas activando el algoritmo con la siguiente línea de código:

```
//*Nagle on
client.setTcpNoDelay(true);
```

Ilustración 19. Activando algoritmo de Nagle

Sin embargo, no se notaron en demasía los cambios que este algoritmo producía por lo tanto los efectos son los mismo al activarlo o no dentro de nuestro código.

Conclusiones

A través de la práctica realizada de envío de archivos usando sockets de flujo pude entender de mejor manera el funcionamiento de estos y la importancia de estos, pues lo presentado en esta práctica representa una de las muchas aplicaciones que se les puede dar a estos sockets, pues aquí solo vimos el funcionamiento elemental de como se pueden enviar archivos a través de un socket para llegar de un cliente a un servidor que se encuentran conectados y viceversa.

Durante el desarrollo de la práctica se presentaron algunas problemáticas, principalmente en el envío y recibo de carpetas ya que estos llegaban incompletos o se cerraba la conexión entre los sockets ya que nunca se terminaban de enviar o recibir y esto causaba una saturación en la conexión, esto se resolvió usando recursión, pues de esta forma se aseguró el envío correcto y completo de los datos.

Cabe mencionar que fue de mucha ayuda usar un entorno gráfico, pues este le da una mejor vista a la aplicación y en caso de que usuarios reales la usen se les facilitará el entendimiento y funcionamiento de esta.

Finalmente considero que fue una buena práctica para poner en marcha lo aprendido en las sesiones referentes a sockets de flujo, pues esta práctica abarco todo lo repasado en las clases.

Bibliografía

1) Buap. (2020). “Sockets”. Obtenido de : http://www.cs.buap.mx/~mtovar/doc/PCPA/Sockets.pdf