Ciudad de México a 19 de marzo de 2024

Instituto Politécnico Nacional

ESCUELA SUPERIOR DE CÓMPUTO

Unidad de Aprendizaje:

Aplicaciones para la Comunicación en Red

Práctica 1:

"Aplicación de Almacenamiento Remoto de Archivos"

Profesor: Axel Ernesto Moreno Cervantes

Grupo: 6CM2

Nombre de los alumnos:
García Cárdenas Ángel Alberto
Zúñiga Olguin Iván

Fecha de entrega:
19 de marzo de 2024

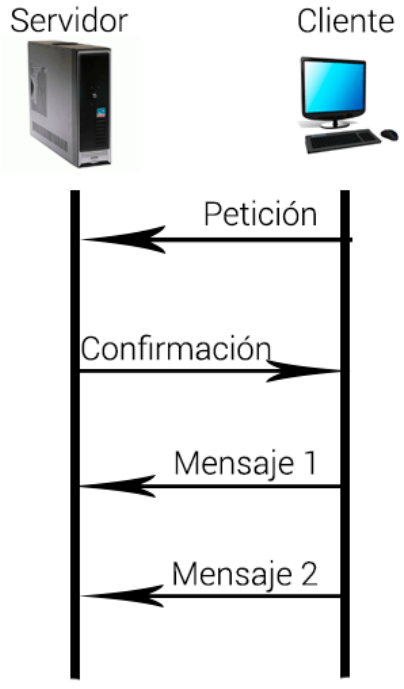
Ciudad de México a 19 de marzo de 2024

Introducción

En el panorama actual de la informática, el almacenamiento remoto de archivos ha pasado de ser una conveniencia a una necesidad básica. Este enfoque permite a los usuarios acceder y gestionar sus archivos desde cualquier lugar del mundo, sin estar limitados por la ubicación física de los datos. Este cambio ha sido impulsado por la creciente dependencia de la movilidad y la necesidad de colaboración en tiempo real en entornos distribuidos.

La práctica que hemos desarrollado se centra en el desarrollo de una aplicación de almacenamiento remoto de archivos utilizando Java. En este contexto, estamos empleando el paradigma de **sockets**, que proporciona una forma estándar de comunicación entre procesos a través de una red. Los sockets son puntos finales de una conexión de red bidireccional que permite la comunicación entre procesos en diferentes dispositivos. En el contexto de nuestra práctica, utilizamos sockets para establecer una conexión entre el cliente y el servidor, lo que permite la transferencia de datos de manera eficiente y segura a través de una red.

Los sockets TCP/IP proporcionan una comunicación fiable y orientada a la conexión, mientras que los sockets UDP ofrecen una comunicación sin conexión y más rápida, adecuada para aplicaciones donde la velocidad es prioritaria sobre la fiabilidad.



La aplicación consta de dos componentes principales: el cliente y el servidor. El cliente es la interfaz que los usuarios utilizan para interactuar con el sistema de almacenamiento remoto, mientras que el servidor es la parte central que gestiona las operaciones de almacenamiento y las solicitudes del cliente. Este enfoque cliente-servidor permite una arquitectura escalable y modular, donde múltiples clientes pueden acceder al mismo conjunto de datos de manera concurrente.

El cliente, implementado utilizando la biblioteca Swing de Java, proporciona una interfaz de usuario gráfica intuitiva y amigable. Los usuarios pueden realizar una variedad de acciones, como cargar archivos locales al servidor, descargar archivos

remotos al sistema local, crear y eliminar carpetas tanto local como remotamente, entre otras operaciones relacionadas con la gestión de archivos.

Por otro lado, el servidor es responsable de recibir las solicitudes del cliente, procesarlas y llevar a cabo las operaciones correspondientes en el sistema de archivos remoto. Utiliza sockets para comunicarse con el cliente y gestionar múltiples conexiones simultáneas de manera eficiente, garantizando un rendimiento óptimo incluso en entornos de alta carga.

Ciudad de México a 19 de marzo de 2024

Desarrollo

La aplicación de almacenamiento remoto de archivos desarrollada es una solución que permite a los usuarios gestionar archivos de manera remota a través de una conexión de red. La aplicación consta de dos componentes principales: un cliente y un servidor.

El cliente es una interfaz gráfica de usuario (GUI) que proporciona a los usuarios una forma intuitiva de interactuar con el sistema. Permite realizar diversas operaciones, como crear y eliminar carpetas locales y remotas, cambiar entre directorios locales y remotos, enviar archivos desde el cliente al servidor, y descargar archivos y carpetas desde el servidor al cliente.

El servidor es la parte central que gestiona las solicitudes del cliente y realiza las operaciones de almacenamiento de archivos. Se encarga de recibir las solicitudes del cliente, procesarlas y realizar las operaciones correspondientes en el sistema de archivos remoto. Esto incluye crear y eliminar carpetas, enviar archivos al cliente y recibir archivos del cliente para almacenarlos en el servidor.

La aplicación de almacenamiento remoto de archivos permite al usuario:

1. Listar el contenido de las carpetas local y remota
2. Crear carpetas local y remotamente
3. Eliminar carpetas/archivos local y remotamente
4. Cambiar la ruta del directorio loca/remoto
5. Enviar archivos/carpetas desde la carpeta local hacia la remota
6. Enviar archivos/carpetas desde la carpeta remota hacia la local
7. Salir de la aplicación

Imagen 1. Vista de la interfaz donde el usuario podrá elegir la acción a realizar.

Ciudad de México a 19 de marzo de 2024

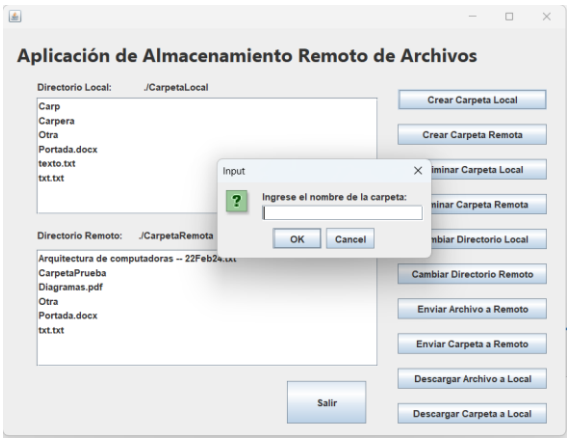


Imagen 2. Vista de la interfaz, el usuario crea una carpeta Local

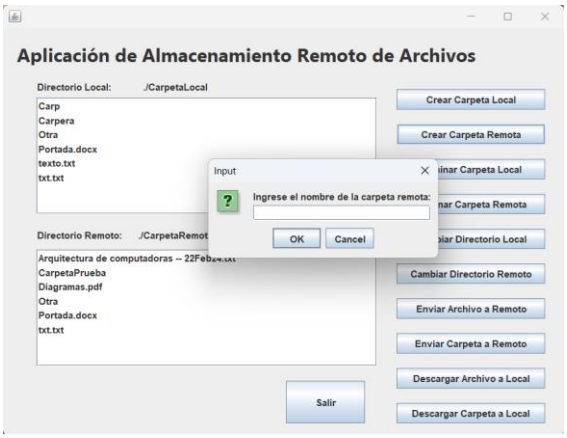


Imagen 3. Vista de la interfaz, el usuario crea una carpeta Remota

Imagen 4. Vista de la interfaz, el usuario selecciona y elimina carpeta Local

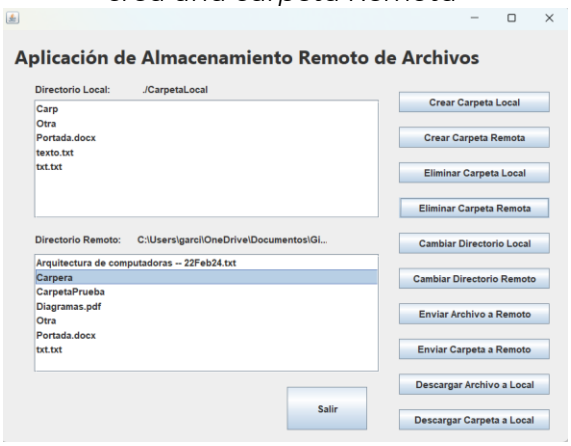


Imagen 5. Vista de la interfaz, el usuario selecciona y elimina carpeta Remota

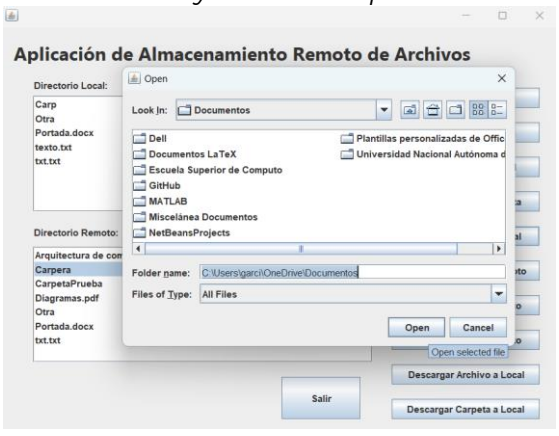


Imagen 6. Vista de la interfaz, el usuario cambia de directorio Local

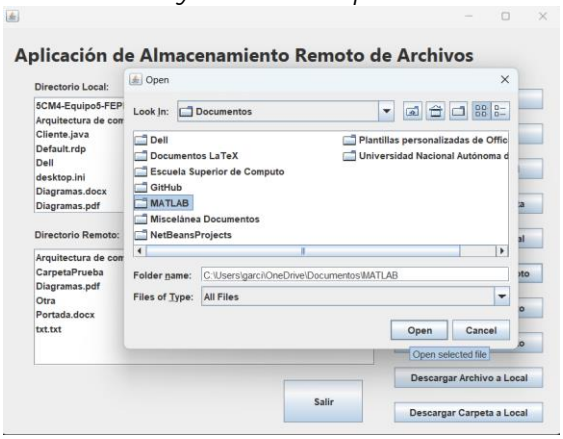


Imagen 7. Vista de la interfaz, el usuario cambia de directorio Remoto

Ciudad de México a 19 de marzo de 2024

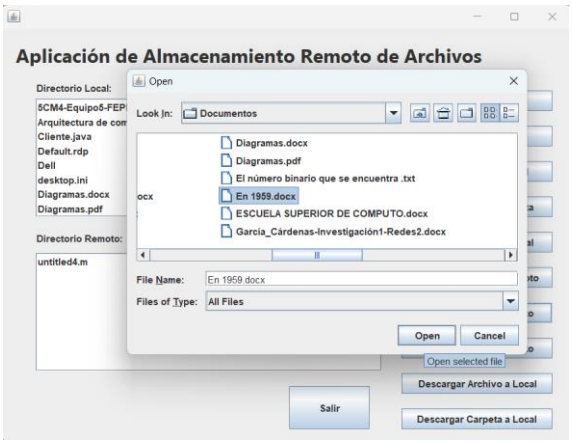


Imagen 8. Vista de la interfaz, el usuario sube archivo al servidor

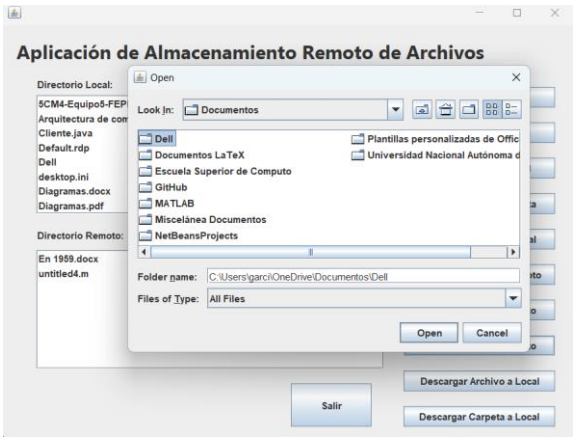


Imagen 9. Vista de la interfaz, el usuario sube carpeta al servidor

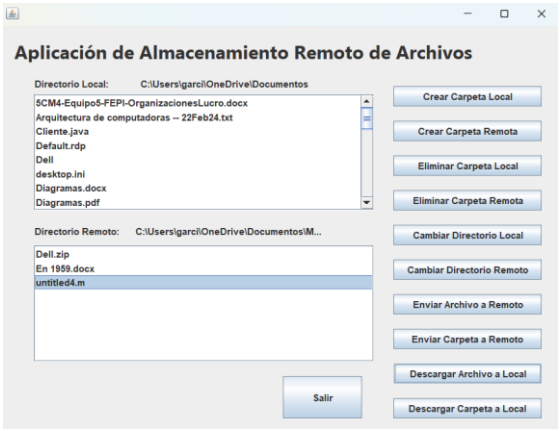


Imagen 10. Vista de la interfaz, el usuario descarga archivo

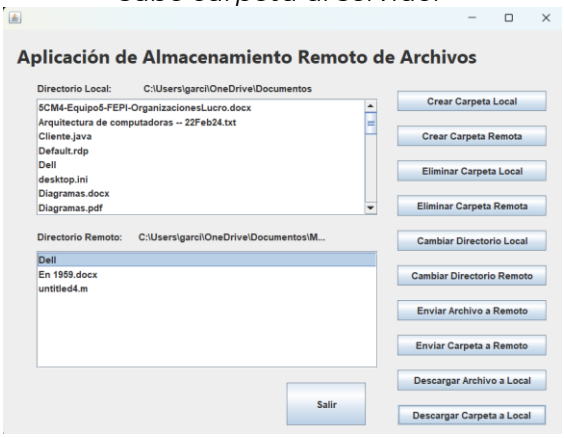


Imagen 11. Vista de la interfaz, el usuario descarga carpeta

Código Cliente.java:

```java
package Cliente;

import Servidor.CarpetaObjeto;
import java.net.Socket;
import java.io.*;
import java.util.Enumeration;
import java.util.Scanner;
import java.util.zip.*;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class Cliente {
    private Socket cl;
    private DataOutputStream dos;
    private DataInputStream dis;
    private ObjectOutputStream oos;
```

Ciudad de México a 19 de marzo de 2024

```
    private ObjectInputStream ois;
    private String rutaCarpetaLocal;
    private String rutaCarpetaRemota;

    public Cliente(String dir, int pto) {
        try {
            cl = new Socket(dir, pto);
            System.out.println("Conexión con servidor establecida...");

            dos = new DataOutputStream(cl.getOutputStream());
            dis = new DataInputStream(cl.getInputStream());
            oos = new ObjectOutputStream(cl.getOutputStream());
            ois = new ObjectInputStream(cl.getInputStream());

            rutaCarpetaLocal = "./CarpetaLocal";
            rutaCarpetaRemota = "./CarpetaRemota";


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

    public String getRutaCarpetaLocal() {
    return rutaCarpetaLocal;
    }

    public void listarContenidoLocal() {
        File localFolder = new File(rutaCarpetaLocal);
        String[] localContents = localFolder.list();
        for (String item : localContents) {
            System.out.println(item);
        }
    }

    public String[] listarContenidoRemoto() {
        try {
            dos.writeInt(1);
            dos.flush();

            System.out.println("Contenido de la carpeta remota:");

            CarpetaObjeto carpetaRecibida = (CarpetaObjeto) ois.readObject();
            String[] lista = carpetaRecibida.getLista();
            return lista;
```

Ciudad de México a 19 de marzo de 2024

```
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return new String[0];
    }

    public void crearCarpetaLocal() {
            String nombreCarpeta = JOptionPane.showInputDialog("Ingrese el
nombre de la carpeta:");
            if (nombreCarpeta != null && !nombreCarpeta.isEmpty()) {
                File newFolder = new File(rutaCarpetaLocal + "/" +
nombreCarpeta);
                if (newFolder.mkdir()) {
                    System.out.println("Carpeta creada exitosamente.");
                } else {
                    System.out.println("Error al crear la carpeta.");
                }
            } else {
                System.out.println("Nombre de carpeta inválido.");
            }
    }

    public void crearCarpetaRemota() {
        String nombreCarpetaRemota = JOptionPane.showInputDialog("Ingrese el
nombre de la carpeta remota:");
        if (nombreCarpetaRemota != null && !nombreCarpetaRemota.isEmpty()) {
            try {
                dos.writeInt(2);
                dos.flush();

                dos.writeUTF(nombreCarpetaRemota);
                dos.flush();

                String mensajeCreacion = dis.readUTF();
                System.out.println(mensajeCreacion);
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Nombre de carpeta remota inválido.");
        }
    }

    public void eliminarCarpetaLocal(String carpetaArchivoEliminar) {
```

Ciudad de México a 19 de marzo de 2024

```
        File itemEliminar = new File(rutaCarpetaLocal + "/" +
carpetaArchivoEliminar);
        if (itemEliminar.exists()) {
            if (itemEliminar.isDirectory()) {
                if (itemEliminar.delete()) {
                    System.out.println("Carpeta eliminada exitosamente.");
                } else {
                    System.out.println("Error al eliminar la carpeta.");
                }
            } else {
                if (itemEliminar.delete()) {
                    System.out.println("Archivo eliminado exitosamente.");
                } else {
                    System.out.println("Error al eliminar el archivo.");
                }
            }
        } else {
            System.out.println("El archivo/carpeta no existe.");
        }
    }

    public void eliminarCarpetaRemota(String carpetaArchivoEliminarRemoto) {
        try {
            dos.writeInt(3);
            dos.flush();

            dos.writeUTF(carpetaArchivoEliminarRemoto);
            dos.flush();

            String mensajeEliminacion = dis.readUTF();
            System.out.println(mensajeEliminacion);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void cambiarDirectorioLocal() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int seleccion = fileChooser.showOpenDialog(null);
        if (seleccion == JFileChooser.APPROVE_OPTION) {
            File selectedFolder = fileChooser.getSelectedFile();
            rutaCarpetaLocal = selectedFolder.getAbsolutePath();
            System.setProperty("user.dir", rutaCarpetaLocal);
            System.out.println("Ruta cambiada exitosamente.");
```

```
        }
    }

    public void cambiarDirectorioRemoto() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int seleccion = fileChooser.showOpenDialog(null);
        if (seleccion == JFileChooser.APPROVE_OPTION) {
            File selectedFolder = fileChooser.getSelectedFile();
            String nuevaRutaRemota = selectedFolder.getAbsolutePath();
            try {
                dos.writeInt(4);
                dos.flush();

                dos.writeUTF(nuevaRutaRemota);
                dos.flush();
                rutaCarpetaRemota = selectedFolder.getAbsolutePath();

                String mensajeCambioRuta = dis.readUTF();
                System.out.println(mensajeCambioRuta);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public String getRutaCarpetaRemota() {
        return rutaCarpetaRemota;
    }


    public void enviarArchivoRemoto(String rutaArchivo) {
        try {
            dos.writeInt(5);
            dos.flush();

            File archivoEnviar = new File(rutaArchivo);
            if (archivoEnviar.exists()) {
                String nombre = archivoEnviar.getName();
                long tam = archivoEnviar.length();
                System.out.println("Preparándose para enviar archivo " + nombre
+ " de " + tam + " bytes\n\n");
                dos.writeUTF(nombre);
                dos.flush();
                dos.writeLong(tam);
```

```
                dos.flush();
                try (FileInputStream fis = new FileInputStream(archivoEnviar))
{
                    byte[] buffer = new byte[3500];
                    int leidos;
                    while ((leidos = fis.read(buffer)) != -1) {
                        dos.write(buffer, 0, leidos);
                        dos.flush();
                    }
                    System.out.println("\nArchivo enviado.");
                } catch (IOException e) {
                    System.err.println("Error al enviar el archivo: " +
e.getMessage());
                }
            } else {
                System.out.println("El archivo no existe.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void enviarCarpetaRemota(String rutaCarpetaEnviar) {
        try {
            dos.writeInt(11); // Envía el código de operación al servidor
            dos.flush();

            File carpetaEnviar = new File(rutaCarpetaEnviar);
            if (carpetaEnviar.exists() && carpetaEnviar.isDirectory()) {
                String nombreCarpeta = carpetaEnviar.getName();
                File carpetaComprimida = new File(carpetaEnviar.getParent(),
nombreCarpeta + ".zip");

                try {
                    // Comprimir la carpeta en un archivo ZIP
                    FileOutputStream fos = new
FileOutputStream(carpetaComprimida);
                    ZipOutputStream zipOut = new ZipOutputStream(fos);
                    zipFile(carpetaEnviar, carpetaEnviar.getName(), zipOut);
                    zipOut.close();
                    fos.close();

                    // Obtener el tamaño del archivo ZIP
                    long tam = carpetaComprimida.length();
```

```
                    // Enviar nombre de carpeta y tamaño al servidor
                    dos.writeUTF(nombreCarpeta + ".zip");
                    dos.flush();
                    dos.writeLong(tam);
                    dos.flush();

                    // Enviar el contenido del archivo ZIP al servidor
                    try (DataInputStream disFile = new DataInputStream(new
FileInputStream(carpetaComprimida))) {
                        byte[] buffer = new byte[1500];
                        int leidos;
                        long enviados = 0;
                        int porcentaje;
                        while ((leidos = disFile.read(buffer)) != -1) {
                            dos.write(buffer, 0, leidos);
                            dos.flush();
                            enviados += leidos;
                            porcentaje = (int) ((enviados * 100) / tam);
                            System.out.print("\rEnviado el " + porcentaje + "%
del archivo");
                        }
                        System.out.println("\nCarpeta enviada.");
                    } catch (IOException e) {
                        System.err.println("Error al enviar la carpeta: " +
e.getMessage());
                    }
                } catch (IOException e) {
                    System.err.println("Error al comprimir la carpeta: " +
e.getMessage());
                } finally {
                    // Eliminar el archivo ZIP después de enviar la carpeta
                    carpetaComprimida.delete();
                }
            } else {
                System.out.println("La carpeta no existe o no es válida.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

        public static void zipFile(File fileToZip, String fileName,
ZipOutputStream zipOut) throws IOException {
        if (fileToZip.isHidden()) {
            return;
```

```
        }
        if (fileToZip.isDirectory()) {
            if (fileName.endsWith("/")) {
                zipOut.putNextEntry(new ZipEntry(fileName));
                zipOut.closeEntry();
            } else {
                zipOut.putNextEntry(new ZipEntry(fileName + "/"));
                zipOut.closeEntry();
            }
            File[] children = fileToZip.listFiles();
            for (File childFile : children) {
                zipFile(childFile, fileName + "/" + childFile.getName(),
zipOut);
            }
            return;
        }
        FileInputStream fis = new FileInputStream(fileToZip);
        ZipEntry zipEntry = new ZipEntry(fileName);
        zipOut.putNextEntry(zipEntry);
        byte[] bytes = new byte[1024];
        int length;
        while ((length = fis.read(bytes)) >= 0) {
            zipOut.write(bytes, 0, length);
        }
        fis.close();
    }

    public void descargarArchivoRemoto(int posicionArchivo) {
        try {
            dos.writeInt(7);
            dos.flush();

            dos.writeInt(posicionArchivo);
            dos.flush();

            String nombre = dis.readUTF();
            long tam = dis.readLong();
            System.out.println("Comienza descarga del archivo " + nombre + " de
" + tam + " bytes\n\n");

            File archivoDestino = new File(rutaCarpetaLocal + File.separator +
nombre);

            try (FileOutputStream fos = new FileOutputStream(archivoDestino);
BufferedOutputStream bos = new BufferedOutputStream(fos)) {
```

```
                byte[] buffer = new byte[1500];
                int leidos, totalLeidos = 0;

                while (totalLeidos < tam && (leidos = dis.read(buffer, 0, (int)
Math.min(buffer.length, tam - totalLeidos))) != -1) {
                    bos.write(buffer, 0, leidos);
                    totalLeidos += leidos;
                }
                bos.flush();
                System.out.println("Archivo recibido correctamente: " +
nombre);
            } catch (IOException e) {
                System.err.println("Error al recibir el archivo " + nombre + ":
" + e.getMessage());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void descargarCarpetaRemoto(int indiceArchivo, String rutaDescarga)
{
        try {
            dos.writeInt(8);
            dos.flush();

            dos.writeInt(indiceArchivo);
            dos.flush();

            String nombreArchivoZip = dis.readUTF();
            long tam = dis.readLong();

            if (tam > 0) {
                System.out.println("Preparándose para recibir la carpeta de " +
tam + " bytes\n\n");
                File carpetaRecibida = new File(rutaDescarga + "/" +
nombreArchivoZip);
                try (FileOutputStream fos = new
FileOutputStream(carpetaRecibida);
                     BufferedOutputStream bos = new BufferedOutputStream(fos))
{
                    byte[] buffer = new byte[3500];
                    int leidos;
                    while (tam > 0 && (leidos = dis.read(buffer, 0, (int)
Math.min(buffer.length, tam))) != -1) {
```

```
                        bos.write(buffer, 0, leidos);
                        tam -= leidos;
                    }
                    System.out.println("\nCarpeta recibida.");
                } catch (IOException e) {
                    System.err.println("Error al recibir la carpeta: " +
e.getMessage());
                }

                // Descomprimir el archivo ZIP
                try (ZipFile zipFile = new ZipFile(carpetaRecibida)) {
                    Enumeration<? extends ZipEntry> entries =
zipFile.entries();
                    while (entries.hasMoreElements()) {
                        ZipEntry entry = entries.nextElement();
                        File entryDestination = new File(rutaDescarga,
entry.getName());
                        if (entry.isDirectory()) {
                            entryDestination.mkdirs();
                        } else {
                            try (InputStream is =
zipFile.getInputStream(entry);
                                 OutputStream os = new
FileOutputStream(entryDestination)) {
                                byte[] buffer = new byte[4096];
                                int bytesRead;
                                while ((bytesRead = is.read(buffer)) != -1) {
                                    os.write(buffer, 0, bytesRead);
                                }
                            }
                        }
                    }
                } catch (IOException e) {
                    System.err.println("Error al descomprimir la carpeta: " +
e.getMessage());
                }

                // Eliminar el archivo ZIP
                carpetaRecibida.delete();
                System.out.println("Archivo ZIP eliminado después de la
extracción.");
            } else {
                System.out.println("La carpeta no existe.");
            }
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }

    public void cerrarConexion() {
        try {
            dos.close();
            oos.close();
            dis.close();
            ois.close();
            cl.close();
            System.out.println("Conexión con servidor cerrada.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        try {
            int pto = 8000;
            String dir = "127.0.0.1";
            Cliente cliente = new Cliente(dir, pto);

            // Crear la interfaz de usuario y conectar los eventos de los
botones
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
                    new Menu(cliente).setVisible(true);
                }
            });
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Acerca de la implementación del cliente:
El cliente de sockets interactúa con un servidor remoto para realizar diversas operaciones de gestión de archivos y carpetas, como listar contenido local y remoto, crear y eliminar carpetas tanto local como remotamente, cambiar directorios local y remoto, enviar archivos y carpetas al servidor, y descargar archivos y carpetas desde el servidor. A continuación, se menciona brevemente cómo funcionan:

1. *Constructor:*
- El constructor establece la conexión con el servidor utilizando la dirección

IP y el puerto proporcionado.
- Inicializa los flujos de entrada y salida de datos para la comunicación con el servidor.
- Define las rutas de las carpetas locales y remotas.

2. *Métodos para listar contenido:*
- listarContenidoLocal(): Utiliza la clase File de Java para obtener la lista de archivos y carpetas en la carpeta local especificada por rutaCarpetaLocal. Luego, simplemente imprime los nombres de los elementos en la consola.
- listarContenidoRemoto(): Envía un código de operación al servidor para solicitar la lista de archivos y carpetas en la carpeta remota. Luego, espera a recibir un objeto CarpetaObjeto del servidor, que contiene la lista, y devuelve los nombres de los elementos.

3. *Métodos para crear carpetas:*
- crearCarpetaLocal(): Utiliza la clase File para crear una nueva carpeta en la ruta local especificada por el usuario. Verifica si la creación fue exitosa o no, imprime un mensaje acerca de la salida.
- crearCarpetaRemota(): Envía una solicitud al servidor para crear una carpeta en la ruta remota.

4. *Métodos para eliminar carpetas:*
- eliminarCarpetaLocal(): Utiliza la clase File para eliminar la carpeta o archivo especificado en la ruta local. Verifica si la eliminación fue exitosa o no, posteriormente imprime un mensaje correspondiente.
- eliminarCarpetaRemota(): Envía una solicitud al servidor para eliminar una carpeta en la ruta remota.

5. *Métodos para cambiar directorios:*
- cambiarDirectorioLocal(): Permite al usuario seleccionar una nueva carpeta local.
- cambiarDirectorioRemoto(): Permite al usuario seleccionar una nueva carpeta remota y notifica al servidor sobre el cambio.

6. *Métodos para enviar archivos y carpetas al servidor:*
- enviarArchivoRemoto(): Envía un código de operación al servidor para indicar que se enviará un archivo. Luego, envía el nombre y tamaño del archivo, seguido por los bytes del archivo en sí.
- enviarCarpetaRemota(): Comprime la carpeta local en un archivo ZIP, luego envía un código de operación al servidor para indicar que se enviará una carpeta. Envía el nombre y tamaño del archivo ZIP, seguido por los bytes del archivo ZIP.

7. *Métodos para descargar archivos y carpetas desde el servidor:*
- descargarArchivoRemoto(): Envía un código de operación al servidor para indicar que se desea descargar un archivo. Luego, envía el índice del archivo al servidor y espera a recibir el nombre y tamaño del archivo, seguido por los bytes del archivo en sí.

- descargarCarpetaRemota(): Envía un código de operación al servidor para indicar que se desea descargar una carpeta. Luego, envía el índice de la carpeta al servidor y espera a recibir el nombre y tamaño del archivo ZIP que contiene la carpeta, seguido por los bytes del archivo ZIP. Después de recibir la carpeta comprimida, la descomprime en la ruta local especificada por el usuario.

8. *Método para cerrar la conexión:*

- cerrarConexion(): Cierra los flujos de entrada y salida de datos y cierra la conexión con el servidor.

El código cliente.java está diseñado para ser interactivo y proporciona una interfaz gráfica para que el usuario realice las operaciones de gestión de archivos y carpetas de manera intuitiva. La interfaz de usuario se ha separado en la clase llamada Menu.java. El diseño modular y la separación de responsabilidades hacen que el código sea más mantenible y escalable.

Ciudad de México a 19 de marzo de 2024

Conclusiones

Angel Alberto García Cárdenas: En conclusión, el desarrollo del cliente ha sido un proceso integral donde se ha logrado implementar una aplicación versátil y robusta. Hemos construido una interfaz que ofrece al usuario una experiencia fluida y amigable al interactuar con el servidor remoto. Mediante la utilización de sockets y flujos de datos, se logró establecido una comunicación eficiente con el servidor, permitiendo la transferencia de información de manera segura y confiable. Hemos integrado una amplia gama de funcionalidades que abarcan desde la gestión básica de archivos y carpetas hasta la compresión y transferencia de grandes volúmenes de datos.

Iván Zúñiga Olguin: En conclusión, el desarrollo del servidor ha sido un proceso de gran envergadura en el que se han implementado diversas funcionalidades para gestionar eficientemente archivos y carpetas de forma remota. Se estableció una comunicación fluida con los clientes a través de sockets, lo que nos ha permitido intercambiar datos de manera segura y confiable. Se incorporaron características avanzadas como la compresión y descompresión de archivos, así como la capacidad de crear, eliminar y renombrar carpetas de manera remota. Esto proporciona a los usuarios una amplia gama de herramientas para gestionar sus archivos de forma efectiva.

Ciudad de México a 19 de marzo de 2024

Bibliografía

Calvert, K. L., & Donahoo, M. J. (2009). TCP/IP Sockets in Java: Practical Guide for Programmers (2nd ed.). Morgan Kaufmann.

Tanenbaum, A. S., & Wetherall, D. J. (2011). Redes de computadoras (5.a ed.). Pearson Educación.