

Instituto Politécnico Nacional

Escuela Superior de Cómputo

*Aplicaciones para comunicaciones de red*

*“Practica 1 - Aplicación estilo DropBox”*

**Grupo:** 3CM15

**Integrantes:**

* Ramírez Olvera Guillermo.
* Sánchez Méndez Edmundo Josue.

**Fecha de entrega:** 11 de marzo de 2021

**Profesor:** Moreno Cervantes Axel Ernesto

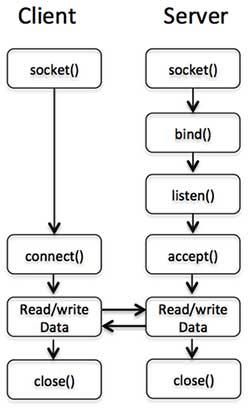


En la actualidad, muchos de los procesos que se ejecutan en una computadora requieren obtener o enviar información a otros procesos que se localizan en una computadora diferente. Para lograr esta comunicación se utilizan los protocolos de comunicación TCP y UDP.

El protocolo TCP (Transmission Control Protocol) establece un conducto de comunicación punto a punto entre dos computadoras, es decir, cuando se requiere la transmisión de un flujo de datos entre dos equipos, el protocolo TCP establece un conducto exclusivo entre dichos equipos por el cual los datos serán transmitidos y este perdurará hasta que la transmisión haya finalizado, gracias a esto TCP garantiza que los datos enviados de un extremo de la conexión lleguen al otro extremo y en el mismo orden en que fueron enviados. Las características que posee TCP hacen que el protocolo sea conocido como un protocolo orientado a conexión.

Los sockets son una forma de comunicación entre procesos que se encuentran en diferentes máquinas de una red, los sockets proporcionan un punto de comunicación por el cual se puede enviar o recibir información entre procesos.

Los sockets tienen un ciclo de vida diferente dependiendo si son sockets de servidor, que esperan a un cliente para establecer una comunicación, o socket cliente que busca a un socket de servidor para establecer la comunicación.



**Desarrollo de la practica:**

En esta práctica se implementará una aplicación estilo Google Drive, que permitirá al usuario ver los archivos que tiene en su carpeta, así como subir o descargar archivos/carpetas a su carpeta del servidor mediante una caja de dialogo.

Para la implementación de esta practica se realizaron 3 clases en java: DropBox, Cliente y Servidor.

**Clase DropBox:**

Se construye la interfaz grafica por medio de Java Swing, en donde el usuario esta viendo constantemente la representación de su carpeta personal con su contenido en el servidor.

El listado de archivos y carpetas de la carpeta personal del cliente, almacenada en el servidor, son desplegados por medio de un JList. Si se trata de un directorio, al dar doble clic sobre el mismo se abrirá este, mostrando su contenido.

Para descargar uno o múltiples archivos solo se deben seleccionar estos en el JList y pulsar el botón ”Descargar archivos”, estos se guardarán como un archivo ZIP.

1. **public** **class** DropBox **extends** [JFrame](http://www.google.com/search?hl=en&q=allinurl%3Ajframe+java.sun.com&btnI=I%27m%20Feeling%20Lucky) **implements** [ActionListener](http://www.google.com/search?hl=en&q=allinurl%3Aactionlistener+java.sun.com&btnI=I%27m%20Feeling%20Lucky) {
3. [JButton](http://www.google.com/search?hl=en&q=allinurl%3Ajbutton+java.sun.com&btnI=I%27m%20Feeling%20Lucky) BtnSubir, BtnActualizar, BtnDescargar, BtnEliminar;
4. **static** JList<String> archivos;
5. **static DefaultListModel<String> modelo;**
6. [MouseListener](http://www.google.com/search?hl=en&q=allinurl%3Amouselistener+java.sun.com&btnI=I%27m%20Feeling%20Lucky) mouseListener;
7. [JPanel](http://www.google.com/search?hl=en&q=allinurl%3Ajpanel+java.sun.com&btnI=I%27m%20Feeling%20Lucky) panelBotones;
8. **static** [JProgressBar](http://www.google.com/search?hl=en&q=allinurl%3Ajprogressbar+java.sun.com&btnI=I%27m%20Feeling%20Lucky) BarraProgreso;
9. [JScrollPane](http://www.google.com/search?hl=en&q=allinurl%3Ajscrollpane+java.sun.com&btnI=I%27m%20Feeling%20Lucky) scroll;
10. ***//File list[];***
12. **public** DropBox() {
13. [Container](http://www.google.com/search?hl=en&q=allinurl%3Acontainer+java.sun.com&btnI=I%27m%20Feeling%20Lucky) c = getContentPane();
14. c.setLayout(**new** [BoxLayout](http://www.google.com/search?hl=en&q=allinurl%3Aboxlayout+java.sun.com&btnI=I%27m%20Feeling%20Lucky)(c, [BoxLayout](http://www.google.com/search?hl=en&q=allinurl%3Aboxlayout+java.sun.com&btnI=I%27m%20Feeling%20Lucky).Y\_AXIS));
16. archivos = **new** JList<String>();
17. archivos.setSelectionMode([ListSelectionModel](http://www.google.com/search?hl=en&q=allinurl%3Alistselectionmodel+java.sun.com&btnI=I%27m%20Feeling%20Lucky).MULTIPLE\_INTERVAL\_SELECTION);
19. */\*Añadimos la funcionalidad de doble clic para navegar por directorios\*/*
20. **mouseListener = new**[**MouseAdapter**](http://www.google.com/search?hl=en&q=allinurl%3Amouseadapter+java.sun.com&btnI=I%27m%20Feeling%20Lucky)**() {**
21. **public** **void** mouseClicked([MouseEvent](http://www.google.com/search?hl=en&q=allinurl%3Amouseevent+java.sun.com&btnI=I%27m%20Feeling%20Lucky) e) {
23. **if** (e.getClickCount() == 2) {
24. **int** index = archivos.locationToIndex(e.getPoint());
25. [**String**](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky)**nombreSeleccion = modelo.getElementAt(index);**
27. *//Revisamos que la seleccion sea un directorio*
28. **if** (Cliente.tipoFile[index] == 1) {
29. modelo.clear();
30. **Cliente.AbrirCarpeta(index);**
31. }
32. }
34. }
35. **};**

**Clase Cliente:**

Es la clase encargada de solicitar todas las peticiones al servidor, y pasarle las respuestas a la clase DropBox, que la ira desplegando de forma gráfica para el usuario.

Primero inicializamos el puerto.

1. **public** **class** Cliente {
3. **private** **static** **int** pto = 4444;
4. **private** **static** [String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) host = "127.0.0.1";
5. **private static**[**String**](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky)**rutaDirectorios = "";**
6. **public** **static** [String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) sep = [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).getProperty("file.separator");
7. **public** **static** **int**[] tipoFile;

Toda la funcionalidad de la clase DropBox se encuentra en los métodos de la clase cliente, dichos métodos son:

void Actualizar ()

1. **public** **static** **void** Actualizar() {
2. **try** {
3. Socket cl = **new** Socket(host, pto);
4. DataOutputStream dos = **new** DataOutputStream(cl.getOutputStream()); *//OutputStream*
6. *//La bandera tiene el valor de 1 = Actualizar*
7. dos.writeInt(1);
8. dos.flush();
10. **DataInputStream dis = new DataInputStream(cl.getInputStream()); *// InputStream***
12. **int** numArchivos = dis.readInt();
13. tipoFile = **new** **int**[numArchivos];
15. **for (int i = 0; i < numArchivos; i++) {**
16. String archivoRecibido = dis.readUTF();
17. DropBox.modelo.addElement(archivoRecibido);
18. tipoFile[i] = dis.readInt();
19. }
21. dis.close();
22. dos.close();
23. cl.close();
24. System.out.println("Carpeta del cliente actualizada.");
26. } **catch** (Exception e) {
27. e.printStackTrace();
28. }
29. }

void RecibirArchivos (nombresArchivos[], tamaño)

Después de seleccionar uno o múltiples archivos y/o carpetas en el JList en la interfaz gráfica de la clase DropBox y dar clic en el botón ”Descargar Archivos”, por medio de un socket se envía una bandera entera con valor 2 al servidor, y luego enviamos el número de archivos/directorios a descargar junto a su índice del JList para indicarle al servidor de que archivos se tratan.

Se reciben un archivo ZIP con los archivos solicitados, leyéndolo con ayuda del socket y de un ´flujo de entrada, y escribiendo en el disco duro del cliente byte a byte con ayuda de un flujo de salida.

1. **public** **static** **void** RecibirArchivos(String[] nombresArchivos, **int** tama) {
2. **try** {
3. Socket cl = **new** Socket(host, pto);
4. DataOutputStream dos = **new** DataOutputStream(cl.getOutputStream()); *//OutputStream*
5. **DataInputStream dis = new DataInputStream(cl.getInputStream()); *// InputStream***
7. *//La bandera tiene el valor de 2 = Descargar seleccion*
8. dos.writeInt(2);
9. dos.flush();
11. dos.writeInt(tama);
12. dos.flush();
14. *//Enviamos los indices de los archivos seleccionados*
15. **String aux = "";**
17. **for** (**int** i = 0; i < tama; i++) {
18. aux = nombresArchivos[i];
19. dos.writeUTF(aux);
20. **dos.flush();**
21. }
23. String nombre = System.getProperty("user.home") + "/Desktop/";
25. **nombre = nombre + dis.readUTF();**
27. **long** tam = dis.readLong();
28. System.out.println("**\n**Se recibe el archivo " + nombre + " con " + tam + "bytes");
30. **DataOutputStream dosArchivo = new DataOutputStream(new FileOutputStream(nombre)); *// OutputStream***
32. **long** recibidos = 0;
33. **int** n = 0, porciento = 0;
34. **byte**[] b = **new** **byte**[2000];
36. **while** (recibidos < tam) {
37. n = dis.read(b);
38. dosArchivo.write(b, 0, n);
39. dosArchivo.flush();
40. **recibidos += n;**
41. porciento = (**int**) ((recibidos \* 100) / tam);
42. System.out.println("**\r** Recibiendo el " + porciento + "% --- " + recibidos + "/" + tam + " bytes");
43. }
45. **JOptionPane.showMessageDialog(null, "Se ha descargado el archivo " + nombre + " con tamanio: " + tam);**
46. dos.close();
47. dis.close();
48. dosArchivo.close();
49. cl.close();
51. } **catch** (Exception e) {
52. e.printStackTrace();
53. }
54. }

void AbrirCarpeta(indice):

Al hacer doble clic en un directorio dentro del Jlist que simboliza nuestra carpeta personal en el servidor, por medio de un socket de flujo bloqueante se envía una banda entera con valor 2 al servidor, luego el índice que corresponde al directorio seleccionado.

1. **public** **static** **void** AbrirCarpeta(**int** indice) {
2. **try** {
3. Socket cl = **new** Socket(host, pto);
4. DataOutputStream dos = **new** DataOutputStream(cl.getOutputStream()); *//OutputStream*
6. *//La bandera tiene el valor de 3 = AbrirCarpeta*
7. dos.writeInt(3);
8. dos.flush();
10. ***//Enviamos el indice en donde se encuentra la carpeta dentro del arreglo de Files[]***
11. dos.writeInt(indice);
12. dos.flush();
14. DataInputStream dis = **new** DataInputStream(cl.getInputStream()); *// InputStream*
16. **int** numArchivos = dis.readInt();
17. tipoFile = **new** **int**[numArchivos];
19. **for** (**int** i = 0; i < numArchivos; i++) {
20. **String archivoRecibido = dis.readUTF();**
21. DropBox.modelo.addElement(archivoRecibido);
22. tipoFile[i] = dis.readInt();
23. }*//for*
25. **dis.close();**
26. dos.close();
27. cl.close();
28. System.out.println("Nueva carpeta abierta: Request recibido.");
30. **} catch (Exception e) {**
31. e.printStackTrace();
32. }*//catch*
33. }

void SeleccionarArchivo()

Al hacer clic en el botón ”Subir Archivos” de la clase DropBox, este método primero despliega un JFileChooser, para elegir desde nuestro ordenador (cliente) los archivos o directorios que deseemos subir al servidor.

Si se va a subir una carpeta, el cliente envía una bandera entera con valor 4 al servidor, si se va a subir un archivo, la bandera es 0.

El cliente se encarga de decirle al servidor como ira guardando las carpetas o archivos en la carpeta personal, ubicada en este último. Una vez seleccionado uno o múltiples archivos, el cliente los enviará byte por byte con ayuda de un socket y un DataOutputStream.

1. **public** **static** **void** SeleccionarArchivos() {
2. **try** {
3. JFileChooser jf = **new** JFileChooser();
4. jf.setMultiSelectionEnabled(**true**);
5. **jf.setFileSelectionMode(JFileChooser.FILES\_AND\_DIRECTORIES);**
6. **int** r = jf.showOpenDialog(**null**);
8. **if** (r == JFileChooser.APPROVE\_OPTION) {
9. rutaDirectorios = "";
10. **File[] files = jf.getSelectedFiles();**
11. **for** (File file : files) {
12. String rutaOrigen = file.getAbsolutePath();
13. *// Tipo caso base: La primera vez que mandemos un archivo*
14. *// Siempre estará en la raíz del servidor*
15. **EnviarArchivo(file, rutaOrigen, file.getName());**
16. }*//for*
17. DropBox.modelo.clear();
18. Actualizar();
19. }*//if*
20. **} catch (Exception e) {**
21. e.printStackTrace();
22. }
23. }

eliminar archivo(nombresArchivos, tama)

Se eliminan los archivos que se mandaron en el arreglo de los nombres de archivos uno por uno.

1. **public** **static** **void** EliminarArchivo(String[] nombresArchivos, **int** tama) {
2. **try** {
3. Socket cl = **new** Socket(host, pto);
4. DataOutputStream dos = **new** DataOutputStream(cl.getOutputStream()); *//OutputStream*
5. **DataInputStream dis = new DataInputStream(cl.getInputStream()); *// InputStream***
7. dos.writeInt(5);
8. dos.flush();
10. **dos.writeInt(tama);**
11. dos.flush();
13. *//Enviamos los indices de los archivos seleccionados*
14. String aux = "";
16. **for** (**int** i = 0; i < tama; i++) {
17. aux = nombresArchivos[i];
18. dos.writeUTF(aux);
19. dos.flush();
20. **}**
22. String nombre = dis.readUTF();
23. **long** tam = dis.readLong();
24. JOptionPane.showMessageDialog(**null**, "Se ha eliminado el archivo o carptea " + nombre + " con tamanio: " + tam);
26. dos.close();
27. dis.close();
28. cl.close();
30. **} catch (Exception e) {**
31. e.printStackTrace();
32. }
33. }

**Clase servidor**

Esta clase es la encargada de atender las peticiones del cliente, procesarlas y enviarle las respuestas é información.

Primero se inicializa el directorio en donde se guardaran los archivos del cliente, el cual es ./serverP1.

También inicializaremos un arreglo de tipo File para ir guardando toda la información de los archivos en esta localización y los de sus subdirectorios, un String con la ruta actual de trabajo en el servidor, y un contador para llevar un control con los archivos solicitados para su descarga en ZIPs numerados.

1. **public** **class** Servidor {
3. **public** **static** String sep = System.getProperty("file.separator");
4. **private** **static** String rutaServer = "." + sep + "serverP1" + sep;
5. **private static File[] list;**
6. **private** **static** String rutaActual = "";
7. **private** **static** **int** numVeces = 0;

main

Aquí es donde todas las peticiones, métodos y respuestas interactúan.

Primero, creamos un ServerSocket, asignándole el mismo puerto que se asignó en el socket del cliente, y luego creamos un ciclo infinito para estar siempre escuchando a los clientes que se conecten en cualquier momento (creando un socket nuevo de respuesta con el metodo accept()). también creamos los flujos de entrada y salida del socket.

1. **public** **static** **void** main(String[] args) {
2. **try** {
3. ServerSocket s = **new** ServerSocket(4444);
4. s.setReuseAddress(**true**);
5. **System.out.println("Servidor de archivos iniciado, esperando cliente...");**
7. *// Espera clientes*
8. **for** (;;) {
9. Socket cl = s.accept();
10. **System.out.println("\n\nCliente conectado desde " + cl.getInetAddress() + " " + cl.getPort());**
11. DataOutputStream dos = **new** DataOutputStream(cl.getOutputStream()); *//OutputStream*
12. DataInputStream dis = **new** DataInputStream(cl.getInputStream()); *// InputStream*
14. **int** bandera = dis.readInt();

Ahora bien, leemos una bandera de tipo entero, que nos ayuda a regular las acciones que tomara el servidor según la acción que se realizó en el cliente DropBox:

*La bandera vale 0:* El cliente va a subir archivos desde su equipo, por lo que el servidor debe recibirlos y guardarlos en disco duro (en la carpeta antes mencionada).

*La bandera vale 1:* El usuario actualizo su carpeta en la interfaz gráfica del cliente.

*La bandera vale 2:* El cliente va a descargar archivos y/o carpetas seleccionadas en la interfaz gráfica, por lo que el servidor debe leerlos del disco, prepararlos, comprimirlos y enviarlos al cliente.

1. **else** **if** (bandera == 2) {
2. *//Descargar archivos -> El servidor prepara y envia archivos*
3. *//Subir archivos -> El servidor recibe*
4. **int** tam = dis.readInt();
5. **String path = "Download" + numVeces + ".zip";**
6. path = rutaServer + path;
7. System.out.println("" + path);
8. File archivoZip = **new** File(path);
9. System.out.println("" + archivoZip.getAbsoluteFile());
11. crearZIP(dis, tam);
13. **if** (archivoZip.exists()) {
14. *//System.out.println("Si existeee");*
15. **System.out.println("La path del archivo esta en: " + path + " Con nombre: " + archivoZip.getName());**
16. EnviarArchivo(dos, archivoZip);
17. *// Lo elimino porque no debe estar en el servidor, solo lo hice temporalmente*
18. **if** (archivoZip.delete()) {
19. System.out.println("Archivo temporal Download" + numVeces + ".zip eliminado");
20. **}**
21. }
23. numVeces++;
25. **}**

*La bandera vale 3:* El usuario dio doble clic en un directorio de su carpeta en la interfaz gráfica del cliente.

1. **else** **if** (bandera == 3) {
2. *//Abrir carpeta -> El servidor envia los nombres de los contenidos de la carpeta seleccionada*
3. **int** ubicacionRuta = dis.readInt();
4. *//Bandera = 1. Se navega dentro de una carpeta*
5. **String nuevaRuta = "" + list[ubicacionRuta].getAbsoluteFile();**
6. ActualizarCliente(cl, dis, nuevaRuta, 1);
7. }

*La bandera vale 4:* El cliente va a subir carpetas desde su equipo, por lo que el servidor primero debe leer el nombre de estas, crearlas dentro de la carpeta del servidor (en disco) y actualizar la ruta actual de trabajo.

1. **else** **if** (bandera == 4) {
2. *//Subir archivos -> El servidor recibe*
3. String rutaDirectorio = dis.readUTF();
4. String path = rutaServer + rutaDirectorio;
5. **File archivosRuta = new File(path);**
6. **if** (!archivosRuta.exists()) {
7. archivosRuta.mkdir();
8. }
9. }

*La bandera vale 5: El cliente va a eliminar archivos/directorios.*

1. **else** **if** (bandera == 5) {
2. **int** tam = dis.readInt();
3. EliminarArchivo(dis, tam, dos);
4. }

void RecibirArchivos(dis,nombre)

De forma muy similar a la clase Cliente, el servidor recibe los nombres y tamaños de los archivos enviados por el cliente por medio de un flujo de entrada del socket, los va leyendo byte a byte y finalmente los escribe en la carpeta ./serverP1 en el disco duro por medio de un flujo de salida

1. **public** **static** **void** RecibirArchivos(DataInputStream dis, String nombre) **throws** IOException {
2. **long** tam = dis.readLong();
3. String pathDestino = dis.readUTF();
4. nombre = rutaServer + pathDestino;
6. System.out.println("**\n**Se recibe el archivo " + nombre + " con " + tam + "bytes");
7. DataOutputStream dos = **new** DataOutputStream(**new** FileOutputStream(nombre)); *// OutputStream*
9. **long** recibidos = 0;
10. **int n = 0, porciento = 0;**
11. **byte**[] b = **new** **byte**[2000];
13. **while** (recibidos < tam) {
14. n = dis.read(b);
15. **dos.write(b, 0, n);**
16. dos.flush();
17. recibidos += n;
18. porciento = (**int**) ((recibidos \* 100) / tam);
19. System.out.println("**\r** Recibiendo el " + porciento + "% --- " + recibidos + "/" + tam + " bytes");
20. **} *// while***
22. System.out.println("**\n**Archivo " + nombre + " de tamanio: " + tam + " recibido.");
23. dos.close();
24. dis.close();
25. **}**

void ActualizarCliente(cl, dis, rutaServer, bandera)

No se envía ningún archivo o carpeta (en bytes), solo su nombre si se trata de un archivo o su ruta relativa si se trata de un directorio. Este método es meramente informativo para que el cliente sepa que hay en su carpeta actualmente.

1. **public** **static** **void** ActualizarCliente(Socket cl, DataInputStream dis, String path, **int** bandera) **throws** IOException {
2. File archivosRuta = **new** File(path);
4. **if** (!archivosRuta.exists()) {
5. **archivosRuta.mkdir();**
6. }*//if*
8. **if** (bandera == 1) {
9. rutaActual = rutaActual + sep + archivosRuta.getName();
10. **System.out.println("Ubicacion: " + rutaActual);**
11. }
13. list = archivosRuta.listFiles();
15. **DataOutputStream dos = new DataOutputStream(cl.getOutputStream()); *// OutputStream***
17. dos.writeInt(list.length);
18. dos.flush();
20. **String info = "";**
21. **int** tipo = 0;
23. **for** (File f : list) {
24. **if** (f.isDirectory()) {
25. **tipo = 1;**
26. **if** (bandera == 0) {*//Ruta raiz - Inicio*
27. info = "." + sep + f.getName();
28. } **else** {*//Abrir ruta y concatenar*
29. info = "." + rutaActual + sep + f.getName();
30. **}**
31. }*//if*
32. **else** {
33. tipo = 2;
34. **if** (bandera == 0) {*//Ruta raiz - Inicio*
35. **info = f.getName();**
36. *//info = f.getName() + " ------- " + f.length() + " bytes";*
37. } **else** {*//Abrir ruta y concatenar*
38. info = "." + rutaActual + sep + f.getName();
39. *//info = "." + rutaActual + sep + f.getName() + " ------- " + f.length() + " bytes";*
40. **}**
41. }*//else*
42. dos.writeUTF(info);
43. dos.flush();
44. dos.writeInt(tipo);
45. **dos.flush();**
47. tipo = 0;
48. }*//for*
49. dos.close();
50. **System.out.println("Informacion enviada al cliente: Carpeta actualizada.");**
51. }

void EnviarArchivos(dos, file)

El servidor lee el archivo ZIP (que contiene ya a los archivos y carpetas solicitadas por el cliente), lo carga byte a byte desde el disco con ayuda de un flujo de entrada, y lo envía al cliente con ayuda de un socket y un flujo de salida.

1. **public** **static** **void** EnviarArchivo(DataOutputStream dos, File f) {
2. **try** {
3. String nombre = f.getName();
4. **long** tam = f.length();
5. **String path = f.getAbsolutePath();**
6. System.out.println("**\n**Se envia el archivo " + nombre + " con " + tam + " bytes");
7. DataInputStream disArchivo = **new** DataInputStream(**new** FileInputStream(path)); *// InputStream*
9. *//Se envia info de los archivos*
10. **dos.writeUTF(nombre);**
11. dos.flush();
12. dos.writeLong(tam);
13. dos.flush();
15. **long enviados = 0;**
16. **int** n = 0, porciento = 0;
17. **byte**[] b = **new** **byte**[2000];
19. **while** (enviados < tam) {
20. **n = disArchivo.read(b);**
21. dos.write(b, 0, n);
22. dos.flush();
23. enviados += n;
24. porciento = (**int**) ((enviados \* 100) / tam);
25. **System.out.println("\r Enviando el " + porciento + "% --- " + enviados + "/" + tam + " bytes");**
26. } *//while*
28. System.out.println("**\n**Archivo " + nombre + " de tamanio: " + tam + " enviado.");
29. **disArchivo.close();**
30. dos.close();
31. } *// try*
32. **catch** (Exception e) {
33. e.printStackTrace();
34. **}**
35. }

void ElimiarArhchivo(dis, tam, dos).

1. **public** **static** **void** EliminarArchivo(DataInputStream dis, **int** tam, DataOutputStream dos) {
2. **try** {
3. String[] nombreArchivos = **new** String[tam];
4. String aux = "";
5. **int i, j;**
6. **for** (i = 0; i < tam; i++) {
7. nombreArchivos[i] = dis.readUTF();
8. String nombre = nombreArchivos[i];
9. **boolean** bandera = **false**;
10. **if (nombre.indexOf(".") == 0) {**
11. nombre = nombre.substring(2,nombre.length());
12. bandera = **true**;
13. }
14. System.out.println("**\n**Archivo: " + nombre);
15. **File f = new File(rutaServer + nombre);**
16. **if** (bandera) {
17. deleteDir(f);
18. System.out.println("Carpeta eliminada");
19. } **else** {
20. **if (f.delete()) {**
21. System.out.println("**\n**Archivo " + nombre + " de tamanio: " + tam + " Eliminado.");
22. } **else** {
23. System.out.println("**\n**Archivo " + nombre + " de tamanio: " + tam + " no eliminado.");
24. **}**
25. }
26. dos.writeUTF(nombre);
27. dos.flush();
28. dos.writeLong(tam);
29. **dos.flush();**
30. }
31. } *// try*
32. **catch** (Exception e) {
33. **e.printStackTrace();**
34. }
35. }

**Funcionamiento:**

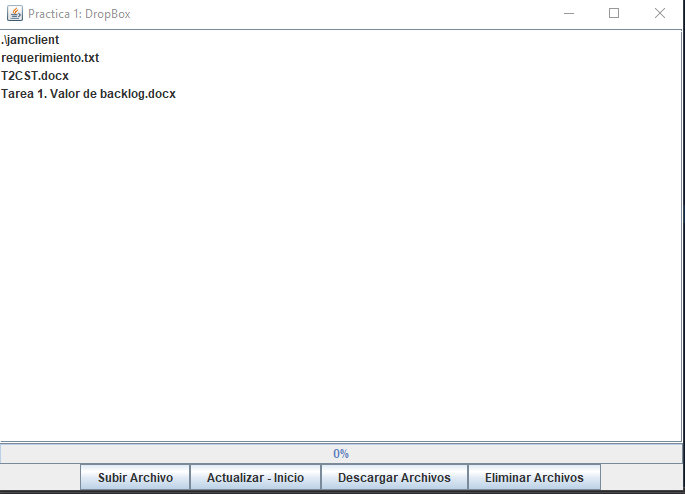


Figura 1: Cliente

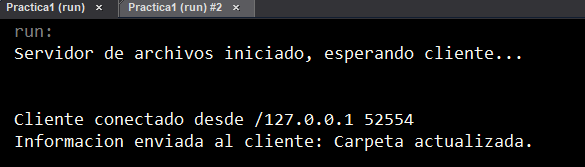


Figura 2: Servidor

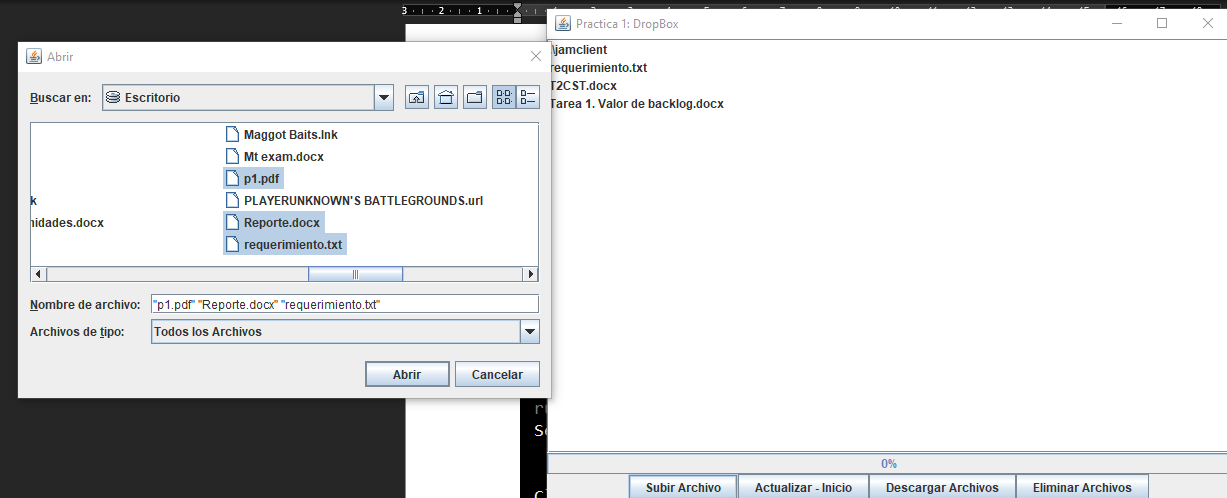


Figura 3: Seleccionando Archivos para subir

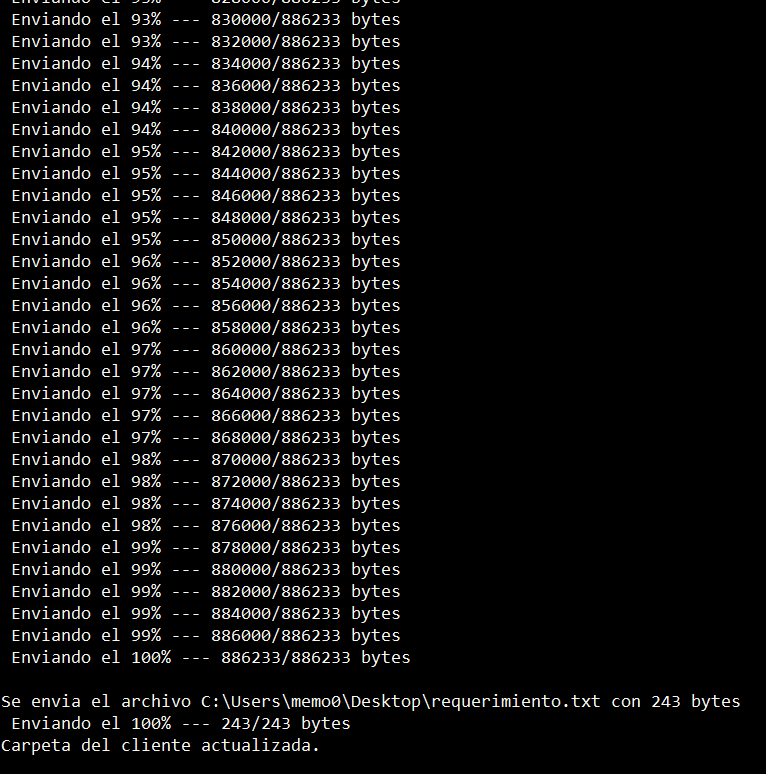
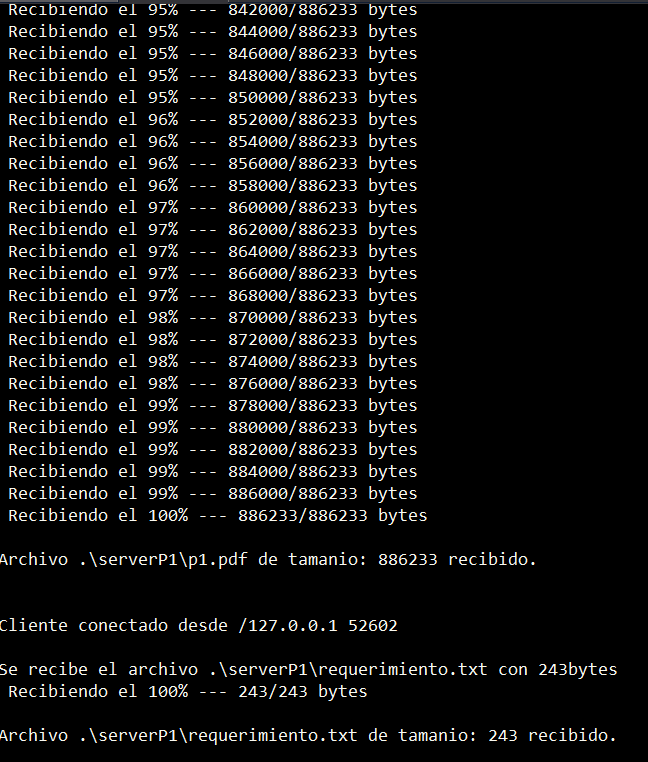


Figura 4: Respuesta del servidor

Figura 5: Respuesta del cliente

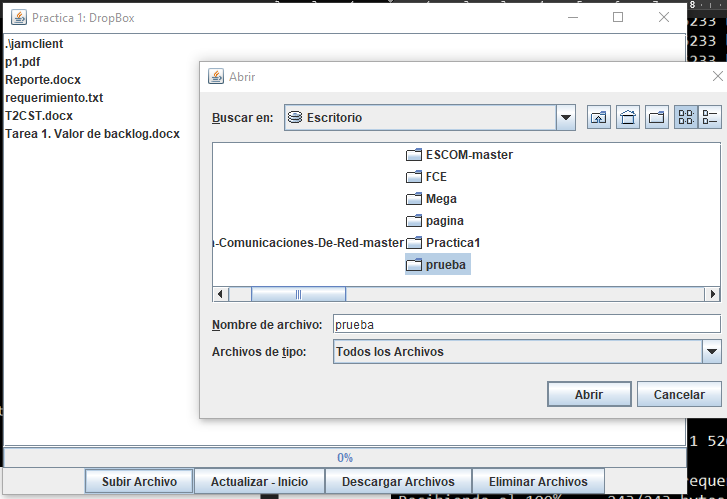


Figura 6: Subiendo carpeta

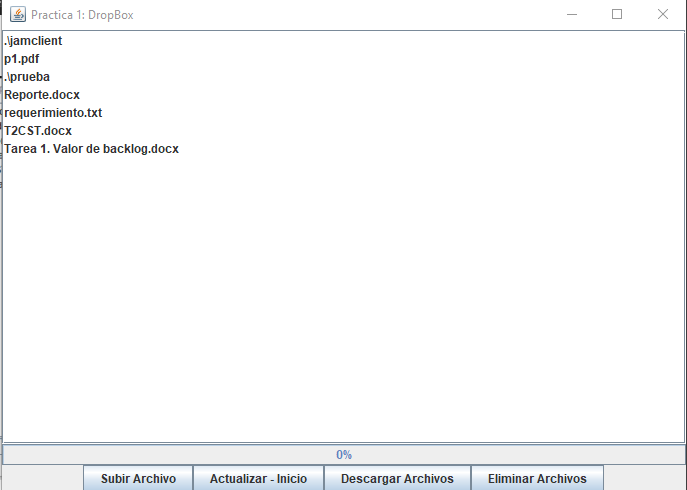


Figura 7: Servidor actualizado.

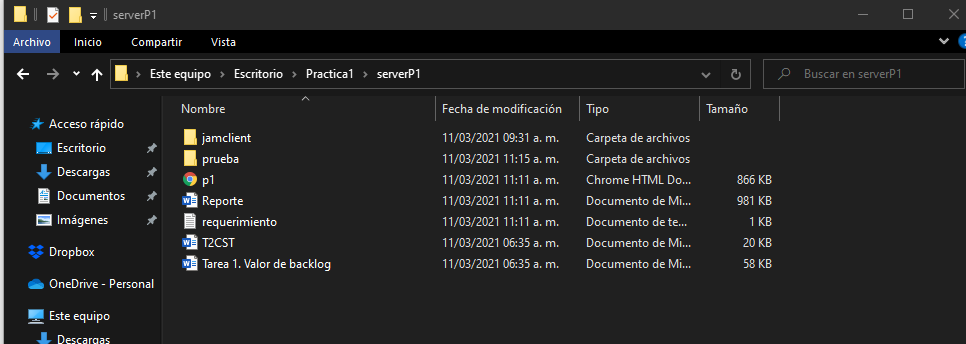


Figura 8: Carpeta del servidor

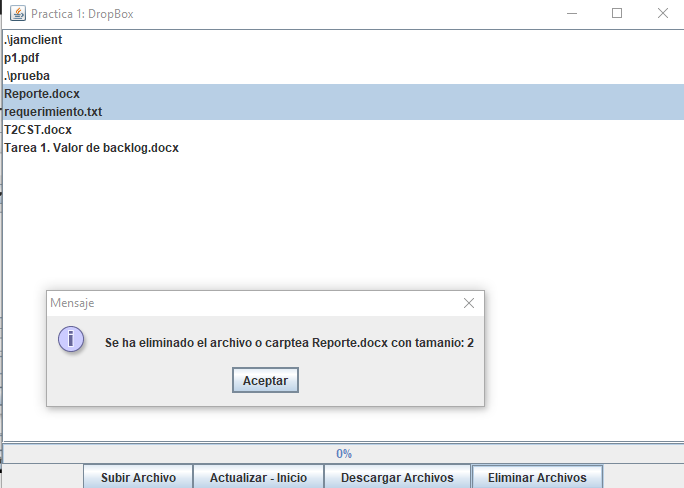


Figura 9: Eliminando archivos

**Conclusiones:**

Ramirez Olvera Guillermo:

Este programa es muy interesante debido a que es algo con lo que tenemos contacto todos los días, por ejemplo, para hacer algún programa o reporte de práctica entre compañeros porque nos compartimos los archivos por medio de Facebook o alguna otra aplicación.

Fue complicado en el manejo de directorios al usar directorios relativos, pero sin duda fue una práctica interesante dándonos a conocer diversas situaciones en donde nos es posible aplicar nuestro conocimiento.

Sánchez Méndez Edmundo Josué

El trabajo de comunicación y transmisión de archivos no fue problema, pues la clase Socket y ServerSockets (sockets de flujo bloqueante) en Java hacen todo este proceso con métodos muy simples de utilizar y comprender.

El verdadero reto fue el manejo de directorios y subdirectorios, tanto para subir como para descargar archivos, ya debemos tener un control del directorio de trabajo actual, y de distinguir entre archivos y carpetas cuando el usuario las seleccionaba en el cliente. Hablando del cliente, también hubo algo de complejidad en emular un explorador de archivos como el de Windows o Google Drive al abrir carpetas, ver sus contenidos.

Nos apoyamos de métodos como recursividad, condicionales, y otros métodos de clases como el de compresión de archivos a descargar en un ZIP, eligiendo esta forma para manejar los archivos y directorios que se descargan desde el servidor.