Instituto Politécnico Nacional

Unidad Académica Interdisciplinaria de Ingeniería Campus Zacatecas

Nombre del alumno : Gerardo Ayala Juárez
Fecha de entrega:13-Febrero-2016
Evidencia: Practica 2 - Construcción de un servicio de transferencia de archivos utilizando sockets orientados a conexión no bloqueante

Nombre de la Maestra: Sandra Mireya Monreal Mendoza
Programa Académico: Sistemas Computacionales
Unidad de Aprendizaje: Aplicaciones para Comunicaciones en Red

Introducción.-

Compartir archivos desde una computadora a otra (inclusive un teléfono inteligente) es un proceso que realizamos de manera cotidiana, ya sea mediante correo electrónico, red social, servicio de mensajería instantánea o chat, etc. Dicho proceso es importante cuando se desea compartir información con otras personas. El desarrollo de una aplicación de intercambio de archivos puede ser especialmente útil cuando es necesario que varios funcionarios de diferentes despachos compartan información de forma eficiente y con la seguridad que será dentro de la misma red de la empresa u organización.

Objetivo(s)

Crear una aplicación que permita la transferencia de archivos de texto mediante la implementación de los sockets de flujo en modo no bloqueante.

Desarrollo

Se debe crear una aplicación cliente-servidor que transmita archivos. Para esto, se debe permitir que:

- El cliente envíe un archivo (usar el JFileChooser) y el servidor lo recibe y escribe.
- Crear una carpeta en el servidor para los archivos recibidos.

Para el servidor y para el cliente, se deberán emplear los siguientes códigos fuente, crear adecuadamente cada clase y agregar comentarios a todas las líneas de código.

Capturas de pantalla.-

```java
public class cliente  {
    public cliente() throws IOException{
        JFileChooser j = new JFileChooser();//Un objeto que nos va permitir escojer un archivo
        j.showSaveDialog(null);//La forma de la seleccion del archivo
        SocketChannel cliente = SocketChannel.open();//La posibilidad de poder conectarse como cliente un servidor
        InetSocketAddress socketAddr = new InetSocketAddress("localhost",9000);//La direccion a la que quiere conectarse
        cliente.connect(socketAddr);//la conexion
        cliente.configureBlocking(false);//Declarandola como no bloqueante
        Path path = (j.getSelectedFile().toPath());//utiliza la direccion del arivho
        FileChannel channel = FileChannel.open(path);//Abre el archivo
        ByteBuffer buffer = ByteBuffer.allocate(1024);//Hace un medio de transporte
        while(channel.read(buffer)>0){//Le la linea del archivo
            buffer.flip();//Se prepara para escribir
            cliente.write(buffer);//Escribe en el buffer
            buffer.clear();//Lo que ya se envio, se limpia para poder leer de nuevo
        }
        channel.close();//finalizo el envio del archivo
        System.out.println("Archivo enviado");//Confirma en la consola
        cliente.close();//Cierra la conexion
    }
```

Clase cliente.

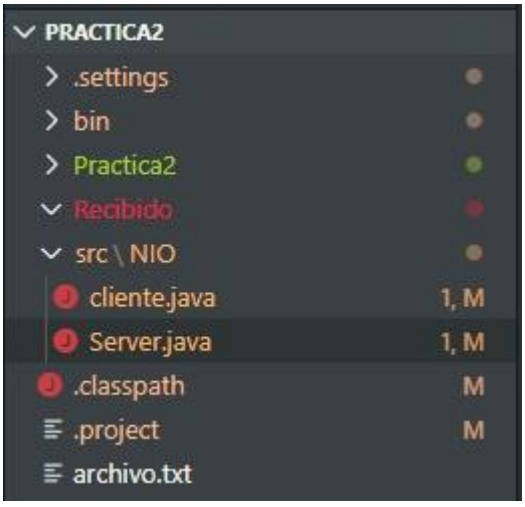
```java
////Inicializacion del servido Bloque
int i=0;//Numero de archivos que llevara el contero de los archivos recibidos;
ServerSocketChannel server = null;//Se declara el servido
SocketChannel client = null;//Se declara el cliente
server = ServerSocketChannel.open();//Abre la posiblidad de recibir solicitudes de comunicacion
server.socket().bind(new InetSocketAddress(puerto));//Se inicializa el servidor
server.configureBlocking(false);//Configuramos que sea no blockeante
System.out.println("Servidor a la escucha...");
    while (client == null) {//busca hasta que encuentre un cliente conectado
        client = server.accept();//Se encontro con un cliente
    }
    i++;//Para que genere un diferente nombre al archivo anterior mencionado
    System.out.println("Se conecto: "+client.getRemoteAddress());//Status de que se conecto el cliente.
    Path path = Paths.get("Recibido/archivo_"+i+".txt");//Declaramos la direccion del archivo que nos van a enviar
    FileChannel fileChannel = FileChannel.open(path, EnumSet.of(StandardOpenOption.CREATE,
    StandardOpenOption.TRUNCATE_EXISTING, StandardOpenOption.WRITE));//EStablecemos una secuencia o posibilidades, en caso de situaciones previstas.
    ByteBuffer buffer = ByteBuffer.allocate(1024);//La via por las que vamos a enviar
    while(client.read(buffer)>0){//LE una linea del archivo
            buffer.flip();//cambia el sentido del buffer
            fileChannel.write(buffer);//Escribe en el archivo lo que se recibio
            buffer.clear();//Se limpia el buffer para si en caso de ser leido de nuevo tener todo el espacio disponible
    }
        fileChannel.close();//cierra el archvio recibido
    System.out.println("Archivo recibido");//Se confirma la recepcion del archivo
    client.close();//Se cierra conexion con el cliente
```

Código en ejecución:

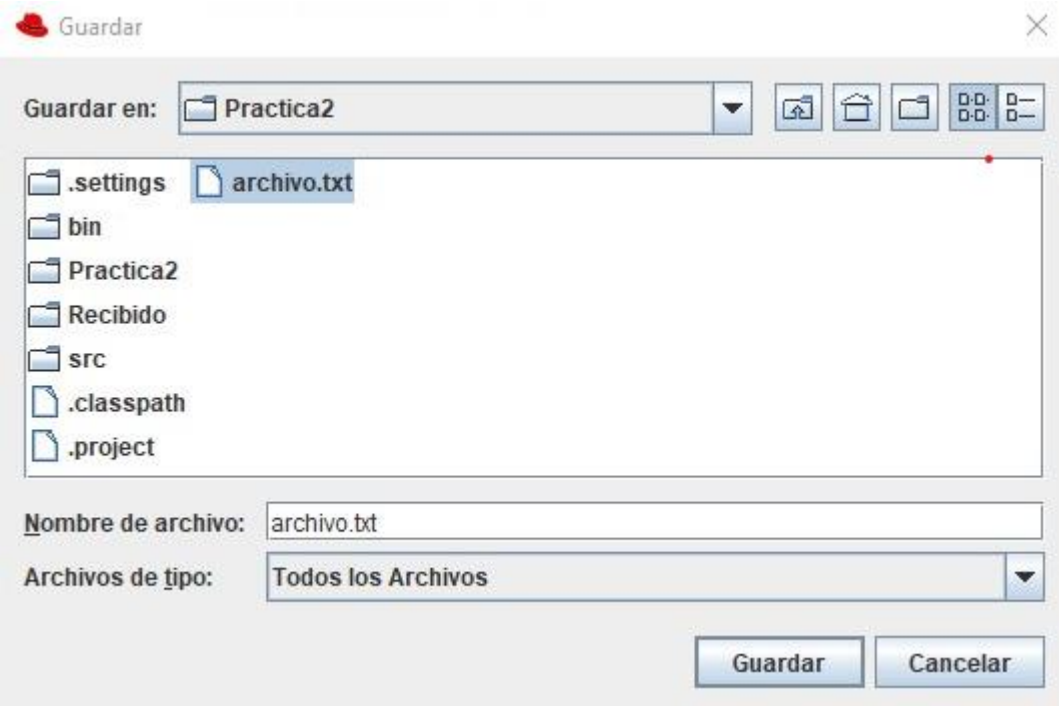
Servidor. –

```
PS D:\Documentos\Git\APCR\Practica2> & 'C:\Users\futbo\.vscode\extensions\vscjava.vscode-java-debug-0.24.0\scripts\launcher.bat' 'D:
\Programacion\java\bin\java' '-Dfile.encoding=UTF-8' '-cp' 'D:\Documentos\Git\APCR\Practica2\bin' 'NIO.Server'
Servidor a la escucha...
```

Carpeta del Proyecto:



Cliente. –



A la hora de Tener el archivo listo, hacemos conexión.

```
PS D:\Documentos\Git\APCR\Pract
\Programacion\java\bin\java' '-
Servidor a la escucha...
Se conecto: /127.0.0.1:54968
```

E inmediatamente se manda el archivo

```
PS D:\Documentos\Git\APCR\Practica2> & 'C:\Users\futbo\.vscode\extensions\vscjava.vscode-java-debug-0.24.0\scripts\launcher.bat' 'D:
\Programacion\java\bin\java' '-Dfile.encoding=UTF-8' '-cp' 'D:\Documentos\Git\APCR\Practica2\bin' 'NIO.cliente'
Archivo enviado
PS D:\Documentos\Git\APCR\Practica2>
```

Y el servidor Lo recibe y lo manda a la carpeta de recibidos.

```
Servidor a la escucha...
Se conecto: /127.0.0.1:54968
Archivo recibido
```

Dificultades.-

- Al estar de manera no bloqueante, puede generar errores debido a que no espera a los procesos, por lo que el primer error encontrado fue a que no se podía conseguir conexión como antes, o dicho de otra manera, tenia que estar iterando hasta encontrar conexión, proceso innecesario de manera bloqueante, debido a que no sigue el código hasta encontrar una conexión valida.

Posibles mejoras.-

Tener más parámetros para poder clasificar los archivos generados y recibidos.

Conclusiones.-

Después de ver el funcionamiento de los sockets de manera no bloqueante, podemos concluir que ambos sistemas a pesar de ser una propiedad muy especifica, ambas tienen situaciones ideales aunque hay que estar atento de ambas posibilidades, la recomendación es usar banderas para poder continuar con procesos y finalizar para evitar errores.