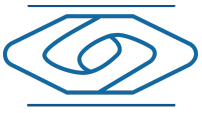
INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



Unidad de Aprendizaje:
Aplicaciones para Comunicaciones de Red

Profesora:
Sandra Ivette Bautista Rosales

Alumnos:
Ramírez Álvarez Ángel Negib
Valentín Mondragón Alfredo Emmanuel

Grupo:
3CV6

Equipo:
1

Práctica Número:
01

Fecha de Entrega:
01 de marzo de 2019

Objetivo

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

Introducción

Un **socket** (enchufe), es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un socket se define como el punto final en una conexión. Los sockets se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets (API, application programming interface).

Un **socket** es también una dirección de Internet, combinando una dirección IP (la dirección numérica única de cuatro partes que identifica a un ordenador particular en Internet) y un número de puerto (el número que identifica una aplicación de Internet particular, como FTP, Gopher, o WWW).

Principio de funcionamiento.

Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto. Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

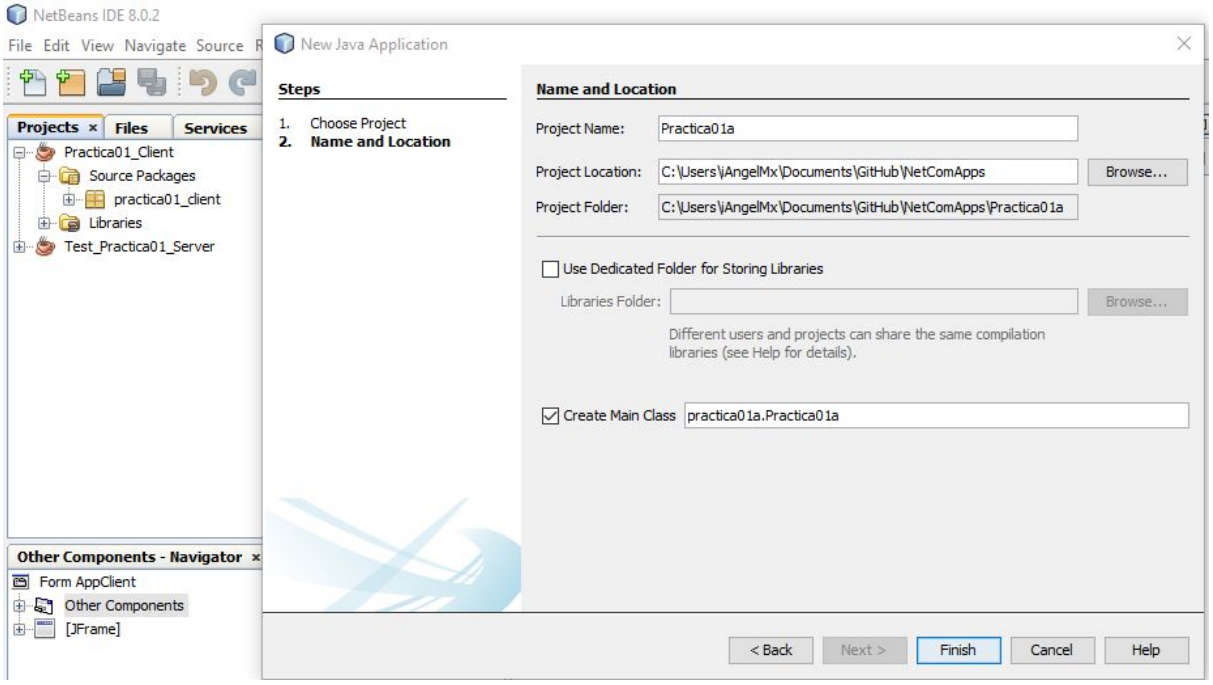
Que un programa sea capaz de localizar al otro.
Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.
Para ello son necesarios los tres recursos que originan el concepto de socket:

Un protocolo de comunicaciones, que permite el intercambio de octetos.
Un par de direcciones del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP), que identifica la computadora de origen y la remota.
Un par de números de puerto, que identifica a un programa dentro de cada computadora.
Los sockets permiten implementar una arquitectura cliente-servidor o peer to peer. La comunicación debe ser iniciada por uno de los programas que se denomina programa cliente. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor. Cabe mencionar que hay distintos tipos de sockets utilizados con TCP y UDP, como sockets dg_ram, sockets_stream, etc.

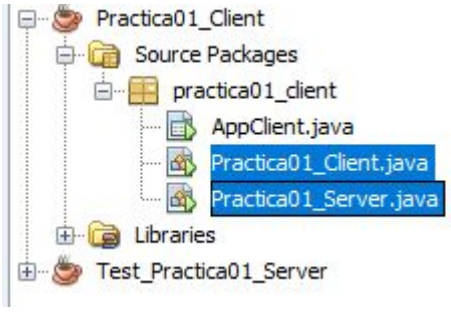
Desarrollo

En el análisis del problema para resolver la práctica decidimos investigar en la documentación de java acerca de los sockets para tener un panorama más amplio al programarlo.

Primero creamos el proyecto en java como se muestra en la imagen siguiente.

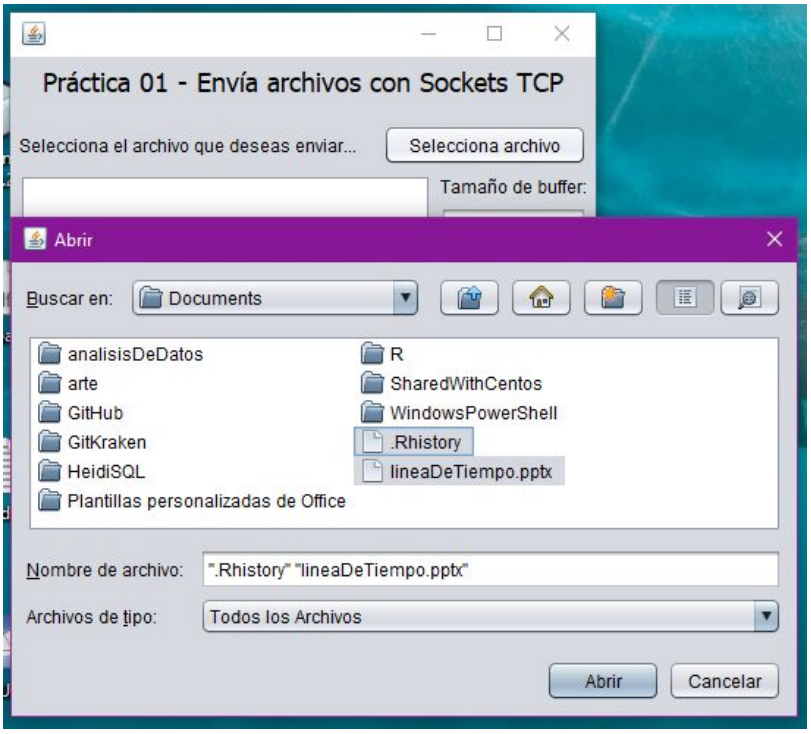


Posteriormente creamos dos clases de java los cuales funcionarán como cliente y servidor.



Cliente

Para el caso del cliente utilizamos un JFileChooser para hacer la selección múltiple de los archivos, los cuales al cargarlos nos muestra en pantalla, nombre del archivo, extensión y peso del mismo. se puede apreciar en la siguiente imagen.



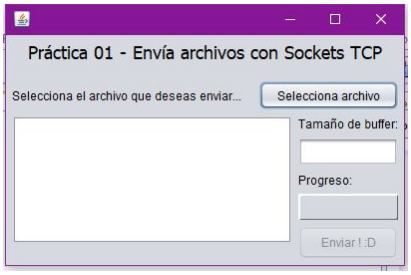
Posteriormente con las bibliotecas de **Net** y **File** hicimos el manejo de los sockets y su configuración previa la cual consta de puerto y dirección ip del servidor como se aprecia en la siguiente imagen.

```
public void enviaInfo(String path) throws IOException{
    Socket socket = null;
    String host = "10.100.73.11";

    socket = new Socket(host, 5555);

    File file = new File(path);
    // Get the size of the file
    long length = file.length();
```

Para el envío de los archivos hicimos uso de un arreglo de bytes y aquí se define el tamaño del buffer para cada archivo, los cuales servirán para escribir en el socket una vez que el servidor acepte la conexión a este, lo podemos ver en la siguiente imágen.




```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {

    int tamBuf = Integer.parseInt(buffer.getText());
    String str = "", detalles="";

    for( File archivo : files ){
        String path = archivo.getPath();
        try{
            enviaInfo(path, tamBuf);
        }
        catch(Exception ex){
        System.out.println(ex);}
    }
    // TODO add your handling code here:
}
```

```
public void enviaInfo(String path, int tamBuffer) throws IOException{
    int var = 0;
    if (tamBuffer == 0){
        var = 16;
    }else{
        var = tamBuffer;
    }
    Socket socket = null;
    String host = "10.100.73.11";
    socket = new Socket(host, 5555);

    File file = new File(path);
    // Get the size of the file
    long length = file.length();
    byte[] bytes = new byte[var * 1024];
    InputStream in = new FileInputStream(file);
    OutputStream out = socket.getOutputStream();
```

En este caso primero creamos un archivo el cual contendrá la información de los archivos que se enviarán al servidor y después se envían estos ya que el servidor tiene esa información.

```
for( File archivo : files ){

    String name = archivo.getName();
    String ext = name.substring(name.lastIndexOf("."));
    String tam = String.valueOf(archivo.length());

    str+= "Archivo: "+name+"\n";
    str+= "Extens.: "+ext+"\n";
    str+= "Tamanio: "+tam+" bytes\n";
    str+= "----------------\n";
    detalles += name+"|"+ext+"|"+tam+"\n";
}
btnEnviar.setEnabled(true);
String ruta = "C:\\Users\\iAngelMx\\Desktop\\archivo.txt";
File archivo = new File(ruta);
try{
    FileWriter fw = new FileWriter(archivo);

    PrintWriter escribir;
    if(archivo.exists()) {
        escribir = new PrintWriter(fw);
        escribir.write(detalles);
    } else {
        escribir = new PrintWriter(fw);
        escribir.write(detalles);
    }
    escribir.close();
    enviaInfo("C:\\Users\\iAngelMx\\Desktop\\archivo.txt", 0);
```

Servidor

Para el caso del servidor solamente hicimos uso de las bibliotecas net y file, las cuales nos sirvieron para recibir la solicitud de conexión por parte del cliente.

Primero creamos el socket de escucha por el cual el servidor recibirá las solicitudes, configurando el puerto de escucha de este, como se muestra en la imagen siguiente

```
public static void main(String[] args) throws IOException {
    int num_env=0;
    ServerSocket serverSocket = null;
    try {
        serverSocket = new ServerSocket(5555);
        System.out.println("server is running");
    } catch (IOException ex) {
        System.out.println("Can't setup server on this port number. ");
    }
    while(true){
    Socket socket = null;
    InputStream in = null;
    OutputStream out = null;
```

Posteriormente creamos el socket de servicio el cual atenderá las peticiones, de los archivos que está recibiendo, ya que obtenemos el flujo de datos escribimos estos en un archivo de salida con el nombre de los archivos

```
if(num_env==0){
    try {
    out = new FileOutputStream("C:\\Users\\neyer\\Documents\\ESCOM\\8vo Semestre\\Aplicacior
    } catch (FileNotFoundException ex) {
    System.out.println("File not found. ");
    }
     byte[] bytes = new byte[32*1024];

    int count;
    int i=0;
    while ((count = in.read(bytes)) > 0) {
        out.write(bytes, 0, count);
        i++;
        System.out.println("linea "+i+". Cuenta servidor: "+count);
    }

    out.close();
    in.close();
    num_env++;
}else{
```

Después programamos una pequeña parte la cual indica que siempre el primer archivo son los datos de los archivos que se van a recibir.

Cuestionario

1. ¿En qué consiste el algoritmo de Nagle y de qué manera se podría aplicar en la práctica? ¿Cuáles son sus ventajas y desventajas?
El algoritmo de Nagle es usado para automáticamente unir un pequeño número de mensajes, y agregarlos dentro de paquetes más grandes, para reducir la congestión de la red y utilizar los recursos más eficientemente, disminuyendo el número de paquetes que tienen que ser usados.

- Una primera mejora que sirve como solución a esta baja eficiencia de transmisión.
- El algoritmo acumula información para enviar entre el último asentimiento recibido y el asentimiento por venir.
- Disminuye el tráfico en la red

2. ¿Qué tipo de archivos se enviaron más rápido?
Los más pequeños.

3. ¿Cuál fue el número máximo de archivos que fue posible enviar a la vez?
No se encontró un límite por parte del servidor, el límite lo imponía el usuario para pocos archivos.

4. ¿Cuál fue el tamaño de archivo más grande que se pudo transferir? **No hubo archivo que no se pudiera transferir** ¿por qué? Puesto que el sistema de archivos de donde se estaban recibiendo permitía la correcta escritura de archivos grandes

5. ¿Qué es el orden de red?
Es el sentido que se le da a los segmentos enviados a través de una red. Existen.

La red de capa de trayecto de orden inferior es la encargada de transportar flujos de usuario de 2Mbit/s, en contenedores virtuales VC-12, y de 34Mbit/s (y para alguna aplicación específica también de 45Mbit/s).
La red de capa de trayecto de orden superior proporciona servicios de transporte tanto a la capa de trayecto de orden inferior como a la capa de circuitos (transportando flujos de usuario de 140Mbit/s).

6. ¿Por qué razón es importante utilizar el orden de red al enviar los datos a través de un socket?
Para garantizar el correcto envío de datos a través de la red sin perder algún segmento y sin tener que saturar todos los anchos de banda que ocupe nuestro socket.

7. 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?

Se debería fraccionar a un nivel inferior para poder transmitir bloques pequeños y a intervalos, dependiendo la latencia y ocupabilidad de la red.
Por ende, el servidor debería estar preparado para recibir la cantidad de fragmentos a recibir para que pueda reensamblarlos sin mayor detalle.

Conclusiones

Ramirez Alvarez Angel Negib:
En esta práctica se pudo observar de forma clara la interacción que pueden tener distintos procesos para poder comunicarse mediante la arquitectura TCP. Los sockets, como vimos, es la base para cualquier aplicación que debe comunicarse con otra… Sin embargo, puede llegar a variar la forma en la que se trata la información que se comparte, como se observó con los archivos de gran tamaño, en comparación de los que eran más pequeños.

Valentin Mondragon Alfredo Emmanuel:

En esta práctica aprendimos el uso de los sockets así como el envío de la información la diferencia que se tiene entre un cliente y un servidor y cual es el papel que debe desempeñar cada uno.

Referencias

[1] Barnett, A. (2009). *What is serialization?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/633402/what-is-serialization [Accessed 15 Mar. 2019].

[2] Es.wikipedia.org. (2018). *Socket de Internet*. [online] Available at: https://es.wikipedia.org/wiki/Socket_de_Internet [Accessed 20 Mar. 2019].

[3] Ecured.cu. (2018). *Socket - EcuRed*. [online] Available at: https://www.ecured.cu/Socket [Accessed 20 Mar. 2019].

[4] Ciscocertificacion.ccna (2016) *Algoritmo de nagle - Blogspot* [online] Available at: http://ciscocertificacionccna.blogspot.com/p/algoritmo-de-nagle.html [Accessed 20 Mar. 2019].

[5] Es.wikipedia.org (2017) *Tipos de redes - Wikipedia* [online] Available at: https://es.wikipedia.org/wiki/Tipos_de_redes [Accessed 20 Mar. 2019].