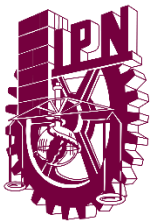


Instituto Politécnico Nacional
Escuela Superior de Cómputo
Aplicaciones para comunicaciones de red
Práctica No 1
Prieto de la Cruz Felipe
Huerta Larrazolo Jehiel David
18/04/18

Profesor: Bautista Rosales Sandra Ivette
Grupo: 3Cv5

Objetivo de la práctica:

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

Introducción

Puertos

Cada proceso que se comunica con otro proceso se identifica a sí mismo a la familia de protocolos TCP/IP por uno o más puertos. Un puerto es un número de 16 bits, usado por el protocolo host-a-host para identificar a qué protocolo de más alto nivel o programa de aplicación (proceso) debe entregar los mensajes de entrada.

Como algunos programas de más alto nivel son protocolos por sí mismos, estandarizados en la familia de protocolos TCP/IP, tales como telnet y ftp, usan el mismo número de puerto en todas las realizaciones de TCP/IP. Aquellos números de puerto "asignados" se denominan puertos bien-conocidos y las aplicaciones estándares servicios bien-conocidos.

Los puertos "bien-conocidos" los controla y asigna la Autoridad de Números Asignados de Internet (IANA) y en la mayoría de los sistemas sólo pueden usarlo los procesos del sistema o programas ejecutados con privilegios de usuario. Los puertos "bien-conocidos" asignados ocupan números de puerto en el rango de 0 a 1023. Los puertos con números dentro del rango 1024-65535 no los controla la IANA y la mayor parte de los sistemas únicamente usan programas desarrollados por usuarios.

Sockets

Consideremos la terminología siguiente:

Un socket es un tipo especial de manejador de fichero que utiliza un proceso para pedir servicios de red al sistema operativo.

Una dirección de socket es la tripleta: {protocolo, dirección-local, proceso-local}

En la familia TCP/IP, por ejemplo: {tcp, 193.44.234.3, 12345}

Una conversación es el enlace de comunicación entre dos procesos.

//Desarrollo

Cómo primer paso tenemos que declarar nuestras variables que vamos a usar, en este caso serán sockets, string donde guardaremos los valores del puerto, y la ip, nuestro buffer, que determinará el tamaño del archivo, estas variables serán similares para nuestras dos interfaces que serán, cliente servidor.

```
private String host = "192.168.0.2";
private int puerto = 9000;
private Socket sock;
private DataOutputStream salida;
private BufferedInputStream bis;
private BufferedReader entrada;
private BufferedOutputStream bos;
private String mensajeRecibido;
private File[] files;
private int countArchivos;
```

Tendremos nuestro siguiente método donde inicializaremos la conexión con el socket, obteniendo la información del host y puerto, los mensajes de conexión para las interfaces cliente y servidor.

```
        setHost("localhost");
        setPuerto(Integer.parseInt(jPuerto.getText()) );
    try{
        sock = new Socket(host, puerto);
        salida = new DataOutputStream(sock.getOutputStream());
        entrada = new BufferedReader(new InputStreamReader(sock.getInputStream())); //1
        //mensajeRecibido =
        TextAreaCliente.append(entrada.readLine() + "\n");
        salida.writeUTF("*****LA CONEXIÓN CON EL SERVIDOR SE HA ESTABLECIDO***** \n");
        TextAreaCliente.append("-Ahora puede enviar archivos. \n");
        sock.close();
    }
    catch(Exception e ){
        JOptionPane.showMessageDialog(null, "Error: " + e.getMessage());
        System.out.println(e.getMessage());
    }
}
```

Continuamos con el siguiente método donde incluiremos la librería JFileChooser que nos permitirá seleccionar diferentes archivos.

```
public void escogerArchivos() throws IOException{
    JFileChooser fc = new JFileChooser();
    fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
    if (!fc.isMultiSelectionEnabled()){
        fc.setMultiSelectionEnabled(true);
    }
    int returnVal = fc.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
         fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
         if (!fc.isMultiSelectionEnabled()){
             fc.setMultiSelectionEnabled(true);
         }
         files = fc.getSelectedFiles();
         countArchivos = files.length;
         archivosSelecionados.setText(Integer.toString(countArchivos));
     }
    TextAreaCliente.append("_____________________________________________________\n");
    TextAreaCliente.append("Archivos seleccionados: " + files.length + "\n");
    for(int i=0; i<files.length;i++){
        TextAreaCliente.append("Archivo: " + files[i].getName() + " Tamaño: " + files[i].length() +
    }
    TextAreaCliente.append("_____________________________________________________\n");
}
```

Para el método donde enviamos los nombres de nuestros archivos lo que vamos a hacer es con nuestro arreglo que se llama "Files" es donde guardamos la información de nuestros archivos, obtenemos su información, tanto como su tamaño.

```
public int sendNames(File file, int bufferSize, DataOutputStream out, InputStream in){
    int count=0;
    int bytesSent = 0;
    long fileSize = 0;
    byte[] byteBuffer = new byte[bufferSize];
    Socket socket;
    float porcentaje=0;float progreso=0;
    try{
        socket  = new Socket(host, puerto);
        out     = new DataOutputStream(socket.getOutputStream());
        in      = new FileInputStream(file);

        porcentaje=100/countArchivos;

        fileSize = file.length();
        TextAreaCliente.append("\tNombre: " +file.getName() +"\n");
        TextAreaCliente.append("\tTamaño: "+fileSize       +"\n");
        out.writeUTF(file.getName());
        out.writeLong(fileSize);
        count+=fileSize;
        socket.close();
        out.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return count;
}
```

Una vez que tenemos la información lo que haremos será establecer la conexión con nuestro socket, en este método uno de nuestros parámetros será el tamaño de nuestro buffer.

Ya recopila la información y haber establecido la conexión con nuestro socket, procederemos a mandar el nombre de los archivos, aquí se mandará nuestro arreglo, en el cual ya están alojados los archivos, únicamente extrajimos sus datos.

```
private void enviarNombresActionPerformed(java.awt.event.ActionEvent evt) {
    enviarNombres.setEnabled(false);
    enviarArchivos.setEnabled(true);
    try {
        InputStream in = null;
        DataOutputStream out = null;
        sendNumFiles(files);
        TextAreaCliente.append("\nEnviando Nombres\n");
        TextAreaCliente.append("_____\n
        for (int i = 0; i < files.length; i++){
            TextAreaCliente.append("Archivo "+(i+1)+": \n");
            files[i] = new File(files[i].getAbsolutePath());
            sendNames(files[i], 4096, out, in);
        }
        TextAreaCliente.append("_____\n
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
```

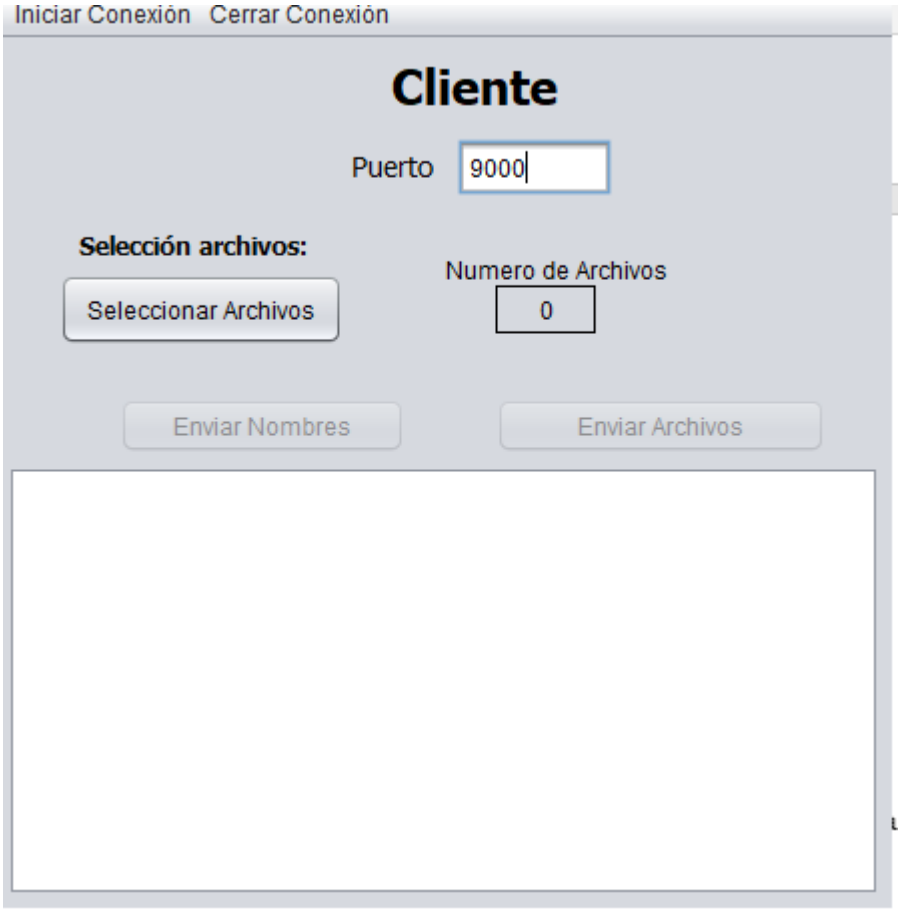
Ahora, para poder enviar nuestros archivos, haremos algo similar como lo que hicimos para poder obtener los nombres, estableceremos el tamaño del buffer, el nombre del archivo y su tamaño, por lo tanto también debemos establecer una conexión con el socket.

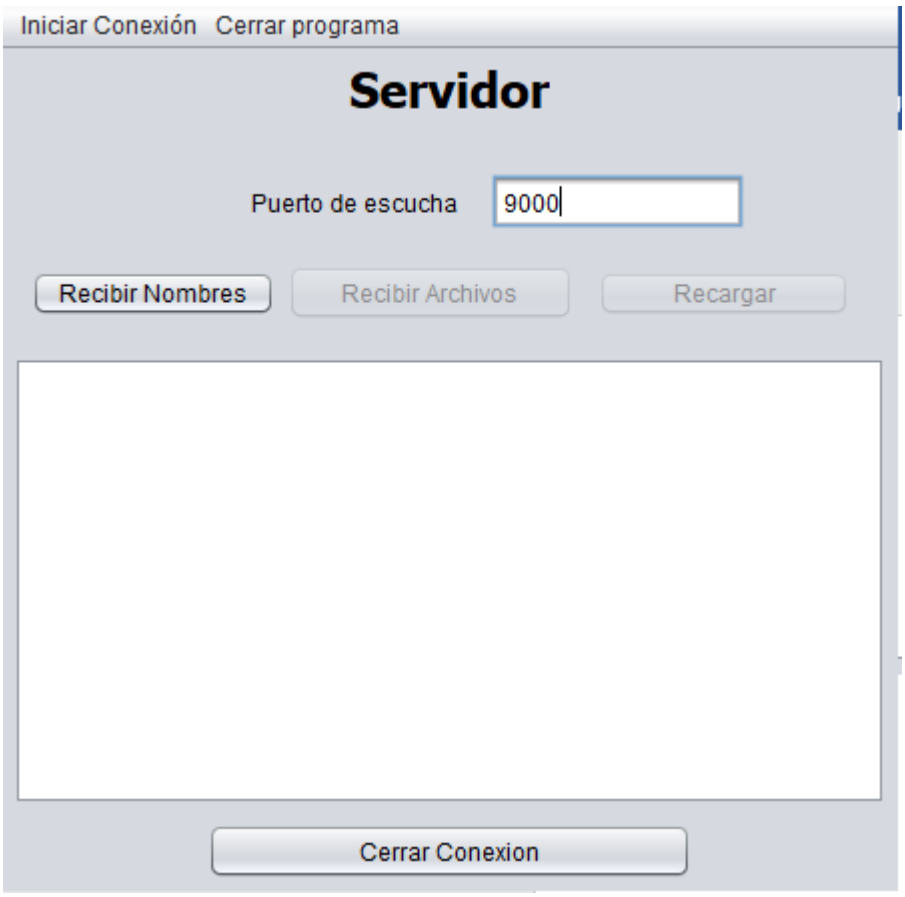
```java
public int sendfiles(File file, int bufferSize, DataOutputStream out, InputStream in){
    int count=0;
    int bytesSent = 0;
    long fileSize = 0;
    byte[] byteBuffer = new byte[bufferSize];
    Socket socket;
    float porcentaje=0;float progreso=0;
    try{
        socket  = new Socket(host, puerto);
        out     = new DataOutputStream(socket.getOutputStream());
        in      = new FileInputStream(file);

        porcentaje=100/countArchivos;

        fileSize = file.length();
        TextAreaCliente.append("\tNombre: " +file.getName() +"\n");
        TextAreaCliente.append("\tTamaño: "+fileSize       +"\n");
        TextAreaCliente.append("Enviando archivo...\n");
        out.writeUTF(file.getName());
        out.writeLong(fileSize);
        while ((bytesSent = in.read(byteBuffer)) > 0){
            out.write(byteBuffer, 0, bytesSent);
            progreso+=porcentaje;
        }
        TextAreaCliente.append("Se enviaron " + fileSize + " bytes.\n");
        count+=fileSize;
        socket.close();
        out.close();
    }
    catch (Exception e) {
```

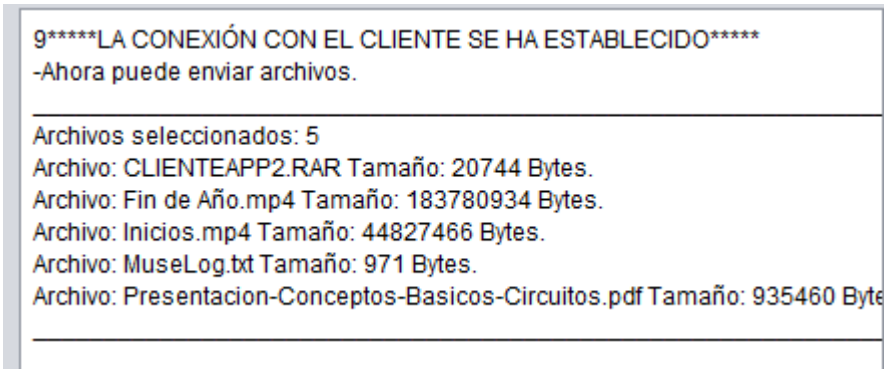
Igualmente, que con nombres el procedimiento para enviar los archivos es similar

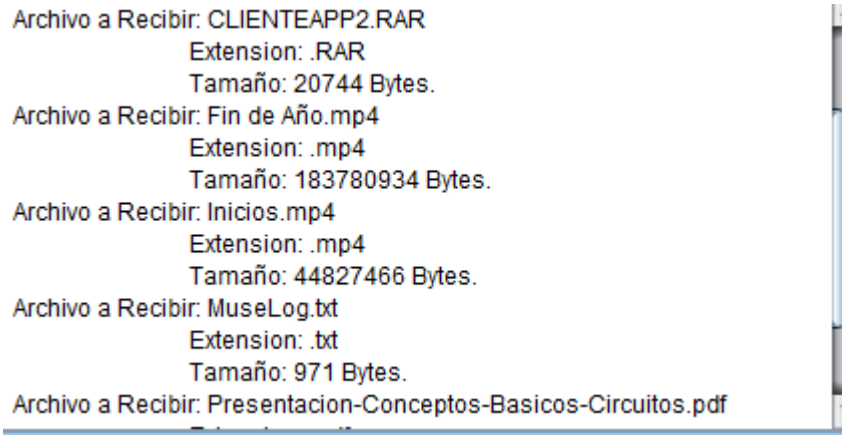
```java
private void enviarArchivosActionPerformed(java.awt.event.ActionEvent evt) {

    enviarNombres.setEnabled(true);
    enviarArchivos.setEnabled(false);
    try {
        InputStream in = null;
        DataOutputStream out = null;
        sendNumFiles(files);
        TextAreaCliente.append("\nEnviando Archivos\n");
        TextAreaCliente.append("_____________________________________________________\n"
        int prom=100/files.length;
        int percent=0;
        for (int i = 0; i < files.length; i++){
            TextAreaCliente.append("Archivo"+(i+1)+":\n");
            files[i] = new File(files[i].getAbsolutePath());
            sendfiles(files[i], 4096, out, in);
            TextAreaCliente.append("ARCHIVO ENVIADO\n");
            TextAreaCliente.append("_____________________________________________________
            percent+=prom;
        }
    }
    catch (Exception e) {
    }
}
```

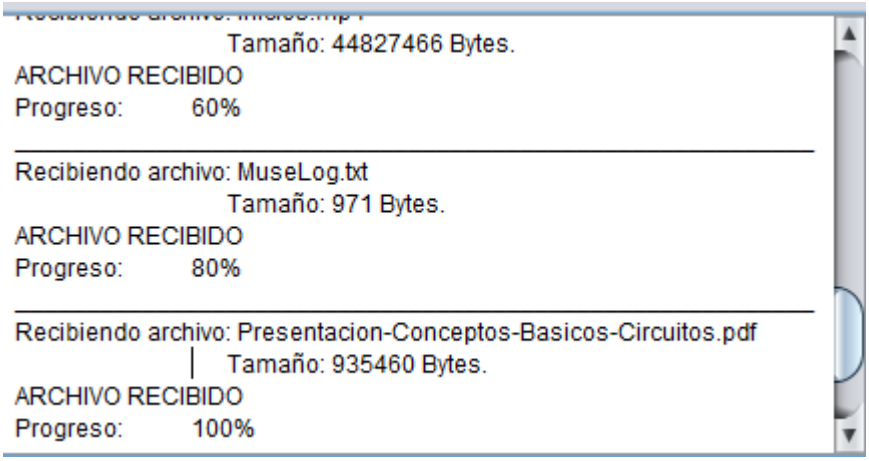
Así es como quedan nuestras interfaces:




Iniciar Conexión
Cerrar programa
Servidor
Puerto de escucha
9000
Recibir Nombres
Recibir Archivos
Recargar
Cerrar Conexion


9*****LA CONEXIÓN CON EL CLIENTE SE HA ESTABLECIDO*****
-Ahora puede enviar archivos.
Archivos seleccionados: 5
Archivo: CLIENTEAPP2.RAR Tamaño: 20744 Bytes.
Archivo: Fin de Año.mp4 Tamaño: 183780934 Bytes.
Archivo: Inicios.mp4 Tamaño: 44827466 Bytes.
Archivo: MuseLog.txt Tamaño: 971 Bytes.
Archivo: Presentacion-Conceptos-Basicos-Circuitos.pdf Tamaño: 935460 Byte


Archivo a Recibir: CLIENTEAPP2.RAR
Extension: .RAR
Tamaño: 20744 Bytes.
Archivo a Recibir: Fin de Año.mp4
Extension: .mp4
Tamaño: 183780934 Bytes.
Archivo a Recibir: Inicios.mp4
Extension: .mp4
Tamaño: 44827466 Bytes.
Archivo a Recibir: MuseLog.txt
Extension: .txt
Tamaño: 971 Bytes.
Archivo a Recibir: Presentacion-Conceptos-Basicos-Circuitos.pdf



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 se trata de un procedimiento que supone una mejora y aumento de eficiencia de las redes de comunicación basadas en Transmission Control Protocol (TCP). El algoritmo de Nagle es un método heurístico para evitar enviar paquetes IP particularmente pequeños, también denominados pequegramas (del inglés tinygrams). El algoritmo de Nagle intenta evitar la congestión que estos paquetes pueden ocasionar en la red reteniendo por poco tiempo la transmisión de datos TCP en algunas circunstancias.

Con esté algoritmo podemos mejorar el tiempo en que se envían los archivos haciéndolo que sea un poco más rápido.

Ventajas:
Utilizando el algoritmo de Nagle, una conexión TCP sólo puede tener un segmento de tamaño pequeño sin que se haya reconocido, solo podemos tener un único segmento.

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

   Los archivos con menor peso.

3. ¿Cuál fue el número máximo de archivos que fue posible enviar a la vez?

   Para esta prueba intentamos con 13 archivos que tenían un peso menor.

4. ¿Cuál fue el tamaño de archivo más grande que se pudo transferir? ¿por qué?

   683 Mb aunque nuestro buffer lo configuramos para que pudiera soportar 1GB.

5. ¿Qué es el orden de red?

6. 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?

Un cambio sería en el tamaño del buffer, la manera en la que estamos descomponiendo los archivos, ya que lo que hacemos es enviarlos por bytes, trozos.

7. ¿Por qué razón es importante utilizar el orden de red al enviar los datos a través de un socket?

   Por la forma en que es la transferencia de los datos.

8. ¿Qué es el orden de red?

   La prioridad en que van llegando los datos.

Conclusiones:

David:

En esta práctica reafirmamos nuestros conocimientos sobre sockets, los cuales nos permitieron establecer conexión entre 2 aplicaciones, la

definición de socket quedo mucho más clara para mí, ya que fue el medio por el cual pudimos abrir una ventana hacia otro ordenador comunicándonos a través de un puerto, observando como viajaba la información de un ordenador a otro. Esto sin duda es de gran importancia ya que sin el avance en estas tecnologías, la redes sociales o la forma en que nos comunicamos hoy en día no serían lo mismo.

Prieto de la Cruz Felipe:

En esta práctica volví a ver lo eficaz que puede ser Java al igual que sus bibliotecas, para la parte de los sockets, llegue a tener problemas en averiguar cómo se conectaban, ya que tienes que configurar bien los puertos, cuando ejecutas un socket, para que no se trabe tu aplicación es necesario que trabes con hilos, también fue interesante el pensar cómo podrías enviar los archivos y una forma de lograrlo fue en trozos, byte por bye, entre menor peso más rápido se enviará tu archivo.

References

[4] [En línea]. Available: http://www.serinfor.net/infraestructura-informatica/.

[5] EcuRED. [En línea]. Available: https://www.ecured.cu/Socket.

[6] SOPA. [En línea]. Available: http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/sockets.pdf.