INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Transferencia de Archivos

PRÁCTICA NO. 1

Nava Lara Luis Roberto / Mendoza Flores Erick Gabriel | 3cv5 | 2018-04-16

Objetivo

El estudiante implementara una aplicación para el envió de múltiples archivos de la red haciendo uso de los sockets de flujo.

Desarrollar un sistema en Java que cumpla con las siguientes funciones:

- Seleccionar diferentes archivos en el lado del cliente por medio de una interfaz gráfica diseñada en Swing.
- Crear un servidor que tenga la función de recibir los archivos enviados por el cliente.

Se hará uso de las primitivas de sockets para establecer la conexión y demostrar las propiedades de los sockets orientados a conexión.

INTRODUCCIÓN

¿Qué es un socket?

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.

Existen dos tipos de sockets:

1.- Sockets de flujo

Los sockets de flujo definen flujos de comunicación en dos direcciones, fiables y con conexión. Si envías dos ítems a través del socket en el orden "1, 2" llegarán al otro extremo en el orden "1, 2", y llegarán sin errores.

¿Cómo consiguen los sockets de flujo este alto nivel de calidad en la transmisión de datos?

Usan un protocolo llamado "Protocolo de Control de Transmisión", más conocido como "TCP".TCP asegura que tu información llega secuencialmente y sin errores. Puede que hayas oído antes "TCP" como parte del acrónimo " TCP/IP ", donde "IP" significa "Protocolo de Internet" IP se encarga básicamente del encaminamiento a través de Internet y en general no es responsable de la integridad de los datos.

2.- Sockets de datagramas

Los sockets de datagramas también usan IP para el encaminamiento, pero no usan TCP; usan el "Protocolo de Datagramas de Usuario" o "UDP".

¿Por qué son sin conexión?

Bueno, básicamente porque no tienes que mantener una conexión abierta como harías con los sockets de flujo. Simplemente montas un paquete, le metes una cabecera IP con la información de destino y lo envías. No se necesita conexión. Generalmente se usan para transferencias de información por paquetes. Aplicaciones que usan este tipo de sockets son, por ejemplo, tftp y bootp.

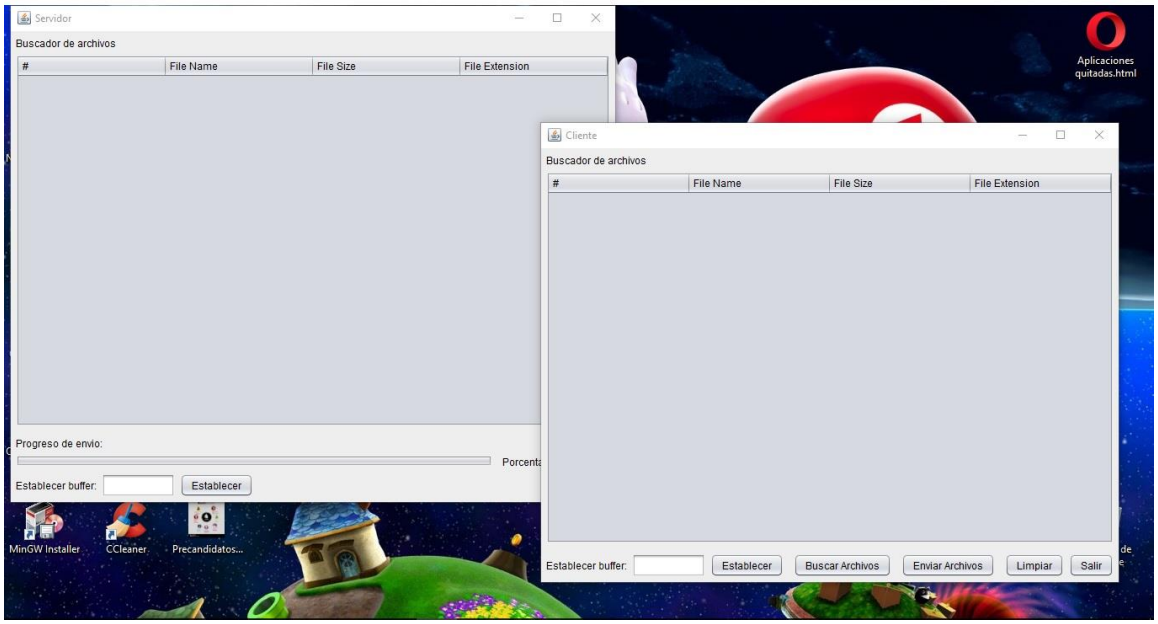
Sockets en Java

Los sockets son básicamente formas en las que podemos interconectar 2 (o más) programas mediante el uso de la internet. En java se utilizan para poder crear conexiones utilizando básicamente una IP/hostname y un puerto para establecer la conexión. Para aprender podemos utilizarla para conectar 2 programas por medio de Internet.

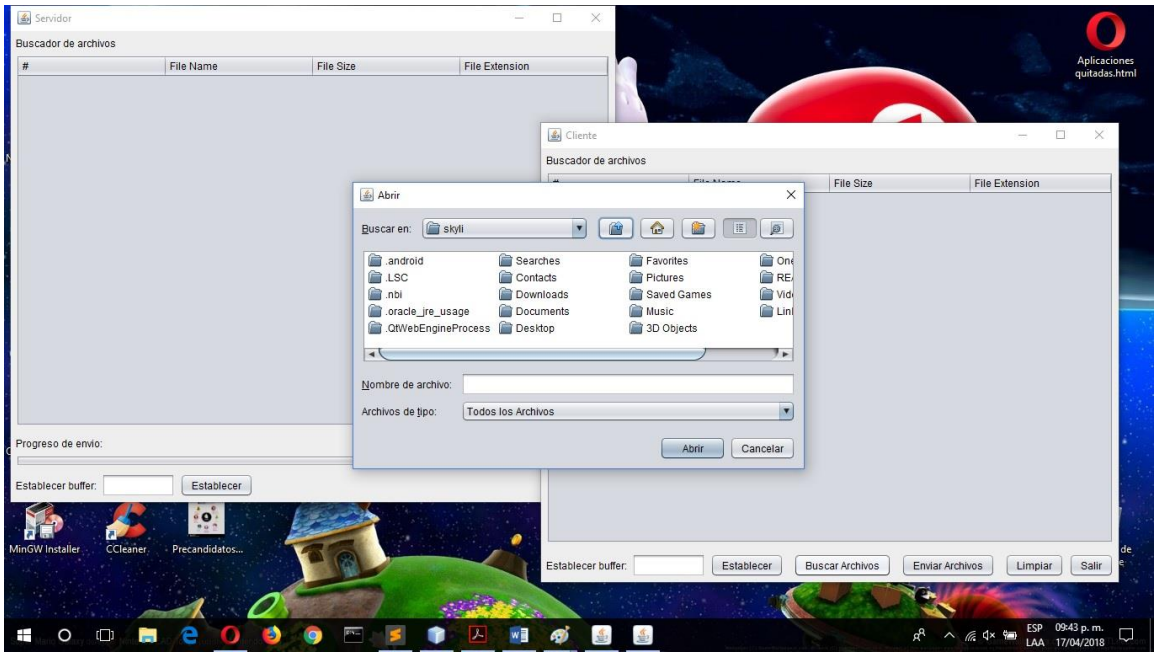
¿Cómo funciona?

El modelo más básico de los sockets consta de 2 simples programas, un servidor y un cliente. Básicamente el programa servidor comienza a “escuchar” en un puerto determinado (nosotros lo especificamos), y posteriormente el programa que la hace de “cliente” debe conocer la ip o nombre de dominio/hostname del servidor y el puerto que está escuchando, al saber esto simplemente solicita establecer una conexión con el servidor. Es aquí cuando el servidor acepta esa conexión y se puede decir que estos programas están “conectados”, de este modo pueden intercambiar información.

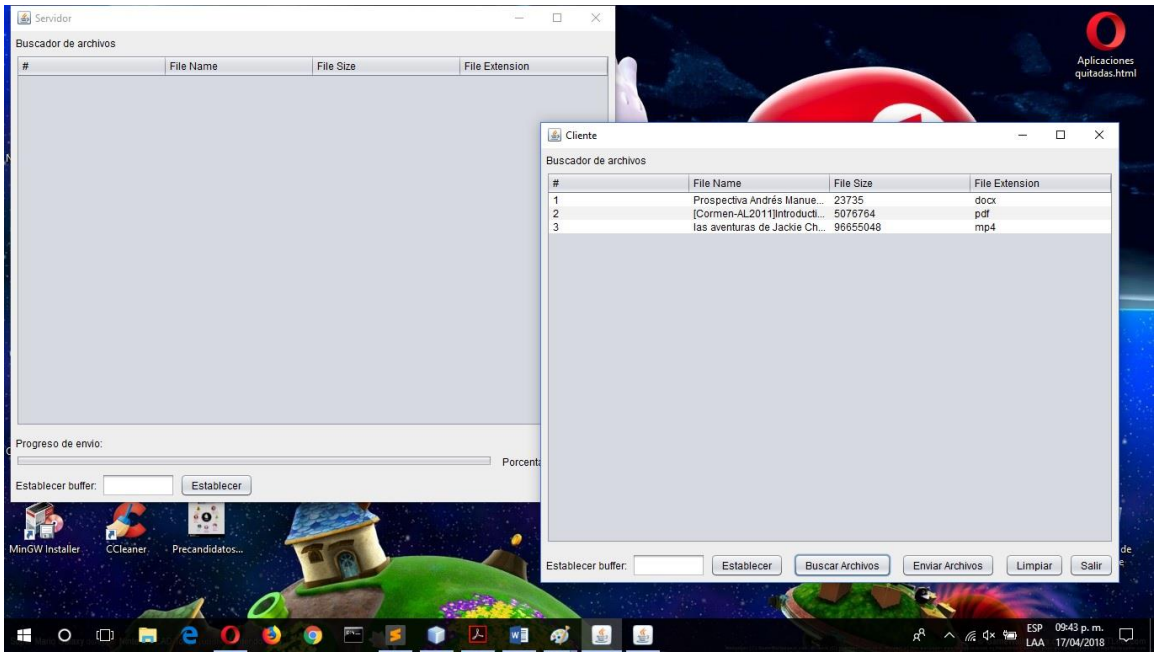
DESARROLLO



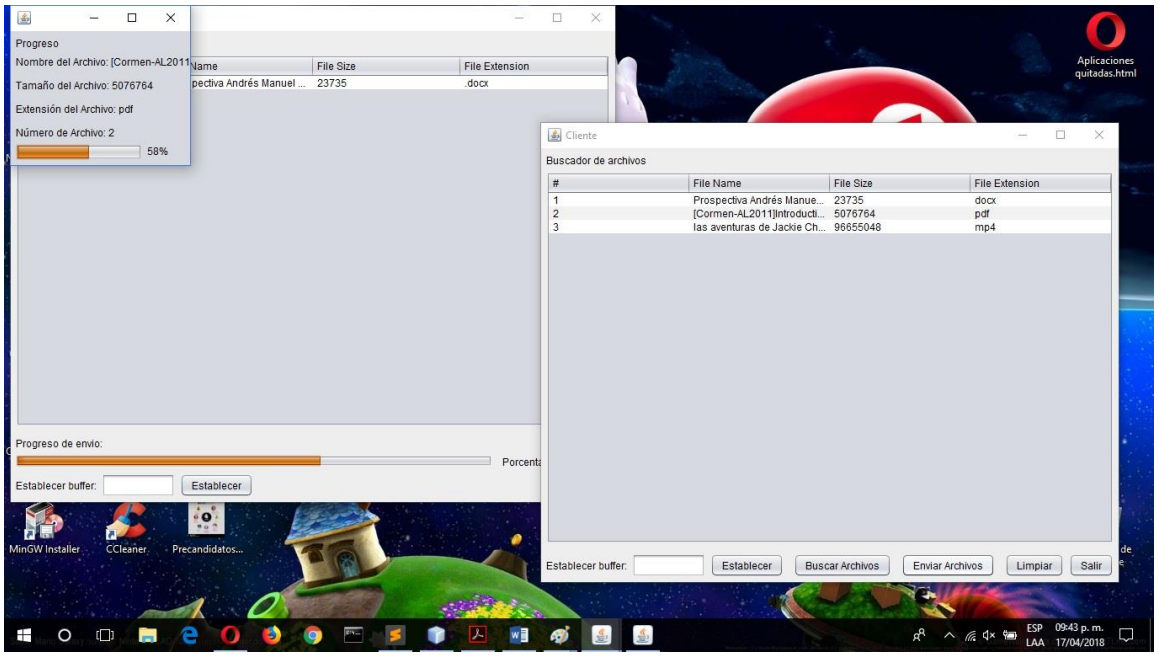
Inicialización del cliente y del servidor

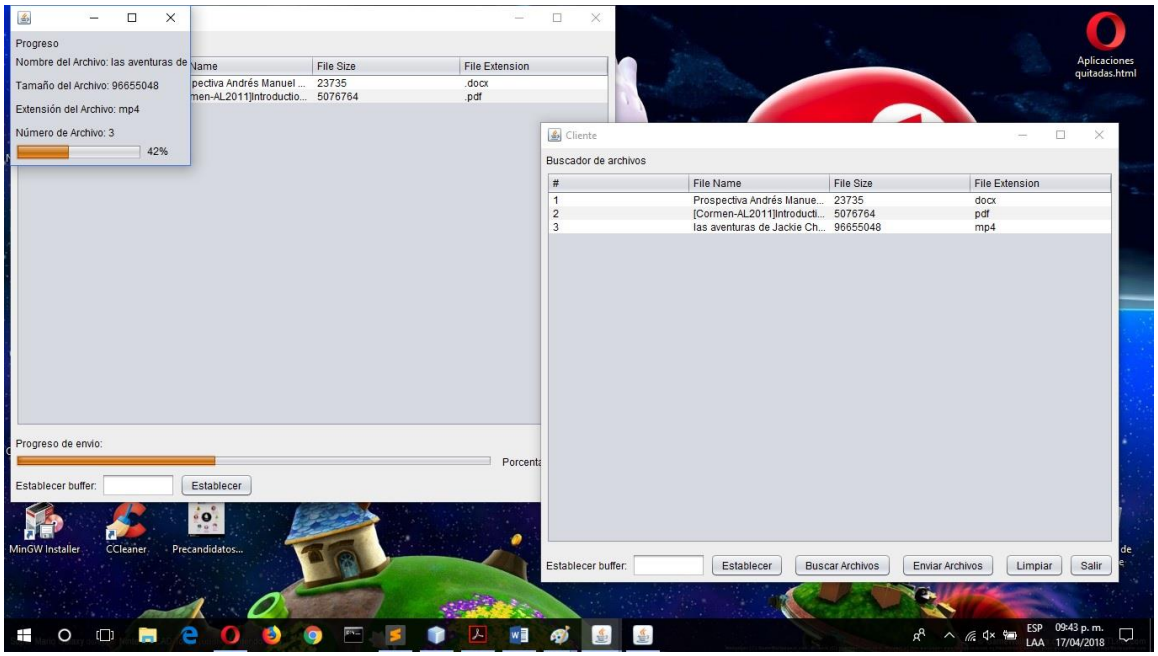


Carga de los archivos a mandar.

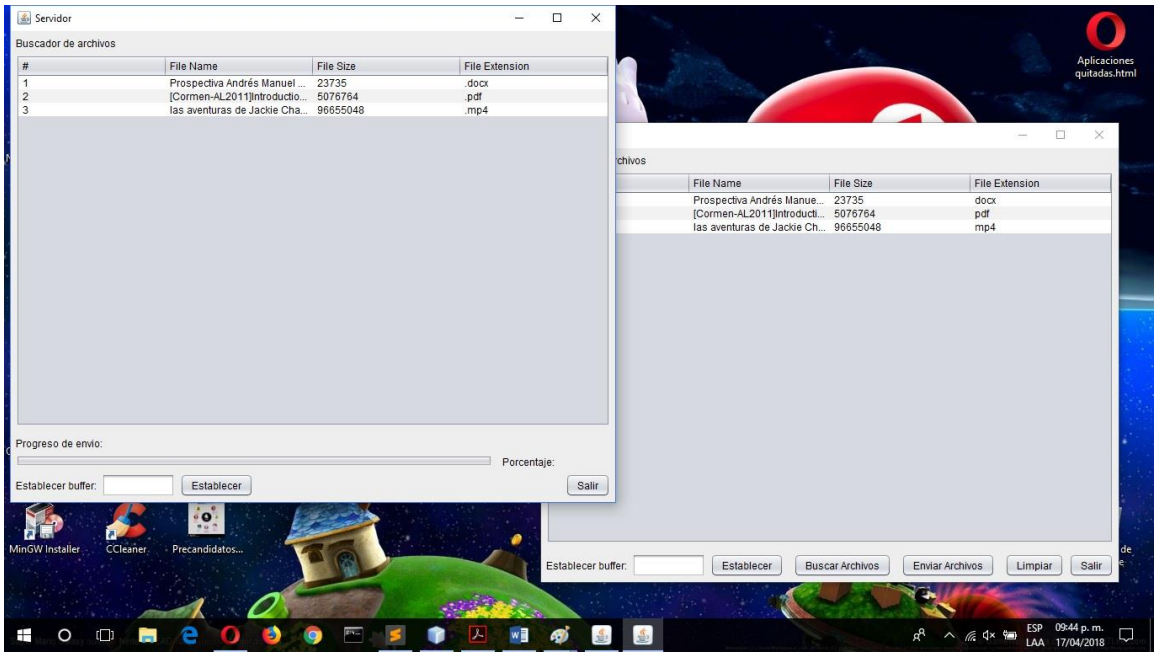


Archivos ya cargados





Envió de los archivos



Recepción de los archivos

Clase Servidor

```
public class Servidor extends Thread{

   private int                 port;
   private int                 sizeBuffer;
   private int                 sizeFile;
   private int                 numberOfFiles;
   private int                 percent;
   private byte []              buffer;
   private String               nameFile;
   private String               direction;
   private String               extensionFile;
   private Socket               socketClient;
   private ServerSocket          socketServer;
   private DataInputStream       dataInputStream1;
   private DataOutputStream      dataOutputStream1;
   private BufferedInputStream   bufferedInputStream1;
   private BufferedOutputStream  bufferedOutputStream1;
   private Menu                  menu;
   private int                 idFile;
   private BarraDeProgreso       barra;
   private Object [][]           o;
   private ArrayList<Object>     obj;

   public Servidor(){
      this.port         = 4000;
      this.sizeBuffer   = 102400;
```

```
    this.buffer         = new byte[this.sizeBuffer];
}

public Servidor(int port){
    this.port           = port;
    this.sizeBuffer     = 102400;
    this.buffer         = new byte[this.sizeBuffer];
}

public void upConexion(){
    try {
        this.socketServer = new ServerSocket(this.port);
    } catch (IOException ex) {
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void recieveNumberOfFiles(){
    try {
        this.dataInputStream1 = new DataInputStream(socketClient.getInputStream());
        this.numberOfFiles = this.dataInputStream1.readInt();
        System.out.println("Número del Archivo: " + this.numberOfFiles);
    } catch (IOException ex) {
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
    public void recieveDataFile(int idFile){
        try {
            this.idFile = idFile;
            System.out.println("\nNúmero del archivo: " + String.valueOf(idFile));
            this.sizeFile = this.dataInputStream1.readInt();
            this.nameFile = this.dataInputStream1.readUTF();
            this.extensionFile = this.dataInputStream1.readUTF();
            System.out.println("\nNombre del archivo: " + this.nameFile);
            System.out.println("\nExtensión del archivo: " + this.extensionFile);
            System.out.println("\nTamaño del Archivo: " + this.sizeFile);

        } catch (IOException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void recieveFile(){
        try {
            this.bufferedInputStream1 = new
BufferedInputStream(socketClient.getInputStream());
            this.bufferedOutputStream1 = new BufferedOutputStream(new
FileOutputStream(this.nameFile+this.extensionFile));
            int bytesRecieved;
            double counter = 0;
            double p = 0.0;
            while((bytesRecieved = this.bufferedInputStream1.read(this.buffer)) != -1){
                this.bufferedOutputStream1.write(buffer, 0, bytesRecieved);
                counter += bytesRecieved;
```

```
            p = (counter*100)/(this.sizeFile);
            this.percent = (int) p;
            this.barra.setProgress(nameFile, extensionFile, idFile, sizeFile, percent);
            System.out.println("Este es el porcentaje: " + this.percent);
        }
        this.bufferedOutputStream1.close();
        this.bufferedInputStream1.close();
        System.out.println("Archivo Recivido");
    } catch (IOException ex) {
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void recieve(){
    barra = new BarraDeProgreso();
    while(true){
        try {
            this.socketClient = this.socketServer.accept();
            recieveNumberOfFiles();
            barra.setProgress(String.valueOf(this.numberOfFiles));
            barra.setVisible(true);
            System.out.println("Este es el numero de archivos: " + this.numberOfFiles);
            obj = new ArrayList<Object>();
            for(int i = 0; i < this.numberOfFiles; i++){
                this.socketClient = this.socketServer.accept();
                recieveDataFile(i+1);
                recieveFile();
```

```
                obj.add(new Object[]{
                    this.idFile,
                    this.nameFile,
                    this.sizeFile,
                    this.extensionFile
                });
            }
            this.dataInputStream1.close();
            this.barra.setVisible(false);
        } catch (IOException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

public void destroyer(){
    this.dataInputStream1 = null;
    this.numberOfFiles = 0;
    this.sizeFile = 0;
    this.nameFile = null;
    this.extensionFile = null;
    this.bufferedInputStream1 = null;
    this.bufferedOutputStream1 = null;
}

@Override
public void run(){
```

```
    upConexion();
    System.out.println("Servidor en escucha...");
    recieve();
}

public int getSizeBuffer() {
    return sizeBuffer;
}

public void setSizeBuffer(int sizeBuffer) {
    this.sizeBuffer = sizeBuffer;
}

public void colocateBufferMenu(int p){
    try {
        this.wait();
        menu.setBuffer(p);
        this.notify();
        //this.setSizeBuffer(p);
    } catch (InterruptedException ex) {
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public Menu getMenu(){
    return menu;
}
```

```
    public ArrayList<Object> getFiles(){
        return obj;
    }

}
```

Clase Cliente

```
public class Cliente extends Thread{

    private int                 port;
    private int                 percent;
    private int                 sizeBuffer;
    private byte []              buffer;
    private String               direction;
    private Socket               socketClient;
    private DataOutputStream       dataOutputStream1;
    private BufferedInputStream    bufferedInputStream1;
    private BufferedOutputStream   bufferedOutputStream1;
    private File []              files;
    private BarraDeProgreso        Barra;

    public Cliente(){
        this.port          = 4000;
        this.direction     = "127.0.0.1";//"192.168.0.3";
        this.sizeBuffer    = 102400;
```

```
        this.buffer         = new byte[this.sizeBuffer];
    }

    public Cliente(String direction, int port){
        this.port           = port;
        this.direction      = direction;
        this.sizeBuffer     = 102400;
        this.buffer         = new byte[this.sizeBuffer];
    }

    public void upConexion(){
        try {
            this.socketClient = new Socket(this.direction, this.port);
            System.out.println("Cliente conectado...");
        } catch (IOException ex) {
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void sendNumberOfFiles(int numberOfFiles){
        try {
            this.dataOutputStream1 = new
DataOutputStream(socketClient.getOutputStream());
            this.dataOutputStream1.writeInt(numberOfFiles);
        } catch (IOException ex) {
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
```

```
        }
    }

    public void sendDataFile(int sizeFile, String nameFile, String extensionFile){
        try {
            this.dataOutputStream1.writeInt(sizeFile);
            this.dataOutputStream1.writeUTF(nameFile);
            this.dataOutputStream1.writeUTF(extensionFile);
        } catch (IOException ex) {
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void sendFile(File file, int id, int size){
        try {
            this.bufferedInputStream1 = new BufferedInputStream(new
FileInputStream(file.getAbsolutePath()));
            this.bufferedOutputStream1 = new
BufferedOutputStream(this.socketClient.getOutputStream());
            int byteSent = 0;
            double counter = 0.0;
            double p = 0.0;
            while((byteSent = this.bufferedInputStream1.read(this.buffer)) != -1){
                this.bufferedOutputStream1.write(this.buffer, 0, byteSent);
                counter += byteSent;
                p = (counter*100)/((int)file.length());
```

```
            this.percent = (int) p;
            Thread.sleep(5);
            this.Barra.setProgress(file.getName().substring(0,
file.getName().lastIndexOf('.')), file.getName().substring(file.getName().lastIndexOf('.')+1),
id, size, this.percent);
            Thread.sleep(5);
          }
          this.bufferedInputStream1.close();
          this.bufferedOutputStream1.close();
          System.out.println("Archivo Enviado.");
      } catch (FileNotFoundException ex) {
          Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
      } catch (IOException ex) {
          Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
      } catch (InterruptedException ex) {
          Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
      }

  }

  public void send(){
      this.Barra = new BarraDeProgreso();
      upConexion();
      sendNumberOfFiles(files.length);
      System.out.println("Este es el numero de archivos a Mandar: " + files.length);
      for(int i = 0; i < files.length; i++){
```

```
            this.Barra.setVisible(true);

            this.Barra.setProgress(0);

            upConexion();

            System.out.println("\nNúmero de archivo: " + String.valueOf(i+1));

            System.out.println("\nNombre del archivo: " + files[i].getName().substring(0,
files[i].getName().lastIndexOf('.')));

            System.out.println("\nExtensión del archivo: " + "." +
files[i].getName().substring(files[i].getName().lastIndexOf('.')+1));

            System.out.println("\nTamaño en bytes del archivo: " +
String.valueOf(files[i].length()));

            sendDataFile((int) files[i].length(), files[i].getName().substring(0,
files[i].getName().lastIndexOf('.')),
'.'+files[i].getName().substring(files[i].getName().lastIndexOf('.')+1));

            sendFile(files[i], i+1, (int)files[i].length());

            this.Barra.setVisible(false);

        }

        try {

            this.dataOutputStream1.close();

        } catch (IOException ex) {

            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);

        }

        try {

            this.socketClient.close();

        } catch (IOException ex) {

            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);

        }

    }
```

```
@Override
public void run(){
    System.out.println("Cliente Mandando Archivos...");
    send();
}

public int getPort() {
    return port;
}

public void setPort(int port) {
    this.port = port;
}

public int getPercent() {
    return percent;
}

public void setPercent(int percent) {
    this.percent = percent;
}

public int getSizeBuffer() {
    return sizeBuffer;
}
```

```
public void setSizeBuffer(int sizeBuffer) {
    this.sizeBuffer = sizeBuffer;
}

public byte[] getBuffer() {
    return buffer;
}

public void setBuffer(byte[] buffer) {
    this.buffer = buffer;
}

public String getDirection() {
    return direction;
}

public void setDirection(String direction) {
    this.direction = direction;
}

public Socket getSocketClient() {
    return socketClient;
}

public void setSocketClient(Socket socketClient) {
    this.socketClient = socketClient;
```

```
    }

    public DataOutputStream getDataOutputStream1() {
        return dataOutputStream1;
    }

    public void setDataOutputStream1(DataOutputStream dataOutputStream1) {
        this.dataOutputStream1 = dataOutputStream1;
    }

    public BufferedInputStream getBufferedInputStream1() {
        return bufferedInputStream1;
    }

    public void setBufferedInputStream1(BufferedInputStream bufferedInputStream1) {
        this.bufferedInputStream1 = bufferedInputStream1;
    }

    public BufferedOutputStream getBufferedOutputStream1() {
        return bufferedOutputStream1;
    }

    public void setBufferedOutputStream1(BufferedOutputStream bufferedOutputStream1)
{
        this.bufferedOutputStream1 = bufferedOutputStream1;
    }
```

```
    public File[] getFiles() {
        return files;
    }

    public void setFiles(File[] files) {
        this.files = files;
    }

    public BarraDeProgreso getBarra() {
        return Barra;
    }

    public void setBarra(BarraDeProgreso Barra) {
        this.Barra = Barra;
    }
```

Clase para buscar los archivos

```
public class BuscadorDeArchivos extends javax.swing.JFrame {

    private JFileChooser chooser;
    private File [] files;

    public BuscadorDeArchivos(){
        this.chooser = new JFileChooser();
```

```
        this.chooser.setMultiSelectionEnabled(true);
    }

    public void seacherFiles(){
        int option = chooser.showOpenDialog(this);
        if(option == JFileChooser.APPROVE_OPTION){
            files = chooser.getSelectedFiles();
        }
    }

    public File[] getFiles() {
        return files;
    }

    public void setFiles(File[] files) {
        this.files = files;
    }

    public void destroyer(){
        this.files = null;
        this.chooser = null;
    }

}
```

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 un sistema usado para mejorar la eficiencia de las redes, sobre todo las de Internet. El sistema intenta evitar que los datos sean enviados en lotes demasiado pequeños, lo cual aumenta el número de lotes. Mientras está en uso, este algoritmo no suele interactuar demasiado con otros elementos de la red de comunicaciones. Funciona con redes que usan protocolos TCP/IP. Estos protocolos le dicen a la red como transmitir los datos. Mientras que se pueden aplicar a cualquier red, suelen asociarse directamente a Internet.

Se podría aplicar en la parte de la segmentación del archivo ya que como dice se está intentando evitar el lote de pequeños paquetes, la forma de aplicar este algoritmo es aumentar el tamaño de los segmentos del archivo.

La ventaja que ofrece es que no se tiene que tratar con tantos segmentos del archivo, disminuye considerablemente los segmentos del archivo.

La desventaja que ofrece es que si hay una pérdida de algún segmento del paquete, la pérdida del ese segmentos es considerable y va a ser mucho más difícil de rearmar el archivo.

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

Aquellos archivos que tenían un menor peso en cuanto a bytes

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

Se hiso la prueba de enviar solo unos cuantos en este caso 10 archivos distintos de diferentes pesos, pero como tal, no hay un límite en el programa, el limite lo pone la misma maquina en donde está el servidor refiriéndose a la parte del espacio libre en el disco duro

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

Se han alcanzado a enviar por el momento solo un archivo de 1 GB, esto gracias a que se incrementó el buffer de los programas

5. ¿Qué es el orden de red?

Se refiere a la prioridad de red que se le asigna al envío de paquetes a través del establecimiento de prioridades. Está relacionado con el control de flujo de datos.

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

- Porque en una computadora actual no hay un solo socket utilizando un puerto físico de salida. Tiene que ser compartido por distintos procesos y es necesario establecer una prioridad de envío y recepción.

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?

De acuerdo con lo realizado, el programa transforma al archivo en un flujo de bits, permitiendo al servidor recibirlos y basta con que el reordenamiento de los paquetes sea correcto para que, no importando el tamaño del archivo, siempre se pueda enviar.

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?

Pues sería cambiar los archivos a tamaños de datos como pueden ser long, pero más que nada se puede incrementar un poco más el límite de los buffer, al igual de ver en qué tipo de dato es mejor enviarlo.

CONCLUSIONES

Los sockets permiten al usuario establecer, en caso de serlo, una conexión con otra computadora y proporcionar una manera simple de realizar la transferencia de datos. Dichos datos no sólo se limitan a archivos, también puede ser enviada información, mensajes o instrucciones remotas. Las aplicaciones de sockets sólo se ven limitadas por los paradigmas del desarrollador.

– Erick Gabriel Mendoza Flores –

Los sockets son de gran ayuda, pues permiten comunicar a varios procesos dentro de una computadora con otra, y de igual forma, que estos puedan llegar a ser bloqueantes y con ello se logre que tengan una sincronización propia para poder tener una recepción de datos mejor hecha, hace que sean útiles en su uso, por otro lado, los requerimientos de un buffer así como los requerimientos de la computadora, hacen que la recepción de los datos sea muy diferente y con ello, no se logre tener con certeza un tiempo estimado para el envio y la obtención de los datos.

– Luis Roberto Nava Lara –

Referencias

[1] Diseño web España. (1997). ¿Qué es un Socket? 2017, de masadelante.com Sitio web: http://www.masadelante.com/faqs/socket

[2] Jorge Villalobos. (2005). sockets en java. 2016, de www.codigoprogramacion-com Sitio web: http://codigoprogramacion.com/cursos/java/103-sockets-en-java-con-cliente-y-servidor.html#.WeGSpo_WzIV

[3] S/N. (2014). ¿Qué es el algoritmo de nagle?. 2016, de ordenadores y portátiles Sitio web: http://www.ordenadores-y-portatiles.com/nagle.html

[4] Damián Marques Lluch. (1995). ¿Qué es un socket?. 2001, de guía beej de programación en redes Sitio web: http://www.tyr.unlu.edu.ar/tyr/TYR-trab/satobigal/documentacion/beej/index.html