

PRÁCTICA 2

Viktor Yosava

`vikyosava@uma.es`

Redes y Sistemas Distribuidos. Ingeniería de la Salud

TAREA 1: Desarrollo de la aplicación de eco usando TCP

El código que se ha desarrollado para la realización de este ejercicio de la práctica es el siguiente:

```
package client;
import java.io.*;

public class ClientTCP {
    public static void main(String[] args) throws IOException {
        String serverName = "0.0.0.0";
        int portNumber = 1111;
        Socket socket = null;

        PrintWriter out = null;
        BufferedReader in = null;

        try {
            InetAddress serverAddr = InetAddress.getByName(serverName);
            socket = new Socket(serverAddr, portNumber);
            System.out.println("Conexión local " + serverAddr);
            System.out.println("-----");

            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())), true);

        } catch (UnknownHostException e) {
            System.err.println("Unknown: " + serverName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for " + "the connection to: " + serverName);
            System.exit(1);
        }

        System.out.println("STATUS: Conectado al servidor ");
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        String userInput;

        System.out.println("Introduzca un texto a enviar (END para acabar)");
        System.out.println("-----");

        userInput = stdIn.readLine();
        out.println(userInput);
        while (userInput.compareTo("END") != 0) {
            out.println(userInput);
            System.out.println("STATUS: Enviando " + userInput);
            System.out.println("STATUS: Esperando eco");
            String echo = null;
            echo = in.readLine();
            System.out.println("echo: " + echo);
            System.out.println("Introduzca un texto a enviar (END para acabar)");
            System.out.println("-----");
            userInput = stdIn.readLine();
        }

        System.out.println("STATUS: El cliente quiere terminar el servicio");
        out.println(userInput);
        System.out.println("STATUS: Sending " + userInput);
        System.out.println("STATUS: Waiting for the reply");
        String ok = in.readLine();
        System.out.println("STATUS: Cerrando conexion " + ok);
        in.close();
        out.close();
        stdIn.close();
        socket.close();
        System.out.println("STATUS: Conexion cerrada");
        System.out.println("-----");
    }
}
```

Fig. 1. ClientTCP

ClientTCP es el código que corresponde a la parte del cliente. Inicia la conexión con el servidor y envía las cadenas que deben ser procesadas y las muestra.

El cliente puede cerrar la conexión enviando la cadena **END**.

```

package server;

import java.io.*;

class ServerTCP {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSock = null;
        Socket socket = null;
        BufferedReader in = null;
        PrintWriter out = null;
        String line;
        int port = 1111;
        try {
            serverSock = new ServerSocket(port);
            System.out.println("Servidor iniciado "+serverSock);
            System.out.println("-----");
        } catch (IOException e) {
            System.out.println("Could not listen on port " + port);
            System.exit(-1);
        }

        while (true)
        {
            try {
                socket = serverSock.accept();
                System.out.println("Nuevo cliente, socke "+socket);
                System.out.println("-----");
            } catch (IOException e) {
                System.out.println("Accept failed: " + port);
                System.exit(-1);
            }
            try {
                in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())),true);
            } catch (IOException e) {
                System.out.println("Exception " + e);
                System.exit(-1);
            }
            boolean end = false;
            while (!end) {
                try {
                    line = in.readLine();
                    System.out.println("Received from client " + line);
                    if (line.compareTo("END") != 0) {
                        String cadenaMayuscula = line.toUpperCase();
                        String cadenaFinal = (new StringBuffer(cadenaMayuscula)).reverse().toString();
                        line=cadenaFinal;
                    } else
                    {
                        line="OK";
                        end = true;
                    }
                    System.out.println("Sending to client "+line);
                    System.out.println("-----");
                    out.println(line);
                } catch (Exception e) {
                    System.out.println("Read failed");
                    System.exit(-1);
                }
            }
            System.out.println("Closing connection with the client");

            in.close();
            out.close();
            socket.close();
            serverSock.close();
            System.out.println("Waiting for a new client");
            System.out.println("-----");
        }
    }
}

```

Fig. 2. ServerTCP

ServerTCP recibe las cadenas del cliente, las revierte y convierte las letras en mayúsculas. En caso de recibir la cadena **END** devuelve **OK** y se cierra la conexión.

TAREA 2: Desarrollo de la aplicación de eco usando UDP

El código que se ha desarrollado para la realización de este ejercicio de la práctica es el siguiente:

```
package client;

import java.io.BufferedReader;

public class ClientUDP {
    public static void main(String[] args) throws IOException {
        String serverName = "127.0.0.1";
        int serverPort = 2222;
        InetAddress serverAddress = null;
        String input = "";
        DatagramSocket socket = new DatagramSocket();
        byte[] bytes = new byte[1024];
        DatagramPacket pack = null;
        String mensajeRecibido = "";
        Scanner sc = new Scanner(System.in);
        try {
            serverAddress = InetAddress.getByName("127.0.0.1");
        } catch (UnknownHostException e) {
            System.err.println("Unknown: " + serverName);
            System.exit(1);
        }
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Introduzca un texto a enviar (END para acabar)");
        System.out.println("-----");

        input = stdIn.readLine();
        while (input.compareTo("END") != 0) {
            try {
                pack = new DatagramPacket(input.getBytes(), input.getBytes().length, serverAddress, serverPort);
                socket.send(pack);
            } catch (IOException ioe) {
            }
            System.out.println("STATUS: Waiting for the reply");

            pack = new DatagramPacket(bytes, bytes.length);
            socket.receive(pack);

            mensajeRecibido = new String(Arrays.copyOfRange(pack.getData(), pack.getOffset(), pack.getOffset() + pack.getLength()));
            System.out.println("echo: " + mensajeRecibido);
            System.out.println("Introduzca un texto a enviar (END para acabar)");

            input = stdIn.readLine();
        }
        System.out.println("STATUS: Closing client");

        socket.close();
        sc.close();
        System.out.println("STATUS: CLOSED");
        System.out.println("-----");
    }
}
```

Fig. 3. ClientUDP

ClientUDP es el código que corresponde a la parte del cliente. Envía las cadenas que deben ser procesadas al servidor. Puede cerrar la conexión enviando **END**.

```

package server;

/* To change this template, choose Tools | Templates */
import java.io.IOException;

public class ServerUDP {

    public static void main(String[] args) throws IOException {
        DatagramSocket serverSock = null;
        DatagramPacket out = null;
        DatagramPacket in = null;
        String resultado = "";
        byte[] bytes;
        String line = "";
        int port = 2222;

        try {
            serverSock = new DatagramSocket(port);
            System.out.println("Servidor encendido, esperando conexion");
            System.out.println("-----");

        } catch (IOException e) {
            System.out.println("Could not listen on port " + port);
            System.exit(-1);
        }

        while (true)
        {
            System.out.println("Waiting for a new UDP client");
            System.out.println("-----");

            bytes = new byte[20];
            in = new DatagramPacket(bytes, bytes.length);
            serverSock.receive(in);
            line = new String(Arrays.copyOfRange(in.getData(), in.getOffset(),
                in.getOffset() + in.getLength()));
            System.out.println("Direccion socket del cliente: " + in.getAddress());
            if (line.compareTo("END") != 0) {
                resultado = modificarCadena(line);
                System.out.println("Sending to client " + resultado);
                System.out.println("-----");
            }

            out = new DatagramPacket(bytes, bytes.length);

            int puertoCliente = in.getPort();
            InetAddress direccion = in.getAddress();
            bytes = resultado.getBytes();
            out = new DatagramPacket(bytes, bytes.length, direccion, puertoCliente);

            serverSock.send(out);
            System.out.println("STATUS: Echo sent ");
            System.out.println("STATUS: Waiting for new echo");
            System.out.println("-----");

            if (line.equalsIgnoreCase("END")) {
                System.out.println("Conexion con cliente finalizada");
                System.out.println("-----");
            }
        }
    }

    public static String modificarCadena(String cadena) {
        String cadenaMayuscula = cadena.toUpperCase();
        String cadenaRevertida = (new StringBuffer(cadenaMayuscula)).reverse().toString();
        return cadenaRevertida;
    }
}

```

Fig. 4. ServerUDP

ServerUDP espera a que un cliente se conecte y envíe cadenas que deban ser procesadas para convertirlas en mayúsculas y revertidas o para terminar la conexión en caso de recibir **END**.

TAREA 3: Análisis de comunicaciones de los servicios desarrollados

Ejercicio 1. Prueba local

Prueba TCP

Podemos observar el proceso completo en las siguientes imágenes:

```
Servidor iniciado ServerSocket[addr=0.0.0.0/0.0.0.0,localport=1111]
-----
```

Fig. 5. Primera ejecución de ServerTCP

El servidor hace posible la conexión.

```
Conexión local /0.0.0.0
-----
STATUS: Conectado al servidor
Introduzca un texto a enviar (END para acabar)
-----
abc
STATUS: Enviando abc
STATUS: Esperando eco
echo: CBA
Introduzca un texto a enviar (END para acabar)
-----
```

Fig. 6. Primera ejecución de ClientTCP

La conexión ha sido completada. El cliente envía una cadena para que el servidor la procese y la devuelva.

```
Servidor iniciado ServerSocket[addr=0.0.0.0/0.0.0.0,localport=1111]
-----
Nuevo cliente, socke Socket[addr=/192.168.1.3,port=60984,localport=1111]
-----
Received from client END
Sending to client OK
-----
Closing connection with the client
Waiting for a new client
-----
```

Fig. 7. Cierre de conexión

Enviando **END** se termina la conexión.

118.10.223104	127.0.0.1	127.0.0.1	TCP	56.51915 → 1111 [SYN] Seq=0 Win=0 Len=0 MSS=65535 WS=256 SACK_PERM
119.50.222206	127.0.0.1	127.0.0.1	TCP	56.1111 → 51915 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=65535 WS=256 SACK_PERM
120.10.223222	127.0.0.1	127.0.0.1	TCP	44.51915 → 1111 [ACK] Seq=1 Ack=1 Win=2019648 Len=0
130.13.342323	127.0.0.1	127.0.0.1	TCP	49.51915 → 1111 [PSH, ACK] Seq=1 Ack=1 Win=2019648 Len=5
131.13.342391	127.0.0.1	127.0.0.1	TCP	44.1111 → 51915 [ACK] Seq=1 Ack=0 Win=2019648 Len=0
132.13.342755	127.0.0.1	127.0.0.1	TCP	49.51915 → 1111 [PSH, ACK] Seq=0 Ack=1 Win=2019648 Len=5
133.13.342790	127.0.0.1	127.0.0.1	TCP	44.1111 → 51915 [ACK] Seq=1 Ack=1 Win=2019648 Len=0
134.13.343953	127.0.0.1	127.0.0.1	TCP	49.1111 → 51915 [PSH, ACK] Seq=1 Ack=1 Win=2019648 Len=5
135.13.344052	127.0.0.1	127.0.0.1	TCP	44.51915 → 1111 [ACK] Seq=11 Ack=0 Win=2019648 Len=0
136.13.344109	127.0.0.1	127.0.0.1	TCP	49.1111 → 51915 [PSH, ACK] Seq=0 Ack=1 Win=2019648 Len=5
137.13.344415	127.0.0.1	127.0.0.1	TCP	44.51915 → 1111 [ACK] Seq=11 Ack=1 Win=2019648 Len=0

Fig. 8. Tramas TCP WireShark

Filtrando las tramas por `tcp.port == 1111` podemos observar los múltiples procesos que hacen al protocolo **TCP** seguro y orientado a la conexión.

Prueba UDP

Podemos observar el proceso completo en las siguientes imágenes:

```
Servidor encendido, esperando conexion
-----
Waiting for a new UDP client
```

Fig. 9. Primera ejecución de ServerUDP

El servidor espera a que un cliente se le conecte.

```
Introduzca un texto a enviar (END para acabar)
-----
abc
STATUS: Waiting for the reply
echo: CBA
Introduzca un texto a enviar (END para acabar)
```

Fig. 10. Primera ejecución de ClientUDP

El cliente establece la conexión y envía una cadena que es procesada por el servidor.

```
Servidor encendido, esperando conexion
-----
Waiting for a new UDP client
-----
Direccion socket del cliente:/127.0.0.1:54639
Sending to client CBA
-----
STATUS: Echo sent
STATUS: Waiting for new echo
-----
```

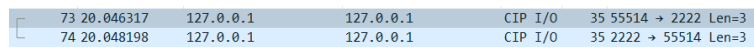
Fig. 11. Respuesta de ServerUDP

Vemos como ServerUDP le atribuye un socket a ClientUDP y envía la cadena procesada.

```
Introduzca un texto a enviar (END para acabar)
END
STATUS: Closing client
STATUS: CLOSED
```

Fig. 12. Cierre de conexión

Enviando **END** el cliente se desconecta del servidor, esta cadena no es recibida por el servidor ya que **UDP** no está orientado a la conexión.



73	20.046317	127.0.0.1	127.0.0.1	CIP I/O	35 55514 → 2222	Len=3
74	20.048198	127.0.0.1	127.0.0.1	CIP I/O	35 2222 → 55514	Len=3

Fig. 13. Tramas UDP WireShark

UDP presenta muchas menos tramas en la captura de WireShark al ser un protocolo no fiable.

Ejercicio 2. Varios clientes

TCP

```
Conexión local /127.0.0.1
-----
STATUS: Conectado al servidor
Introduzca un texto a enviar (END para acabar)
-----
a
STATUS: Enviando a
STATUS: Esperando eco
```

Fig. 14. Intento de conexión de Segundo Cliente TCP

En este caso el servidor **TCP** no recibe al segundo cliente.

UDP

```
Servidor encendido, esperando conexion
-----
Waiting for a new UDP client
-----
Direccion socket del cliente:/127.0.0.1:55514
Sending to client CBA
-----
STATUS: Echo sent
STATUS: Waiting for new echo
-----
Waiting for a new UDP client
-----
Direccion socket del cliente:/127.0.0.1:62453
Sending to client CBA
-----
STATUS: Echo sent
STATUS: Waiting for new echo
-----
Waiting for a new UDP client
-----
```

Fig. 15. Conexión de dos Clientes UDP

73	20.046317	127.0.0.1	127.0.0.1	CIP I/O	35 55514 → 2222	Len=3
74	20.048198	127.0.0.1	127.0.0.1	CIP I/O	35 2222 → 55514	Len=3
154	66.076795	127.0.0.1	127.0.0.1	CIP I/O	35 62453 → 2222	Len=3
155	66.077130	127.0.0.1	127.0.0.1	CIP I/O	35 2222 → 62453	Len=3

Fig. 16. Tramas UDP WireShark con varios Clientes

Podemos observar que **UDP** sí permite que se lleve a cabo la conexión de varios clientes.

Ejercicio 3.

```
ServerTCP [Java Application] C:\Users\vikto\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspc
Servidor iniciado ServerSocket[addr=0.0.0.0/0.0.0.0,localport=1111]
-----
ServerUDP [Java Application] C:\Users\vikto\.p2\pool\plugins\org
Servidor encendido, esperando conexion
-----
Waiting for a new UDP client
-----
```

Fig. 17. Servidores TCP y UDP encendidos simultaneamente

```
ClientTCP [Java Application] C:\Users\vikto\.p2\pool\plugins\org.ec
Conexión local /127.0.0.1
-----
STATUS: Conectado al servidor
Introduzca un texto a enviar (END para acabar)
-----
ClientUDP [Java Application] C:\Users\vikto\.p2\pool\plugins\org.ec
Introduzca un texto a enviar (END para acabar)
-----
```

Fig. 18. Clientes TCP y UDP encendidos simultaneamente

Podemos ver que, aunque compartan el mismo puerto *1111*, los scripts se siguen pudiendo ejecutar simultaneamente sin problema ya que **TCP** y **UDP** utilizan protocolos diferentes.