

Práctica 2 - Redes y Sistemas Distribuidos

22 de Noviembre del 2023



SERVIDOR Y CLIENTE TCP

Requisitos TCP

Cliente

Debe permitir al usuario introducir una cadena desde el teclado, la cual se envía a través de un socket TCP al servidor

Servidor

Debe devolver esa cadena revertida y en mayúsculas al cliente, que muestra por pantalla.

Interacciones

En el servidor, se debe indicar la dirección IP y el puerto del cliente que se conecte

Se debe cerrar esa conexión tras escribir por teclado “END” que tras recibirla el servidor, este responde con “OK”.

Caso de Uso (1)

1. Inicio Server

- Se ejecuta el servidor
- El servidor crea un socket en el puerto especificado y entra en un bucle infinito para esperar conexiones de clientes.
- Muestra el mensaje "Waiting for a new client" en la consola.

2. Inicio Cliente

- Se ejecuta el cliente
- El cliente solicita la IP y el número de puerto al usuario.
- Crea un socket y se conecta al servidor
- Inicializa flujos de entrada y salida (in, out) para comunicarse con el servidor
- "STATUS: Conectado al servidor" en la consola.

3. Envío

- El usuario ingresa un texto y presiona Enter en el cliente.
- El cliente envía el texto al servidor a través del flujo de salida del socket y espera la recepción del mensaje procesado

4. Procesamiento

- El servidor recibe el texto a través del flujo de entrada del socket.
- Realiza el servicio de eco (invierte y convierte a mayúsculas la cadena) y envía el eco de vuelta al cliente a través del flujo de salida del socket.

Caso de Uso (2)

5. Recepción

- Cuando el cliente recibe la respuesta, muestra el eco del servidor en la consola.

6. Repetición

- El cliente solicita nuevamente al usuario que introduzca texto para enviar al servidor.

7. Finalización

- El cliente envía "END" al servidor para indicar que desea terminar la interacción.
- El servidor recibe "END", envía "OK" al cliente y cierra la conexión con ese cliente.
- El cliente cierra los flujos y el socket, y muestra un mensaje indicando que se ha cerrado.

SERVIDOR Y CLIENTE UDP

Requisitos UDP

Cliente

Debe permitir al usuario introducir una cadena desde el teclado, la cual se envía a través de un socket UDP al servidor

Servidor

Debe devolver esa cadena revertida y en mayúsculas al cliente, que muestra por pantalla.

Interacciones

Esta interacción deberá finalizar cuando el usuario introduzca por teclado la cadena “END”, que no enviará al servidor (será el propio cliente quien termina la ejecución)

El servidor seguirá activo.

Caso de Uso (Cliente)

1. Enviar Mensaje al Servidor:

- El cliente solicita al usuario que introduzca un mensaje.
- El cliente convierte el mensaje a bytes.
- Se crea un DatagramPacket con la información del servidor.
- Se envía el paquete al servidor utilizando el socket del cliente.

2. Respuesta

- El cliente espera recibir un paquete del servidor.
- Cuando se recibe el paquete, se muestra el eco del servidor en la consola.

3. Nueva Entrada

- El cliente solicita nuevamente al usuario que introduzca texto para enviar al servidor.

4. Finalización

- El cliente envía "END-UDP" al servidor.
- Se espera la respuesta "OK" del servidor.
- El cliente cierra su socket.
- Se muestra un mensaje indicando que se ha cerrado
-

Caso de Uso (Servidor)

1. Recibir Mensaje

- El servidor recibe el paquete del cliente.
- Muestra la dirección IP y el puerto del cliente en la consola.
- Revierte y convierte a mayúsculas el texto recibido.

2. Respuesta

- Se crea un nuevo paquete con la respuesta.
- Se envía el paquete de vuelta al cliente.

3. Recibir Fin de Interacción

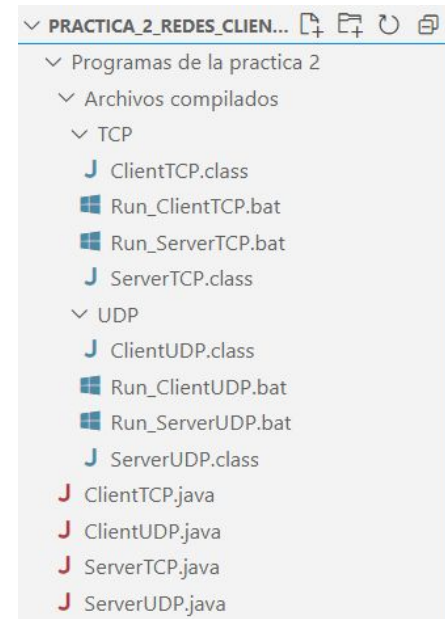
El servidor recibe "END-UDP" del cliente. Envía "OK" al cliente para confirmar la terminación. Cierra la conexión con ese cliente.

4. Repetir Interacciones

- El servidor espera recibir un nuevo mensaje del cliente.
- Se repiten los pasos de recibir mensaje, procesar, enviar respuesta hasta que el cliente envíe "END-UDP".

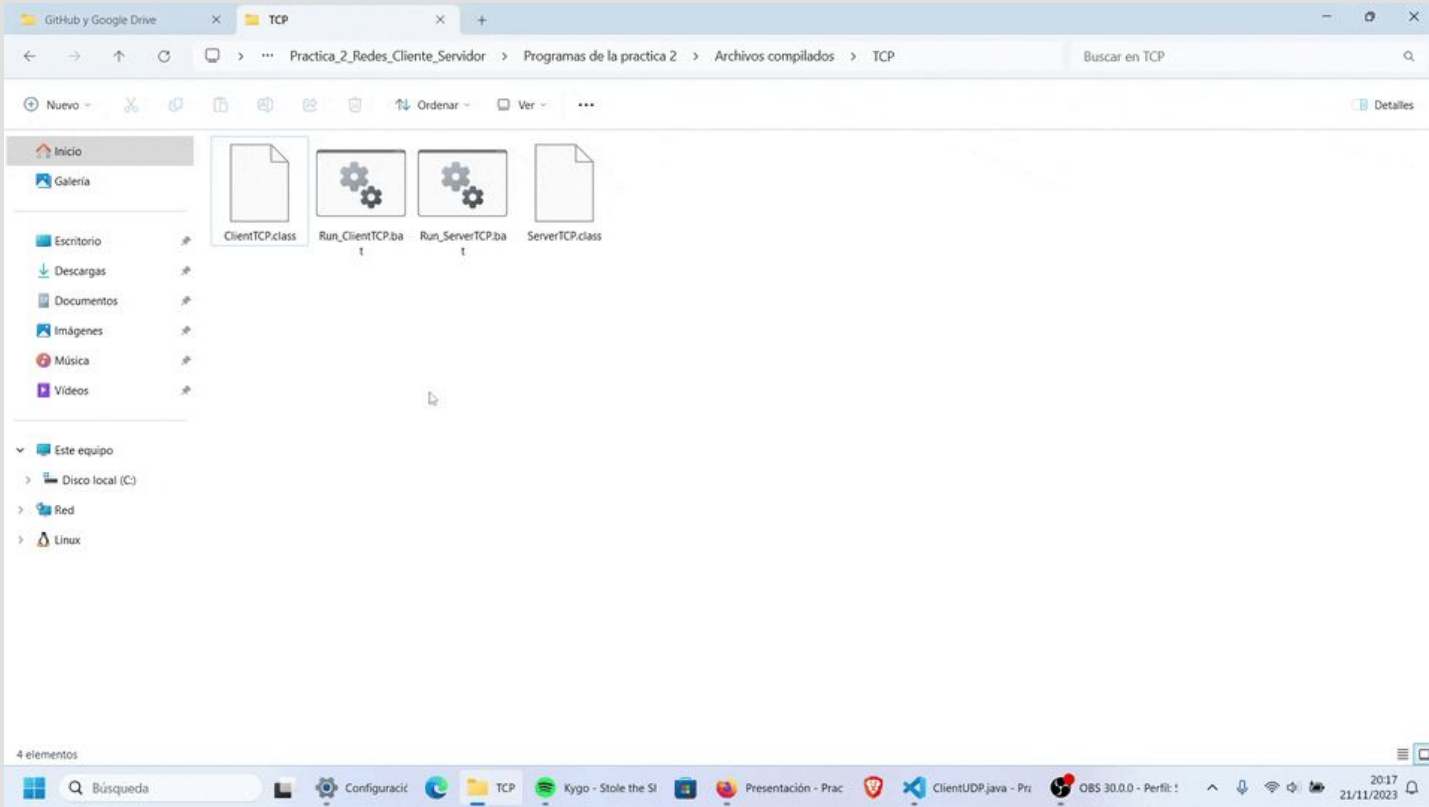
Uso de los programas

Para poder facilitar el uso de pruebas en todos nuestros dispositivos hemos creado unos scripts para windows con los que tener ya el terminal listo con cada una de las clases



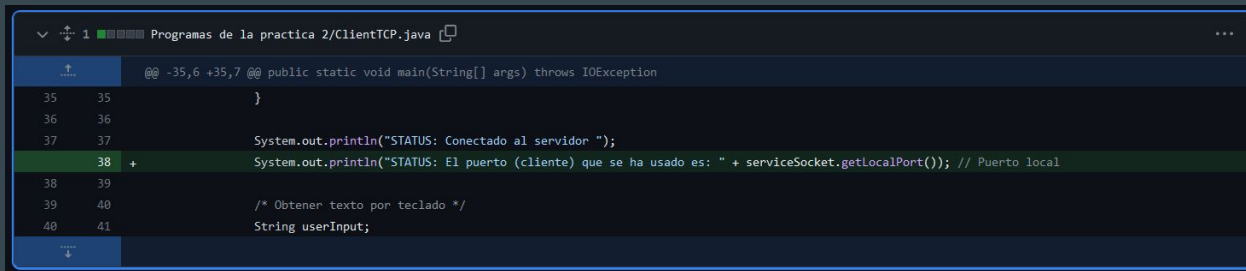
Las primera pruebas las hicimos en eclipse, y una vez que funcionaban, compilamos las clases para ejecutarlas

Así solo tenemos que hacer doble click y tenemos el terminal que deseemos



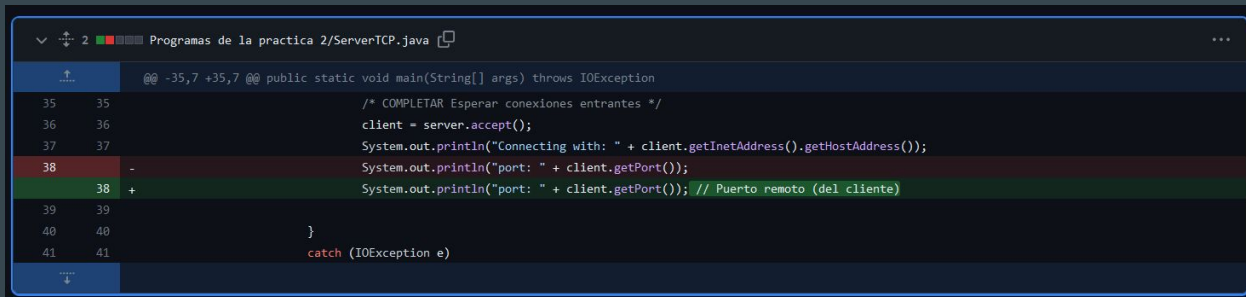
Añadidos extra

Para identificar correctamente los sockets hemos añadido también en el cliente muestre el número de puerto (sobre todo al probar distintas instancias, para que se puedan identificar usando la misma IP)



```
Programas de la practica 2/ClientTCP.java
35 35      }
36 36
37 37      System.out.println("STATUS: Conectado al servidor ");
38 38      System.out.println("STATUS: El puerto (cliente) que se ha usado es: " + serviceSocket.getLocalPort()); // Puerto local
39 39
40 40      /* Obtener texto por teclado */
41 41      String userInput;
```

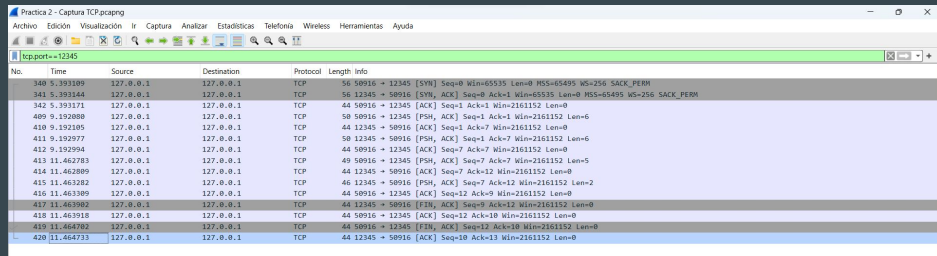
Así podemos identificar el puerto tanto en el extremo del servidor, como del cliente (El del servidor hemos fijado el mismo)



```
Programas de la practica 2/ServerTCP.java
35 35      /* COMPLETAR Esperar conexiones entrantes */
36 36      client = server.accept();
37 37      System.out.println("Connecting with: " + client.getInetAddress().getHostAddress());
38 38      System.out.println("port: " + client.getPort());
39 39
40 40      }
41 41      catch (IOException e)
```

Verificamos lo que hemos aprendido en teoría observando las tramas con Wireshark

TCP



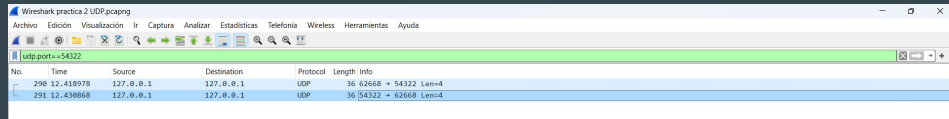
Practica 2 - Captura TCP.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

tcp.port==12345

No.	Time	Source	Destination	Protocol	Length	Info
340	5.393389	127.0.0.1	127.0.0.1	TCP	56	58916 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
341	5.393444	127.0.0.1	127.0.0.1	TCP	56	12345 → 58916 [SYN, ACK] Seq=0 Ack=1 Wlen=65535 Len=0 MSS=65495 WS=256 SACK_PERM
342	5.39371	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=1 Ack=1 Wlen=2161152 Len=0
400	9.192088	127.0.0.1	127.0.0.1	TCP	50	58916 → 12345 [PSH, ACK] Seq=1 Ack=1 Wlen=2161152 Len=6
410	9.192095	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [ACK] Seq=1 Ack=7 Wlen=2161152 Len=0
411	9.192077	127.0.0.1	127.0.0.1	TCP	50	12345 → 58916 [PSH, ACK] Seq=1 Ack=7 Wlen=2161152 Len=6
412	9.192994	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=7 Ack=7 Wlen=2161152 Len=0
413	11.462783	127.0.0.1	127.0.0.1	TCP	40	58916 → 12345 [PSH, ACK] Seq=7 Ack=7 Wlen=2161152 Len=5
414	11.462889	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [ACK] Seq=7 Ack=12 Wlen=2161152 Len=0
415	11.463282	127.0.0.1	127.0.0.1	TCP	46	12345 → 58916 [PSH, ACK] Seq=7 Ack=12 Wlen=2161152 Len=2
416	11.463309	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=12 Ack=9 Wlen=2161152 Len=0
417	11.463982	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [FIN, ACK] Seq=9 Ack=12 Wlen=2161152 Len=0
418	11.463918	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=12 Ack=10 Wlen=2161152 Len=0
419	11.464782	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [FIN, ACK] Seq=12 Ack=10 Wlen=2161152 Len=0
420	11.464773	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [ACK] Seq=10 Ack=13 Wlen=2161152 Len=0

UDP



Wireshark practica 2 UDP.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

udp.port==54322

No.	Time	Source	Destination	Protocol	Length	Info
290	12.418978	127.0.0.1	127.0.0.1	UDP	36	62668 → 54322 Len=4
291	12.430866	127.0.0.1	127.0.0.1	UDP	36	54322 → 62668 Len=4

Comparación

Ofrece confiabilidad mediante la garantía de entrega, retransmisión de paquetes perdidos y control de flujo.

Mayor complejidad debido a la gestión de conexiones, control de flujo y retransmisión

Generalmente más lento debido a la gestión adicional y garantía de entrega

No garantiza la entrega ni el orden de los paquetes

Más simple ya que carece de mecanismos de control de flujo y retransmisión

Más rápido, pero puede resultar en una entrega no confiable

Casos en los que pueden fallar los programas

En el propio enunciado de la práctica salen unos ejercicios, en los que se dan casos especiales:

Servidor:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redес_Cliente_
Servidor\Programas de la practica 2\Archivos compilados\TCP>java ServerTCP
Waiting for a new TCP client
Connecting with: 127.0.0.1
port: 50356
Received from client pruebamuchosclientes
Sending to client SETNEILCSOHCUMABEURP
|
```

Cliente 1:

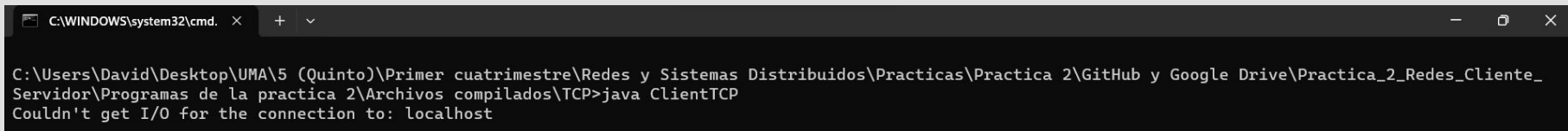
```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Go
Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP
Conexion localhost/127.0.0.1
STATUS: Conectado al servidor
STATUS: El puerto (cliente) que se ha usado es: 50356
Introduzca un texto a enviar (END para acabar)
pruebamuchosclientes
STATUS: Enviando pruebamuchosclientes
STATUS: Esperando eco
echo: SETNEILCSOHCUMABEURP
Introduzca un texto a enviar (END para acabar)
|
```

Cliente 2:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Go
Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP
Conexion localhost/127.0.0.1
STATUS: Conectado al servidor
STATUS: El puerto (cliente) que se ha usado es: 50345
Introduzca un texto a enviar (END para acabar)
segundocliente
STATUS: Enviando segundocliente
STATUS: Esperando eco
|
```

Casos en los que pueden fallar los programas

Al ejecutar dos servidores consecutivos, el segundo no se puede iniciar, ya que solo se puede crear un socket con el mismo puerto en un mismo dispositivo (que es como hemos probado en todos los ejercicios) aunque si se prueba desde dispositivos distintos si se podría, ya que la dirección IP sería distinta, y de esta forma los identificadores del socket serían distintos también:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.' and standard window controls. The command prompt shows the full path to a Java file: 'C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP'. The output of the command is an error message: 'Couldn't get I/O for the connection to: localhost'.

```
C:\WINDOWS\system32\cmd.  X  +  v

C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP
Couldn't get I/O for the connection to: localhost
```