DESARROLLO DE SERVICIOS TELEMÁTICOS **CURSO 22/23**

Prácticas de Laboratorio

BLOQUE Nº 1. Programación con sockets

Nota: El lenguaje de programación recomendado para el desarrollo de las prácticas es Java
Para la entrega de cada bloque, se hará en un zip con el código y además una memoria donde debe aparecer para cada ejercicio los siguientes puntos:

- Descripción de la tarea realizada
- Capturas de pantalla de la ejecución
- Conclusiones

I. Desarrollo de servicios no estándar.

1. Implementar un servicio de "eco" sobre UDP que lea las líneas de teclado hasta leer una línea que contenga solamente un punto. Deberán tenerse en cuenta los siguientes requisitos de implementación:

- El servidor UDP debe ser concurrente. Nota: tener en cuenta que no puedo tener dos procesos UDP asociados al mismo puerto en una misma máquina
- Modelar el final del servicio con el envío de un "." solo en una línea (<CRLF>.<CRLF>).
- El cliente UDP debe gestionar un temporizador de retransmisiones y establecer un número máximo de retransmisiones
- Tanto el cliente como el servidor deben controlar que los mensajes del servicio de eco pertenezcan todos a la misma transacción chequeando el puerto

2. Implementar el mismo servicio de "eco" que en el ejercicio anterior pero esta vez sobre TCP. Deberán tenerse en cuenta los siguientes requisitos de implementación:

- Usar la interfaz *Executor* para implementar el servidor concurrente
- Opcionalmente implementar la otra opción de grupo de hebras y comparar ambas implementaciones
- El cliente le notificará al servidor el final del servicio mediante el cierre del *socket*.

3. A partir del servicio de "eco" dado en clase realizar las siguientes modificaciones para probar las diferentes opciones de sockets sobre TCP:

- En el servidor, limitar el tiempo de espera de una nueva conexión dando por finalizado el servidor tras dicho tiempo (ej: varios minutos). Nota: probar a arrancar el servidor justo después de su finalización, antes de implementar la característica que se propone a continuación, ¿qué ocurre?
- En el servidor establecer la opción de reutilización del puerto para facilitar el arranque del servidor tras su finalización por el temporizador
- En el servidor, limitar el tiempo total del servicio de "eco" (ej: un minuto)
- Cambiar en el cliente el tamaño del buffer de escritura a 512 bytes y comprobar si se ha cambiado visualizando el valor anterior y el nuevo.
- Deshabilitar el algoritmo de *Nagle*, que es especialmente recomendado si se va a realizar el eco a nivel de caracteres o mensajes muy pequeños.

II. Desarrollo de servicios estándar.

1. Protocolo de transferencia de ficheros trivial (TFTP). Implementar un proceso servidor y un proceso cliente que realicen la transferencia de ficheros sobre UDP de acuerdo al protocolo estándar TFTP definido en el RFC 1350. Tener en cuenta los siguientes puntos.

- Usar otro puerto diferente al estándar 69 de TFTP para el servidor.
- La transferencia podrá hacerse en cualquiera de los dos sentidos (get o put). La fase de transferencia de datos y su finalización es igual para ambos procesos cliente y servidor, variando sólo el establecimiento de la comunicación.
- El fichero que se transfiera debe ser textual ya que se mostrará por pantalla (opcionalmente pueden almacenarse los datos transferidos en un fichero).
- El proceso cliente tendrá un pequeño interfaz de usuario para pedir el tipo de la transferencia y el nombre del fichero a transferir. Emular los comandos del cliente tftp estándar (como mínimo: *connect, get, put y quit*).
- Tanto el cliente como el servidor comprobarán que todos los mensajes que le llegan han sido enviados por el proceso correcto (comprobando el identificador de transferencia que será la IP y el puerto), sino contestará con un mensaje de error. El servidor podrá ser concurrente o iterativo.
- Ambos procesos implementarán los mecanismos para garantizar que la transferencia se haga de forma fiable. Para ello se comprobará que los números de secuencia de los paquetes sean consecutivos y ante la pérdida de un paquete se retransmitirá después de esperar 1 segundo. Poner un número máximo de reintentos (ej: 3 o 5).
- Cada proceso tendrá dos modos funcionamiento: el correcto y con pérdida de paquetes. El modo de funcionamiento con pérdida de paquetes consiste en que durante el inicio y la transferencia de un fichero se podrán perder paquetes (acks o datos) hasta un 5%. La pérdida de paquetes se hará simulada, cada vez que un proceso cliente o servidor tenga que enviar un mensaje, de forma aleatoria decidirá si lo envía o no. Al finalizar se dará un resumen estadístico con el numero/porcentaje de paquetes perdidos y retransmitidos.
- Por simplicidad no se considerará que el primer paquete enviado (READ o WRITE) pueda perderse.
- El proceso cliente (y/o el servidor) dará la traza de la transferencia con el formato que se sigue en el ejemplo:

```
----> RRQ “fichero.dat” “octet”
                                                    <---- ACK 0
----> DATA 1 512 bytes
                                                    <---- ACK 1
----> DATA 2 10 bytes
                                                    <---- ACK 2 (P)
----> DATA 2 10 bytes (R)
                                                    <---- ACK 2

% P: paquetes perdidos; % R: paquetes retransmitidos
```

2. Agregar al ejercicio 4, la renovación de configuraciones antes de que expire el tiempo de arriendo, o “*lease time*” en el cliente. Sacar como parte de la traza de pantalla los estados por

los que va pasando el cliente y los temporizadores que se van inicializando y expirando. Considerar la mayoría de los escenarios contemplados en el RFC.

3. Desarrollar un cliente de `smtp` simplificado que sea capaz únicamente de enviar mensajes de correo electrónico. El protocolo SMTP está definido en el RFC 821 que se adjunta.

- Esta aplicación debe probarse en la máquina virtual de Linux, con el servidor de correo estándar instalado en dicha máquina.
- De los comandos del protocolo propuestos en el RFC sólo tendrán que implementar: HELO, MAIL, RCPT, DATA, QUIT (implementación mínima, pág. 41).
- La interfaz de usuario será muy sencilla
    - Compose
        - Introducir una dirección de correo destino (opcionalmente podrían ser varias)
        - Introducir el asunto del mensaje (el *Subject*)
        - Introducir el cuerpo del mensaje (podrá ser una o varias líneas, hasta finalizar con un punto)
    - Quit
        - Cerrar la conexión
- Para comprobar que los mensajes enviados con tu aplicación han llegado correctamente utiliza el cliente de correo estándar de Linux Mail. Un ejemplo sería:

```
[alumno@localhost ~]$ Mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/alumno": 1 message 1 new
>N  1 alumno@localhost.loc  Thu Oct 17 13:57  22/759   "Prueba"
& 1
Message 1:
From alumno@localhost.localdomain  Thu Oct 17 13:57:30 2013
Date: Thu, 17 Oct 2013 13:57:29 +0200
From: alumno@localhost.localdomain
To: alumno@localhost.localdomain
Subject: Prueba

Estimados alumnos:
espero que la práctica no les resulte difícil.

Saludos,
Lidia
&
```

- Nota: No usar la API `javax.mail`
- Nota: La dirección de correo de este usuario es `nombre_usuario@nombre_dominio`. Para obtener el nombre del usuario actual, usar los siguientes métodos de Java:
    - `nombre_usuario: System.getProperty("user.name")`
    - `nombre_dominio: InetAddress.getLocalHost().getHostName();`

III. Patrones de diseño.

1. Implementar un multi-servidor que proporcione al menos dos servicios de eco sencillos, ambos sobre TCP. Nota: pueden ser dos versiones diferentes del servicio de "eco".

2. Implementar un servidor multi-protocolo que proporcione un servicio de "eco" a nivel de línea, pero que acepte varias líneas de entrada (similar al servicio definido para los ejercicios I.1 y I.2). Se implementarán dos versiones del mismo servicio, uno sobre TCP y otro sobre

UDP, que utilizarán el mismo número de puerto. El servidor multi-protocolo se implementará usando el paquete NIO (Selector), y redirigirá la petición de servicio al servidor de eco sobre TCP o sobre UDP, según corresponda. Se podrá probar la solución con los mismos clientes que los implementados para los ejercicios I.1 y I.2. Se valorará el grado de modularización de la solución presentada.

3. Implementar un servidor multi-cliente de chat, utilizando los canales TCP del paquete NIO y el *Selector*. El servidor leerá mensajes de entrada de cada cliente y los difundirá entre todos los canales almacenados en el selector. Dispondrán de una implementación de un cliente de chat con interfaz gráfica que pueden usar para probar la aplicación ó opcionalmente pueden proporcionar una propia. Se valorará el grado de modularización de la solución presentada. Nota: Tener cuidado con el tamaño del buffer, procurar dimensionar el buffer del tamaño exacto de los datos a enviar.

Obligatorios: II.1 y III.2
Optativo clase: el resto de ejercicios

Fecha entrega: 31/10/22