Práctica 5

Desarrollo de una aplicación distribuida de chat utilizando UDP.

Información básica

El objetivo es el desarrollo de una aplicación distribuida utilizando sockets sin conexión UDP. La aplicación consistirá en un chat que permita la comunicación entre usuarios.

Tarea 1: Desarrollo de la aplicación de chat usando UDP

Para el desarrollo de la práctica se facilita el código correspondiente al interfaz gráfica y el tratamiento de todos los elementos necesarios que no están relacionados con las comunicaciones. Se ha usado el patrón Modelo-Vista-Controlador y se proporciona el código para la Vista y el Controlador, dejando el Modelo (módulo de comunicaciones) para el alumno. Dicho código se encuentra disponible en el campus virtual de la asignatura (la clase ejecutable es **`ControladorImpl`**).

El objetivo es que el alumno desarrolle una clase (**`ComunicacionImpl`**) que implemente la siguiente interfaz para encargarse de las comunicaciones:

```
public interface Comunicacion
{
	public void crearSocket(PuertoAlias puerto);
	public void setControlador (Controlador c);
	public void runReceptor();
	public void envia(InetSocketAddress sa, String mensaje);
	public void joinGroup(InetAddress multi);
	public void leaveGroup(InetAddress multi);
}
```

La descripción de los métodos es la siguiente:

`public void crearSocket(PuertoAlias puerto);` Se encarga de la creación del socket UDP por el que se van a realizar las comunicaciones. Como argumento recibe una clase que contiene el puerto de escucha del socket local y el alias del usuario. Este alias será necesario para el intercambio de mensajes, como se explica más abajo. Este método será el segundo en ejecutarse, después de `setControlador`.

`public void setControlador (Controlador c);` Recibe el objeto controlador que utilizaremos para mostrar en el interfaz gráfica los mensajes que recibamos. Abajo se muestran los métodos relevantes de la interfaz `Controlador`. Este método es el primero en invocarse, ya que se llama cuando se están configurando los distintos componentes de la aplicación al inicio de su ejecución. Se recomienda que el alumno simplemente guarde el objeto controlador como atributo de instancia de la clase de comunicación.

`public void runReceptor();` Este método se encarga de recibir los segmentos de datos y mostrarlos en el interfaz gráfica, utilizando el objeto controlador recibido en el método anterior.

`public void envia(InetSocketAddress sa, String mensaje);` Se encarga del envío de mensajes al otro proceso de chat, cuya dirección de socket se indica como primer argumento.

`public void joinGroup (InetAddress multi);` Este método se utiliza para unirse a un grupo multicast cuya dirección se indica. De esta forma se podrán simular las salas de chat en las que más de dos personas pueden participar en una conversación.

`public void leaveGroup (InetAddress multi);` Este método se utiliza para dejar el grupo multicast cuya dirección se indica.

No es necesario estudiar todo el código que se proporciona. Los **únicos elementos** del código facilitado que el alumno necesita conocer son la clase `PuertoAlias`, que se encuentra definida dentro de la clase `DialogoPuerto` y el método `mostrarMensaje` de la interfaz `Controlador`. A continuación se muestran ambas y se explica su papel en la aplicación.

```
public static class PuertoAlias
{
      public int puerto;
      public String alias;
}
```

Esta clase se utiliza para guardar el alias del usuario y el puerto UDP en el que debe escuchar la aplicación. Ambas se piden al usuario al comienzo de la ejecución. Un objeto de esta clase se le pasará al objeto de comunicación a través del método `crearSocket`, el cual será invocado antes de cualquier otro relacionado con las comunicaciones. Este método debe crear el socket que se utilizará para las conexiones, así como guardar el alias del usuario para un uso posterior.

```
public void mostrarMensaje(SocketAddress sa, String nick, String msj);
```

Por su parte, el método `mostrarMensaje` de la clase `Controlador`, debe ser utilizado en el método `runReceptor` de la clase de comunicación para mostrar el mensaje en la interfaz gráfica. El chat permite comunicaciones unicast y multicast. En el caso de que el mensaje recibido haya sido un mensaje multicast (enviado a grupo de máquinas), la dirección de socket estará formada por la dirección IP de grupo y el puerto en el que la aplicación está escuchando. La dirección multicast será necesario recuperarla de los propios datos del segmento UDP (abajo se explica el formato que tendrán los segmentos). En el caso multicast, tenga en cuenta en no mostrar duplicados los mensajes enviados por su máquina (la información sobre sus interfaces pueden consultarse en la clase `NetworkInterface`). El puerto se puede consultar mediante los métodos disponibles en la clase del socket. Si el mensaje es unicast, la dirección de socket será la del origen del mensaje (el otro proceso). Esta dirección se obtiene fácilmente llamando algún método de la clase `DatagramPacket`.

El contenido del segmento UDP es una cadena de texto con codificación UTF-8. La cadena está formada por tres campos separados por el carácter > (signo de mayor). El primer campo es la dirección IP destino del datagrama. A continuación se encuentra el alias del usuario. El resto de la cadena contiene el mensaje que el usuario escribió. La cadena de texto **no tiene** terminador. El objetivo de incluir el alias en el segmento es que el usuario pueda ver el nombre delante del mensaje, como suele ocurrir en las aplicaciones de chat o mensajería instantánea. Un par de ejemplos de mensajes son:

```
10.0.0.1>francis>Hola! ¿Cómo estás?
224.0.0.3>Gabriel>Bien. Liado con la práctica >.>
```

Nota importante: como los equipos tienen múltiples interfaces, si no indicamos por cual enviar, UDP delegará la responsabilidad de elegir el interfaz a la capa de red (IP). Esto puede provocar que los envíos no se realicen de forma apropiada. Para evitar esta situación: en la creación del socket (ya sea `DatagramSocket` como `MulticastSocket`) y en la unión/abandono de grupo multicast, elija el constructor o método en el cuál le podamos indicar la dirección y/o interfaz que queremos utilizar. En todos los casos queremos utilizar la interfaz física (real) del equipo.

Ayuda: para crear un array de bytes que represente a la cadena en UTF-8 puede usar el método `getBytes(Charset)` de la clase `String`, para crear la cadena a partir de la secuencia de bytes puede usar el constructor `String(byte [], Charset)`.

Una vez completada la clase de comunicaciones realice los siguientes ejercicios:

Ejercicio 1. *Prueba local.* Pruebe a lanzar dos instancias del chat en puertos distintos y pruebe el correcto funcionamiento de la misma (como dirección indique su dirección IP externa: 192.168.X.Y). Haga una captura de pantalla donde se vea la conversación en las dos instancias de la aplicación.

Ejercicio 2. *Prueba remota.* Una vez comprobado el buen funcionamiento de su código, póngase de acuerdo con uno o varios compañeros y prueben el funcionamiento de la aplicación en un entorno realmente distribuido. Hagan capturas de pantalla donde se vean las conversaciones.

Ejercicio 3. *Análisis de las comunicaciones UDP*. Utilizando Wireshark (**fichero p5e3.pcapng**) analice las comunicaciones que se producen y haga alguna captura donde se observe la cabecera de algún datagrama UDP. Para la prueba envíe mensajes de diferente longitud y con caracteres especiales (tildes por ejemplo). Observe el campo longitud en la cabecera UDP, ¿se corresponde ese valor con los datos enviados? Adjunte una captura de pantalla del datagrama UDP.

Ejercicio 4. *Multicast.* Póngase de acuerdo con tres o más compañeros para ejecutar el chat en el mismo puerto e iniciar una conversación usando la dirección de grupo (multicast) **239.194.17.132**. Haga una captura de pantalla donde se vea claramente la ventana de conversación en cada máquina.

Nota sobre la memoria

- Si elabora la memoria en Word, se aconseja utilizar la plantilla proporcionada para la práctica. En cualquier caso, la memoria debe contener toda la información que se pide en la plantilla y seguir su estructura.
- La memoria de esta práctica se entregará en conjunto a la memoria de la 4.
- Cuando la práctica consista en el desarrollo de un código, **en la memoria se explicará el esquema del mismo**, únicamente detallando (y explicando) las partes más significativas del mismo.
- Dicha memoria debe constar de una portada donde se indique el conjunto de prácticas que incluye la memoria, así como todos los datos del alumno.
- La memoria de cada práctica debe empezar en una nueva página.
- No es necesario copiar el enunciado completo de la práctica pero sí debe copiarse el enunciado de cada ejercicio antes de indicar su respuesta. Debe utilizarse algún sistema de estilos que permita distinguir lo que es el enunciado de lo que es la respuesta al ejercicio.
- Para cada ejercicio que obtenga la información de algún proceso realizado en el ordenador (traza de wireshark, comando...) realice una captura (con <alt>+<impr pant> sólo capturaremos la ventana activa actual). Además de incluirla captura se deben utilizar las herramientas de dibujo del procesador de texto usado para marcar la parte donde se observa lo que pide el ejercicio. Finalmente en el texto añada una pequeña descripción de la captura.
- El formato de entrega de las prácticas será PDF. Además del fichero de la memoria, deberá entregarse el código desarrollado (los `ComunicacionImpl.java`) y la traza de wireshark (`p5e3.pcapng`).

Ficheros en la entrega del Bloque II):

En caso de estar realizada en pareja, **solo debe** subirlo **uno** de los dos. Aseguraros que en la memoria aparece el nombre de ambos.

En la tarea del campus virtual debe subir un fichero comprimido (en **zip**) con los siguientes ficheros (**no** cree subdirectorios):

- **Memoria.pdf:** Documento con la solución a los ejercicios de la práctica 4 y 5 de forma conjunto. Incluya una portada inicial. Los códigos debe explicarlos, ya se añadiendo una explicación en la memoria o añadiendo comentarios en el código.
- **Ficheros de las trazas de Wireshark:**
  - Práctica 4: **p4e1-7.pcapng – p4e8.pcapng – p4e9-10.pcapng – p4e11-12.pcapng**
  - Práctica 5: **p5e3.pcapng**
- **Código fuente: NO** se entrega el proyecto eclipse completo, solo los código **.java**:
  - Práctica 4: **Cliente.java – Servidor.java**
  - Práctica 5: **ComunicacionImpl.java**
- **Otros:** ficheros no solicitados pero que considere importantes (imágenes, ficheros explicativos, documentación externa consultada, …)