INFORME AEC1 SISTEMAS DISTRIBUIDOS

Antonio Luis Ojeda Soto

1.Explicación del proceso de la práctica.

Al principio comencé la práctica implementando las ventanas con swing, cosa que no debí hacer, ya que empezó a ser bastante lioso, con muchas clases, así que reinicié la práctica, siguiendo tus recomendaciones, comenzando a coger como plantilla uno de los códigos puestos en la asignatura, en este caso los códigos de Streams del Tema 3. Primero pensé que debía hacer la práctica de forma **básica** con cliente-servidor intercambiando mensajes de forma síncrona. Para ello elegí hacerlo con **Sockets de tipo Stream**, orientado a conexión con protocolo TCP en capa de transporte y sin límites de tamaño de mensajes (esto da problemas en Datagramas ya que si no se pone el tamaño adecuado de datos en los buffers, uno por cada proceso, puede haber bloqueos), resultando a mi entender más fácil que el Socket de tipo Datagrama, ya que una vez aceptada la conexión no hay que poner destinatario cada vez que se quieran comunicar los dos procesos, mientras que en el de Datagramas sí.

El socket de tipo Stream también tiene la ventaja de que se comunican con un único buffer y un socket para cada proceso y éste es otro de los aspectos por el que me decanté a favor de los Streams. El único problema es que la conversión de tipos es un poco más intrincada que con Datagramas, pero con un par de líneas se código se soluciona.

Una vez que veo que funciona quiero avanzar un poco más e intento hacer el chat en modo concurrente. En un principio no sabía qué poner en los hilos, si las entradas o las salidas o ambos, veo que en el código del Tema 4, se deja en la parte del cliente, las salidas en el main y las entradas en el hilo, hago lo mismo en el servidor y después de varias pruebas funciona correctamente.

En la sesión de dudas te pregunto cómo se podría hacer para conectar a varios clientes creando el chat entre más de dos personas. Me resuelves la duda diciéndome "creando unas instancias de clientes" y así lo hago. Finalmente funciona correctamente.

2.Documentación del código implementado.

Para el código AEC_1_BASICO: Lo primero que hice fue probar el código ofrecido de tipo Stream y ver cómo funcionaba. En un primer momento me di cuenta de que enviaba un mensaje el cliente y automáticamente el servidor se lo devolvía junto con la hora de entrega, y luego el cliente se desconectaba. Entonces pensé en poner otro bucle "while" en **lado Cliente** y sacar el cierre del socket fuera para que hubiese comunicación constante. En este bucle "while" los mensajes serían enviados y recibidos y no se sale del mismo hasta que reciba un mensaje "finalizar" por parte del servidor para después cerrar el socket. En el **lado Servidor** lo que hice simplemente fue borrar la marca_temporal y que la respuesta fuese metida por teclado, todo en el bucle "while" de aquí ya estaba hecho.

Para el código AEC_1_CONCURRENTE: cojo los códigos de la asignatura del tema 4 y trabajo con ellos. Para ello pongo en la clase main de cada proceso, todo lo anterior, excepto que aquí en el bucle "while" solo dejo los mensajes salientes, dejando en los hilos de ambos procesos las peticiones que llegan del exterior, o sea, los mensajes entrantes. Lo pruebo unas cuantas veces y funciona de forma concurrente, mandando mensajes no bloqueantes (asíncronos) sin ningún tipo de problema ni orden por parte de cada proceso.

Para el código AEC_1_CHAT: aquí elijo hacer un salón chat, donde los mensajes enviados por un cliente son procesados y enviados al resto. Para ello sigo las

indicaciones de los códigos anteriores: utilizo Sockets de tipo Stream, paradigma cliente-servidor y servidor concurrente multihilo. Para hacer esto último lo primero que hago es crear una estructura de datos donde vamos a ir metiendo los sockets de los clientes que se vayan conectando, elijo **ArrayList**. Después creo la clase "**OtroCliente**" para más tarde crear instancias de clientes. Lo siguiente es poner en el método main lo mismo que hasta ahora, y el bucle "**while**" lo pongo a partir de las aceptaciones de conexión para los clientes, introduciendo en el **ArrayList** los sockets de los clientes que se vayan conectando, luego creando la instancia de cada cliente y por último el hilo de cada uno. En la clase "**HiloServido**r", a diferencia de lo anterior, creamos una variable de cliente que enlazamos con el socket del mismo y poder trabajar con él, lo que nos viene de un cliente lo recogemos y se lo enviaremos a todos a través del método "**enviarATodos ()**" el cual recogerá como parámetro el mensaje del cliente que quiere comunicar y los distribuirá al resto. ¿Cómo? Se van mirando los sockets que estén conectados recorriendo el **ArrayList** (con un bucle **for** del libro) que creamos anteriormente con los sockets de los clientes que se van añadiendo y se les va enviando a todos los que estén en esta estructura de datos. Se prueba y el código funciona correctamente.