Documentación linkChat

En esta actividad se ha creado un chat llamado ”LinkChat” en el que varias personas pueden comunicarse entre sí. Es un chat de mensajería uno a uno en el que se utiliza el paradigma de cliente servidor. La aplicación se conecta al servidor ”LinkServer” para poder intercambiar mensajes entre diferentes usuarios.

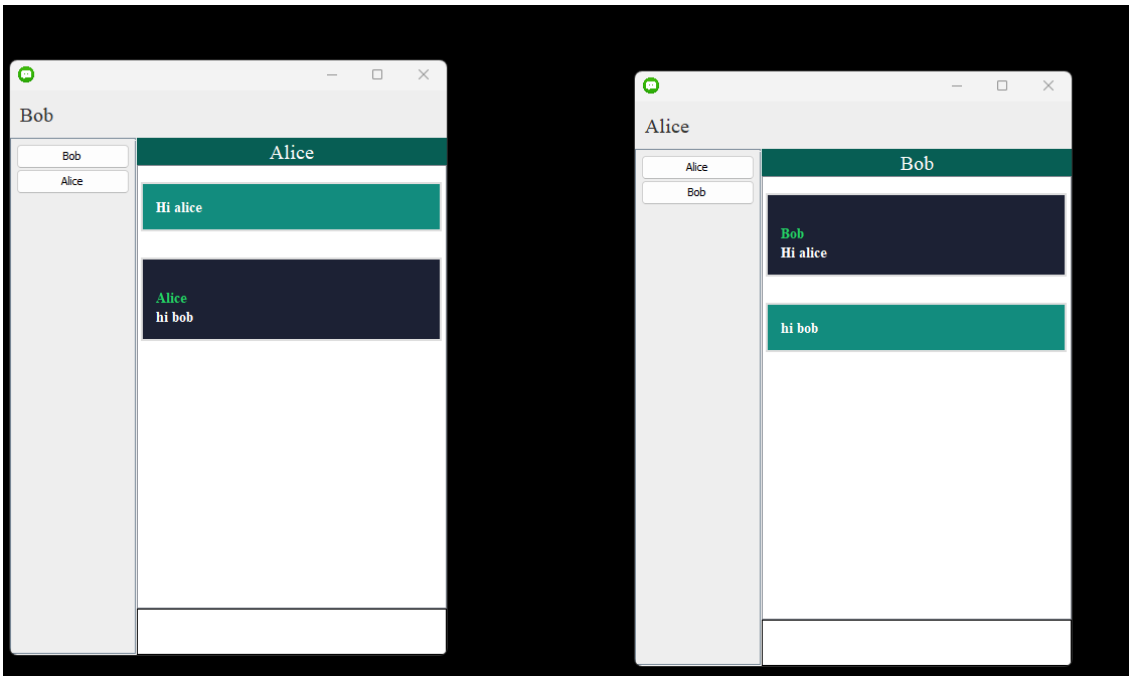

Figura 1. conversación en linkChat.

Tipo de socket y protocolo de transporte

Se ha elegido utilizar TCP/IP como protocolo de transmisión. Para esto se han utilizado los sockets de tipo stream que permiten a un proceso establecer una comunicación orientada a conexión.

La utilización de este protocolo frente a UDP, es la garantía de que no se pierde información, que esta llega en el orden adecuado y que proviene del destinatario adecuado. Sin esta garantía el chat necesitaria código extra para manejar los posibles errores de transmisión.

Protocolo de aplicación

A parte de esto se ha diseñado un protocolo de aplicación con el que el cliente se comunica con el servidor. Este protocolo consiste en varios tipos de mensajes. El tipo de mensaje esta codificado en el propio datagrama de transmisión. Algunos tipos de mensaje son exclusivos al servidor o cliente.

Los tipos de mensajes que existen son:

- ERROR
- MESSAGE
- REGISTRATION
- USER_CONNECTED
- USER_DISCONNECTED

- INVALID_USER_NAME

Mensaje tipo ERROR

Este mensaje es enviado por el servidor a los clientes cuando un error ha ocurrido. El cliente no envía mensajes de error ya que no es necesario. Cualquier error ocurrido con un cliente es detectado con el cierre del socket, el servidor procede a notificar los otros usuarios de que ese usuario ya no esta en linea.

El datagrama de ERROR tiene la siguiente estructura:

0	4	5
4 bytes message length	1 byte message type	Error message

4 bytes indican el tamaño del mensaje en bytes, un byte reservado para indicar el tipo de mensaje y el resto es el contenido del mensaje con la longitud indicada previamente.

Mensaje tipo REGISTRATION

Este mensaje es enviado exclusivamente por los clientes al servidor cuando se establece una conexión con el servidor. El cliente envía el nombre de usuario que es analizado por el servidor. Si no existe ningún usuario con ese nombre el servidor acepta el registro.

El datagrama de REGISTRATION tiene la siguiente estructura:

0	4	5
4 bytes message length	1 byte message type	user name

4 bytes que indican el tamaño del mensaje en bytes, un byte reservado para indicar el tipo de mensaje y el resto es el contenido del mensaje con la longitud indicada previamente.

Mensaje tipo USER_CONNECTED

Este mensaje es enviado exclusivamente por el servidor a los clientes cuando un nuevo usuario se ha conectado. Una vez aceptado el registro el servidor envía este mensaje con una lista de los usuarios que están conectados en ese momento. Los otros clientes que ya están utilizando el chat, reciben únicamente el nombre del nuevo usuario conectado.

El datagrama mensaje error tiene la siguiente estructura:

0	4	5
4 bytes message length	1 byte message type	user name list

4 bytes que indican el tamaño del mensaje en bytes, un byte reservado para indicar el tipo de mensaje y el resto es el contenido del mensaje con la longitud indicada previamente.

Mensaje tipo USER_DISCONNECTED

Este mensaje es enviado exclusivamente por el servidor a los clientes cuando un usuario se ha desconectado.

El datagrama USER_DISCONNECTED tiene la siguiente estructura:

0	4	5
4 bytes message length	1 byte message type	user disconnected name

4 bytes que indican el tamaño del mensaje en bytes, un byte reservado para indicar el tipo de mensaje y el resto es el contenido del mensaje con la longitud indicada previamente.

Mensaje tipo INVALID_USER_NAME

Este mensaje es enviado exclusivamente por el servidor a los clientes cuando un usuario se ha desconectado.
El datagrama INVALID_USER_NAME tiene la siguiente estructura:

0	4	5
4 bytes message length	1 byte message type	user name

4 bytes que indican el tamaño del mensaje en bytes, un byte reservado para indicar el tipo de mensaje y el resto es el contenido del mensaje con la longitud indicada previamente.

Mensaje tipo MESSAGE

Este mensaje es enviado a los clientes al servidor. Una vez allí el servidor examina el mensaje y lo redirecciona al usuario adecuado. El servidor envía estos mensajes pero solo como mediador no crea o escribe mensajes.
El datagrama mensaje error tiene la siguiente estructura:

0	4	5	9	13
4 bytes message length	1 byte message type	4 bytes destiny name size	4 bytes size of the sender name	from + to + message

4 bytes que indican el tamaño del mensaje en bytes, un byte reservado para indicar el tipo de mensaje, 4 bytes para indicar el tamaño en bytes del nombre del usuario de destino, 4 bytes que indican el tamaño en bytes del usuario que envía el mensaje. El resto del mensaje esta compuesto del nombre del destinatario, el nombre del remitente y por último el contenido del mensaje.

Todos los mensajes tienen que pasar por el servidor, una vez en el servidor, este inspecciona el mensaje para averiguar qué tipo de mensaje es y a quién va dirigido. Posteriormente, el servidor envía el mensaje al correspondiente destinatario.

Secuencia de establecimiento de conexión

La conexión entre un proceso y el servidor se muestra a continuación:

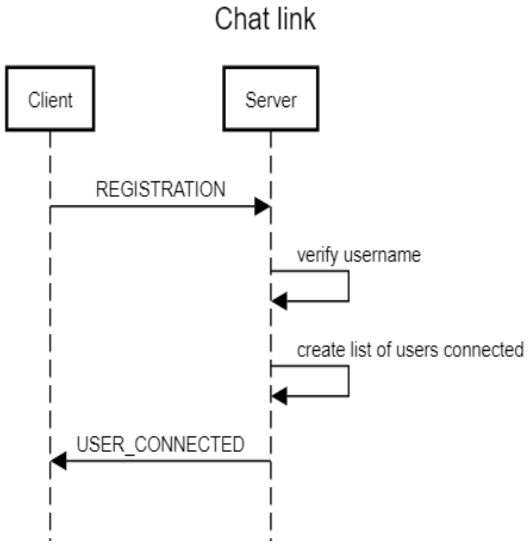

Figura 2. Conexión exitosa

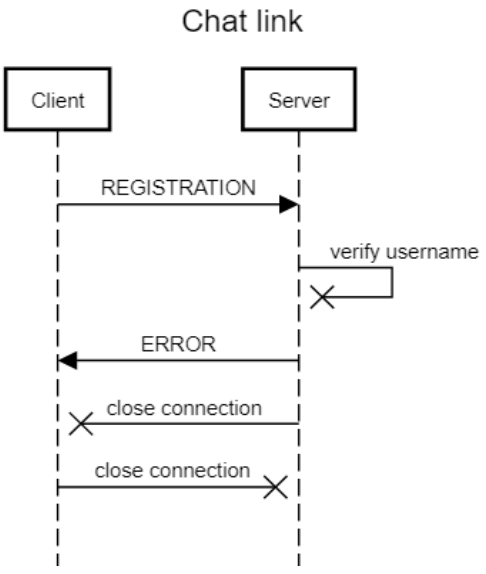

Figura 3. Conexión fallida

1. El cliente envía un mensaje tipo REGISTRATION con el nombre de usuario.
2. El servidor comprueba que ese nombre de usuario no existe. Si existe cierra la conexión y avisa al cliente.
3. El servidor envía al cliente todos los usuarios en línea y notifica a todos los otros clientes que un nuevo usuario se ha unido.
4. El cliente envía mensajes al servidor hasta que cierre la conexión.

Una vez que la comunicación con el servidor se ha establecido, el envío de mensajes entre servidor y cliente se produce de forma asíncrona. El cliente envía mensajes al servidor en cualquier momento y el servidor envía los mensajes al cliente cuando existe alguno que va dirigido hacia él sin necesidad de coordinación.

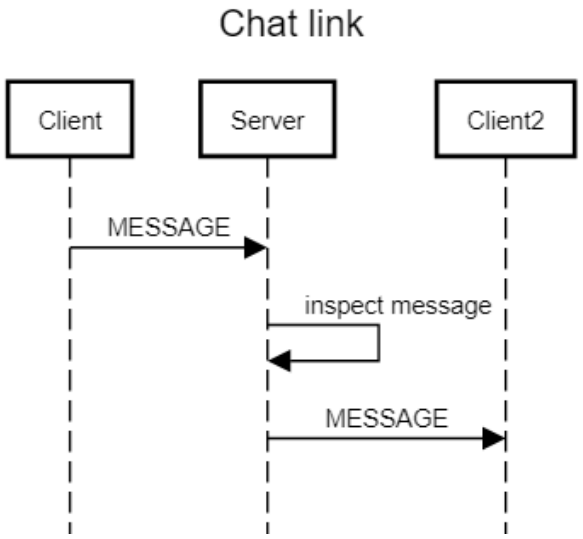

Figura 4. Envío de mensajes entre usuarios.

Si un usuario se desconecta, el servidor cierra la conexión y notifica a los demás usuarios que cliente se ha desconectado como se puede observar en el diagrama:

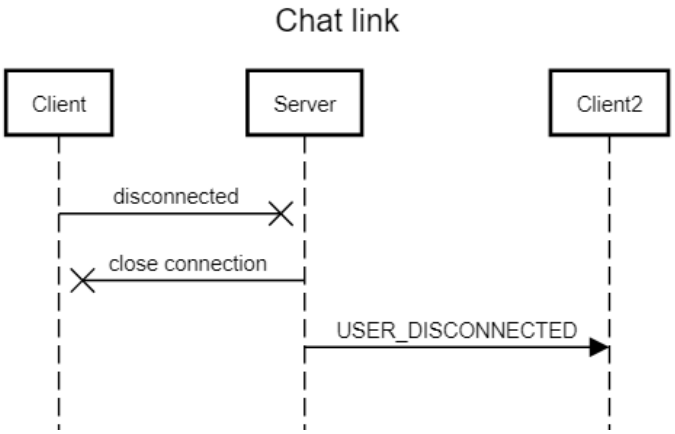

Figura 5. Cliente/usuario se desconecta.

Procesos involucrados

El numero de procesos que intervienen es como mínimo dos: el servidor y el cliente. Sin embargo, pueden estar presentes múltiples procesos. El número máximo de procesos soportado no se conoce, ya que puede variar según el hardware en el que se ejecuta. Este sin embargo ha sido probado con hasta 50 procesos simultáneos sin incurrir en ningún tipo de problema.

Arquitectura de software

El servidor utiliza sockets con bloqueo. Esto significa que el socket se bloquea hasta que recibe algún mensaje. Existen técnicas para crear sockets sin bloqueo. Sin embargo, en esta implementación se ha optado por utilizar varios threads para lidiar con el bloqueo de los sockets.

Al iniciar el servidor lo primero que se crea es un thread de lectura que lee una lista de tipo FIFO de mensajes. Esta lista es accedida por todas las conexiones. Cuando un mensaje llega por una conexión, este es añadido a la lista y posteriormente. Cuando un cliente se conecta el servidor crea un nuevo thread el cual se encarga de verificar, establecer la conexión y registrar el usuario. Una vez se registra con éxito, se lanza un nuevo thread que se dedica exclusivamente a leer mensajes (uno por cada cliente) provenientes de ese socket y añadirlos a la lista. Si no existen mensajes el thread de lectura se duerme hasta que un nuevo mensaje es añadido a la lista. El read thread lee todos los mensajes de la lista y una vez vacía se vuelve a dormir.

En el siguiente diagrama se puede observar las principales clases que participan en el funcionamiento de la aplicación. Por simplicidad solo se muestran las principales clases.

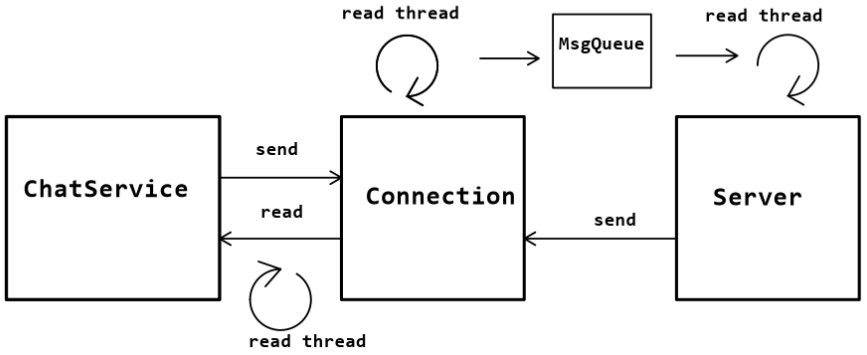

Figura 6. diagrama de interacción entre las clases principales

Las principales clases que participan en el manejo del envío de mensajes y conexión se encuentran en el paquete ***link.net.**** con el objetivo de reutilizar el mismo código tanto en cliente como en servidor. ***ChatService*** es una abstracción utilizada para poder implementar un cliente chat de cualquier tipo. Esto permite crear nuevas Interfaces o chats en la línea de comandos sin tener que cambiar el código o adaptarlo.

Instalación y compilación del código fuente

El chat ha sido desarrollado usando windows por lo tanto para instalar el cliente se puede utilizar el installer msi proporcionado. Para ejecutar el programa en linux el usuario deberá tener instalado java 11 o versión compatible y compilar el código directamente.

Instalación del cliente en windows 11

Para Windows podemos utilizar el installer proporcionado, crear un installer ejecutando build.bat o importar el proyecto y compilarlo desde un IDE.

A continuación se muestran los pasos a seguir utilizando el installer:

1. Haz click en el instalador **linkChat-1.0.msi**.
2. Elige el destino en el que deseas instalar el chat.
3. Ejecuta el programa desde el icono mostrado en el escritorio.

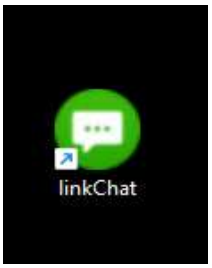

Figura 7. Icono del programa link chat tras la instalación

Instalación y ejecución del servidor en windows:

Al igual que el cliente, para Windows podemos utilizar el installer proporcionado, crear un installer ejecutando build.bat o importar el proyecto y compilarlo desde un IDE..
A continuación se muestran los pasos a seguir utilizando el installer:

4. Haz click en el instalador **linkServer-1.0.msi**.
5. Elige el destino en el que deseas instalar el chat.
6. Ejecuta el programa desde el icono mostrado en el escritorio.

Figura 8. Icono del servidor link chat tras la instalación

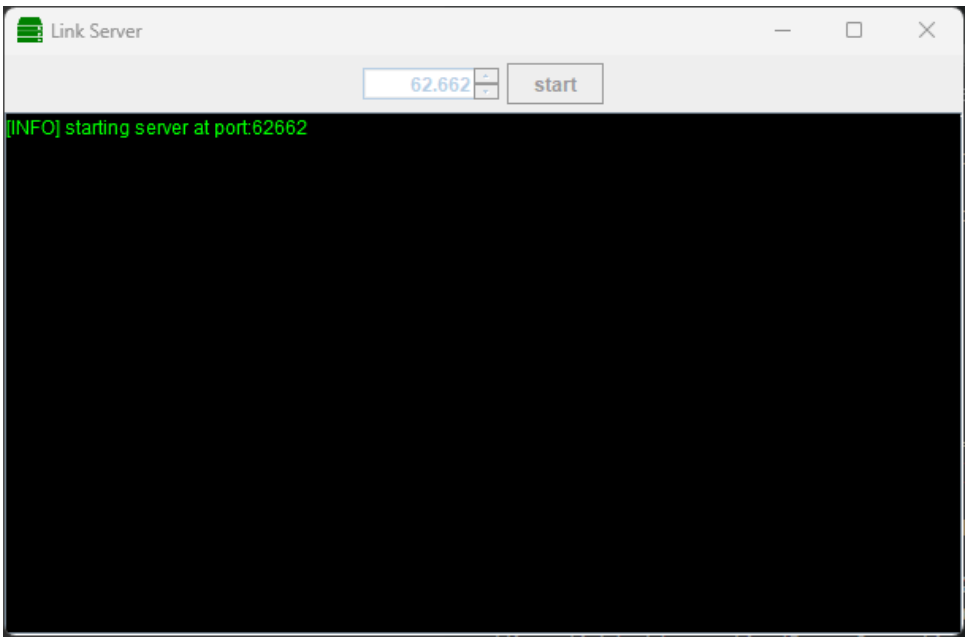

Figura 9. Servidor en ejecución