Práctica 2
Creación de un cliente IRC

Introducción

El objetivo de la segunda práctica es diseñar en C un cliente IRC para el intercambio de mensajes de texto, siguiendo, como se ha hecho anteriormente lo descrito los RFC y utilizando como guia HEXCHAT y XCHAT.
El cliente debe ser capaz tener varios canales abiertos, mantener conversaciones con otros usuarios conectados al mismo servidor y además, debe implementar el envío seguro de ficheros y el envío de audio.
Para poder levar a cabo todos estos requisitos, crearemos hilos para la recepción de mensajes enviados por otros usuarios. A la hora de enviar un fichero/audio procederemos de la misma manera, crearemos un hilo de envío/recepción de datos que permitirá mantener el resto de funcionalidades activas mientras se envía.

Al igual que en la practica anterior, todas las funciones a implementar estarán organizadas en una librería con el fin de facilitar el trabajo para el resto de prácticas, con su man correspondiente.

Diseño

El programa contiene los siguientes módulos:
● xchat2.c: Este es el modulo principal, tal y como se solicitaba, se implementaron todas las funciones que en el aparecían. Más adelante entraremos en detalles pero ahora destacamos el main. En el seteamos la funcionalidad del cliente e inicializamos los parámetros SSL en caso de ser necesario.
● messages.c: En este modulo se implementan las funciones que reaccionan a los mensajes del servidor. Se procesan estos mensajes y se realizan los cambios oportunos. De esta forma se consigue interactuar con el servidor de manera fidedigna y que el cliente sepa en todo momento el estado real del servidor, pudiendo reflejarselo al usuario.
● userCommand.c: En este módulo se encuentran las funciones que invoca el usuario al realizar un comando (“/XXX”). La funcionalidad de este modulo se basa en recibir lo solicitado por el usuario, y traducirlo para que el servidor puedo manejar dicha información una vez la reciba.

Funcionalidad IRC

Funciones relacionadas con los messages:
Todas las funciones de este modulo reciben como único parámetro el mensaje que el servidor le envío al cliente.
Dado a que el funcionamiento de todas las funciones es similar aquí mencionamos algunas de las más significativas.

Por cuestiones de implementación cabe destacar **msgWelcome**. Dado que para implementarlo de manera eficaz utilizamos expresiones regulares.
Por cuestiones de funcionalidad cabe mencionar **msgPrivmsg** dado que es gracias a ella a la que es posible mandar ficheros y audio como se explica a continuación.

Ademas destacamos que hay varios comandos de usuario cuyas respuestas se traducen en varias funciones de este modulo. Esto se debe a que cada uno de los mensajes que envía el servidor se

corresponde con un numero y es el parseo de cada uno de esos mensajes lo que se trabaja en este modulo. Es el caso de comandos como **WHOIS** o de mensajes del servidor como **MOTD**.

Funciones relacionadas con comandos de usuario:
Todas las funciones reciben como parámetro una cadena en el formato especificado en el RFC que deben seguir los clientes y una estructura SSL para implementar también el envío seguro de mensajes.
Se han implementado los comandos básicos que necesita un cliente IRC para considerarse funcional: Names, List, Join, Nick, Whois, Privmsg, Mode, Kick, Part, Quit, Away, Back y Topic.

Se decidió implementar estos dado que es la misma lista que se implemento en el servidor y de esta forma podemos lograr una completa interacción entre ellos.

Ficheros

Para implementar el envío de ficheros se han seguido las indicaciones de Moodle.

Todo el protocolo se ha realizado encapsulado en PRIVMSG. Esto permite que el servidor no tenga que implementar nueva funcionalidad. Las funciones implementadas se han modularizado en files.c. Se han creado 3 funciones, **fileDialog**, **fileSender**, **fileReceiver**.

FileDialog se encarga de realizar el handShake previo a la transmisión del fichero. Esta función es llamada tras recibir un código especial en PRIVMSG como se especifica en Moodle y se encarga de crear un hilo con el que conectarse al emisor y recibir datos mientras mantiene toda la funcionalidad del cliente activa.

Aquí hemos de comentar que no nos ha sido posible conectarnos al otro extremo de la comunicación si no era escribiendo la IP del emisor en el código.

FileSender intenta establecer una conexión durante un periodo de 30s. Si salta dicho timeout se cesa el intento de conexión con el extremo receptor.

FileReceiver se encarga de recibir el fichero, para la cual debe haber recibido correctamente el path donde escribir los datos y abre un descriptor con ese nombre. Posteriormente recibe mediante un bucle todos los datos enviados a través del sockect especificado anteriormente.

Audio

Para el envio de audio hemos decidido seguir un esquema similar al implementado para el envio de ficheros.

Se han implementado 3 funciones dentro de audio.c : **audioChat**, **audioSend** y **audioRecv.**

AudioChat realiza la etapa inicial, en la que se pide inicializar la conexión. En esta etapa volvemos a recurrir a PRIVMSG para encapsular la peticion. Tras esto crea un hilo que sera e encargado de enviar o recibir el audio.

AudioSend se encarga de enviar el audio, para ello inicializa un socket y si todo a ido correctamente graba el audio hasta que se presione el botón de parar. Una vez hecho esto se manda un mensaje especial en el que pone STOP. Para finalizar cierra el socket abierto y el hilo que tiene esta función.

AudioRecv sigue un esquema similar a la función anterior, abre un socket, lee todos los datos hasta procesar el mensaje de STOP y tras esto cierra el socket y elimina el hilo en el que se esta ejecutando.

Cabe mencionar que del mismo modo que en el envío de ficheros, es necesario en el audio escribir la IP en el código para que funcione correctamente.

Conclusiones técnicas

Durante esta practica la mayor dificultad que hemos tenido ha sido la implementacion de los ficheros y el audio. Sobre todo por los graves problemas que conlleva enfrentarse a NAT. Ademas, surgio un pequeño problema añadido, en los ordenadores de la EPS no se resuelve el host. Por ello nos vimos obligados a escribir a mano las IP en el código.

En cuanto al resto de la practica agradecemos la implementación de la API proporcionada, facilita en gran medida todo el trabajo, haciendo sencillo la interacción con la interfaz.

Conclusiones personales

Para esta practica el periodo de adaptacion fue más corto que el que necesitamos para acostrumbrarnos al servidor, pero no desprecable. Esto se debe a que tocaba cambiar el método, y el punto de vista.

Pensamos, que al contrario que en la practica anterior, las funciones tienen todas un nivel de dificultad similar, sin contar la implementacion del envio de audio y de ficheros, en lo que nos hemos visto con varios problemas, como ya se explica arriba.
Nos gustaria decir que a pesar de que se nota la disminución de carga de trabajo en comparación con la practica anterior, donde implementábamos el servidor, no es suficiente el periodo de dos semanas en las que se pretendía que implementásemos todo para la entrega opcional.