Práctica 2

Cliente IRC

Grupo 2311 - Pareja 02

Francisco Andreu Sanz

Javier Martínez Hernández

Francisco Andreu Sanz
Javier Martínez Hernández

Índice

Introducción:

Esta práctica consiste en la implementación de un cliente implementado **en lenguaje C** que siga el protocolo **IRC** utilizando las funciones de la librería **eps-redes2 .**

No se pide que sean implementados todos y cada uno de los comandos IRC existentes, pero sí se pide que sea robusto y carezca de errores.

Para probar el correcto funcionamiento de nuestro cliente hemos ido probando su funcionalidad con dos instancias del mismo viendo que tanto los mensajes como las respuestas concuerdan.

Evidentemente y al igual que el servidor, también sigue el protocolo según los estándares **RFCs-1459, 2812, 2813 y 2811.**

Diseño:

Nuestra práctica está compuesta del módulo **xchat2.c** principal del programa y un módulo que contiene más funciones implementadas llamado **clientfunctions.c** , así como su fichero de cabecera comentado siguiendo las pautas de **doxygen.**

El cliente ha sido implementado siguiendo la estructura de la función “básica” **IRCInterface_Run** y implementando el lado del cliente a partir de las llamadas a partes de la interfaz(como el botón de topic, recuadro de texto, etc). Para controlar el envío y recepción de mensajes hacemos uso de:

- **recv(...)** para el recibo de mensajes a través de sockets.
- **send(...)** para el envío de mensajes a través de sockets.

Para controlar **sockets**(en la conexión, envío y recepción de archivos...) haremos uso de las funciones:

- **socket(...), bind(...), listen(...)** para la creación, conexión y establecimiento de modo escucha de los sockets del servidor.
- **accept(...)** para confirmar el establecimiento de conexión entre socket del cliente y del servidor.

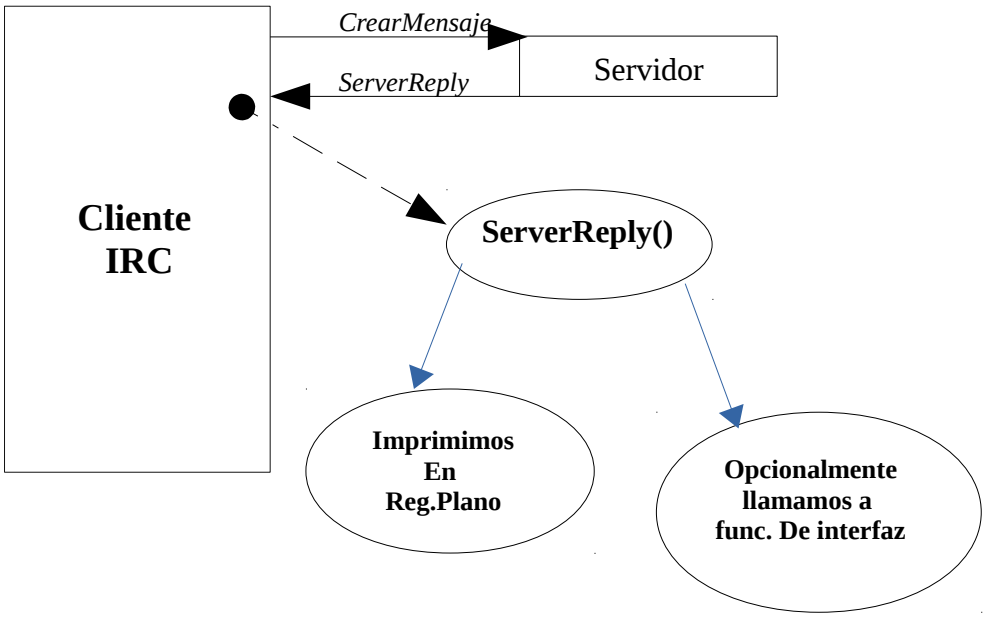
La manera con la que operamos es la siguiente:

1. Inicializamos y conectamos el **socket** del lado cliente al server(enviándole en la misma función de **connect** los datos del usuario). Además, creamos un hilo que envía PING periódicamente siempre y cuando el servidor responda. En caso de que pasado uno de esos períodos de tiempo el server no haya contestado, desconectamos el cliente.
   Otro hilo lo creamos para la recepción de mensajes del servidor, que emitirá continuos **recv** los cuales tratará(como veremos más adelante).
2. El programa queda esperando a que el usuario teclee algún **comando** o pulse algún botón.
3. En el caso de que el cliente haya tecleado un mensaje, llamamos a una función implementada por nosotros que comprueba qué tipo de mensaje hemos creado(con un **switch**) y dependiendo del mismo, ejecutará una u otra acción.

4. Tras ejecutar uno de los *case* de la función propia **crearMensaje,** en **casi** todos los casos(no en el caso de QUIT por ejemplo) se espera unarespuesta del servidor, que obviamente hay que procesar.
5. Para procesar la respuesta del server, tenemos que analizar todo lo que llega por el hilo del **recv**(hilo de recepciones de mensaje). Hacemos uso de otra función auxiliar llamada **ServerReply**, que procesa lo que llega a nuestro socket con un switch similar al de envío. En prácticamente cualquiera de estos casos imprimimos lo que nos llega por reg. plano(también lo hacemos con lo enviado) y, opcionalmente, **hacemos uso de alguna función de interfaz que nos sirva para el caso.**
6. Cabe destacar por último que, para cada grupo que nos unimos, **creamos un hilo para ese grupo.** El motivo de esto es que se emitan llamadas al comando **WHO** periódicas(similar a PING) para mantener la lista de usuarios actualizada. Cuando se sale de un canal ese hilo se cierra, así que ya no se enviarán más WHOS periódicos al mismo.

(*)Cabe destacar que para el comando PART hacemos una “ligera” modificación con respecto al XChat original. Cuando hacemos un PART, además de irnos del canal y eliminar nuestro NICK de la lista de NICKS, también cerramos la ventana del canal. Ésto es porque no hemos visto ningún botón en la interfaz para cerrarla y lo veíamos oportuno. Además ayuda al tema de los hilos por canal con WHO periódicos.

Un esquema muy simplificado del funcionamiento del cliente podría ser el siguiente:



Hay que añadir que la funcionalidad de los botones es muy similar a la de envío de mensajes. De hecho, la mayoría de ellos se limita a llamar a una función de crear mensaje y enviarlo.

Para la funcionalidad del envío de archivos lo que hemos hecho ha sido que la función de **SendFile** cree un socket y lo ponga a la espera de recibir un connect del receptor del archivo. Para notificar al receptor enviamos un mensaje con un patrón (**001 en ascii**) que contendrá el nombre de archivo, nombre de host... etc. Tras esto, se creará un socket que conectará con el servidor.

El envío de ficheros **es correcto para ordenadores que compartan una misma conexión a internet(wlan0).** Sin embargo, sólo hemos sido capaces de que funcione con archivos pequeños como una nota de texto.

Conclusiones técnicas:

Hemos aprendido a implementar el la parte **cliente** de un protocolo con todo lo que esto conlleva:

- Aprendizaje de generación de mensajes que se enviarán al servidor.
- Gran utilización de hilos.
- Comunicación entre dos clientes para el envío de archivos.

Conclusiones personales:

Aunque en un principio pensamos que no sería así, quizá esta práctica nos ha llevado más trabajo(al menos más lineas de código) que la parte servidor, y sobre todo tras varios intentos no hemos sido capaces de probar todo lo que nos gustaría el envío de ficheros y menos el de audio.