Redes de Comunicaciones II

Práctica 2 - Cliente IRC

Alfonso Bonilla Trueba
Mónica de la Iglesia Martínez

Pareja 7
Grupo 2313

Introducción

Esta práctica ha consistido en la elaboración de un cliente IRC implementado en C utilizando la librería de la asignatura **Redes de Comunicaciones 2** y el código base para el cliente xcaht2.

Para probar su correcto funcionamiento hemos probado la funcionalidad de los comando que acepta el cliente conectándolo tanto al servidor realizado en la práctica anterior como al servidor de la asignatura (metis.ii.uam.es). Hemos abierto dos instancias de nuestro cliente y hemos probado los comandos y visto que funcionan en ambos.

Al igual que el servidor de la práctica anterior, este cliente sigue el protocolo según los RFCs: RFC-1459, RFC-2810, RFC-2811, RFC-2812 y RFC-2813

Diseño

El módulo principal es *G-2313-07-P2-xchat2.c* que contiene el código base proporcionado con las funciones implementadas por nosotros. Este módulo principal para el cliente hace uso de dos módulos más, *G-2313-07-P2-ClientParser.c* y *G-2313-07-P2-ClientFunctions*.c
Toda la documentación de estos módulos ha sido generada mediante **doxygen** mediante comentarios especiales basados en tags.

Hemos implementando el lado del cliente a partir de las llamadas a partes de la interfaz (botón de topic, botón secret, cuadro de texto ...). Para controlar el envío y recepción de mensajes hacemos uso de:

- ➔ **recv()** para recibir los mensajes del servidor a través del socket
- ➔ **send()** para el envío de los comandos al servidor a través del socket

Para el control del socket, es decir, conexión, envío y recepción de archivos hacemos uso de las siguientes funciones:

- ➔ **socket()** crea el socket y devuelve descriptor del socket recién creado
- ➔ **bind()** asigna un puerto al socket
- ➔ **listen()** prepara el socket para escuchar conexiones
- ➔ **accept()** para confirmar el establecimiento de la conexión entre el socket del cliente y el del servidor

La manera en la que funciona el cliente es la siguiente:

1. Inicializamos el socket del cliente y lo conectamos al servidor enviando con la función **connect()** los datos del usuario. Además, creamos un hilo que es el encargado de enviar un PING al servidor de manera periódica cada 30 segundos y

que este responda. Si el servidor no contesta desconectamos al cliente. Otro hilo es el encargado de estar recibiendo los mensajes del servidor para tratarlos.

2. El programa queda esperando alguna acción por parte del usuario, ya sea mediante el cuadro de texto inferior, o pulsando algún botón de la interfaz.

3. Si el cliente escribe en el cuadro de texto inferior y se pulsa intro o el botón enviar se llama a una función de las que hemos tenido que implementar del código proporcionado, esta parsea lo escrito para generar el mensaje apropìado y enviarlo al servidor.

4. Tras parsear el comando y enviarlo al servidor, se espera siempre una respuesta excepto en el caso de QUIT.

5. Para procesar la respuesta del servidor hacemos uso de la función IRC_Reply_Parser, esta es muy similar a el parseador para componer los mensajes a enviar, solo que hace el trabajo opuesto, descompone las respuestas del servidor para tomar las acciones necesarias en nuestro cliente (por ejemplo, si se ha realizado correctamente el cambio de nick se actualiza el campo inferior izquierdo donde sale nuestro nick). Este parseador implementa todas las opciones posibles excepto la que se indican en el FAQ de la asignatura que no se deben hacer.

6. Después de parsear se realiza la acción en la interfaz gráfica que corresponda, ya sea cambiar el modo de un usuario en la lista, o un cambio de nick, o la operación más común que es escribir en el system o en la pestaña del canal activo.

Conclusiones Técnicas

Hemos aprendido a implementar un cliente IRC usando el protocolo.
Aprendizaje de generación de mensajes para enviar al servidor, así como parsear las respuestas que nos envía este.
Hemos profundizado un poco más en el manejo de hilos, ya que a estos les pasamos una estructura que contiene varios parámetros lo cual facilita la tarea y evita el uso excesivo de variables globales.

Conclusiones Personales

Aunque pensamos que esta práctica iba a ser más sencilla y corta en tiempo de implementación no ha sido así, ya que al tener que implementar dos parseadores (uno para los mensajes que se envían y otro para los que se reciben) es bastante más codigo que en la práctica anterior del servidor. Sobre todo a la hora de probar es mucho más complejo, puesto que esta práctica se han incluido el envío de ficheros y de audio. Y que no tenga un corrector como en la anterior teníamos r2d2 hace que la tarea de probar sea más lenta y pesada.

Además, con la caída de metis, el nuevo servidor de la asignatura no llega a funcionar bien, enviando a veces mensajes erróneos en formato, o repitiendo mensajes. Esto nos ha causado más de un problema a la hora de acabar el cliente.