Javier Gómez Martínez
Carlos Li Hu

Memoria Redes 2 Práctica 1

• Introducción: una descripción de lo que se pretende realizar en la práctica

En esta práctica vamos a diseñar y codificar un servidor que sigue el protocolo IRC, para ello empezaremos codificando un daemon que correrá nuestro servidor en segundo plano y abrirá un socket para escuchar y donde aceptará peticiones de conexiones nuevas para los diferentes clientes. En cada conexión, el cliente irá enviando diversos comandos que se irán interpretando y ejecutando de acuerdo con lo establecido por el protocolo IRC.

• Diseño: una explicación de los módulos de los que se compone el programa, así como de las decisiones que se han tomado (procesos o hilos, sincronización, etc.)

tools.c

En tools podemos encontrar varias funciones auxiliares que nos hemos declarado debido a su uso habitual a la hora de codificar comandos, de forma que haga el código más legible y más modularizado.

Entre ellas podemos encontrar las funciones de la familia “log***Error”, que imprimen una salida para saber si ocurre un error, hay 3 funciones, dependiendo de si la función que lo llama debe devolver un entero, un puntero, o void.
También encontramos funciones como el daemonizador, y varios getters útiles, como getIP (saca la IP de un cliente), getHostName (guarda el nombre de host del dispositivo), getNickFromSocket (obtiene el nick asociado a un socket), getSocketFromNick (obtiene el socket asociado a un nick),getAwayFromSocket (obtiene la cadena away de un socket asociado a un cliente).

socket.c:

Aquí guardamos las funciones de manejo de sockets del servidor para modularizar el main del servidor, con funciones para abrir el socket (opensocket), enlazar el socket al puerto (bindsocket), aceptar una conexión(accept connection) y conectarse al host (connectTo)

commands.c:

Aquí tenemos todos los manejadores de comandos del protocolo IRC que comprueba las pruebas del corrector r2d2.
Cabe destacar el comando commandDefault, que es el que se ejecuta si el cliente envía un comando que no se reconoce como ninguno de los implementados.
El resto de comandos siguen una estructura más o menos parecida:
Podemos ver como ,en general, todos los comandos empiezan haciendo un parse del comando para poder sacar los campos importantes del comando, y poder realizar las operaciones oportunas dentro de la función, y al acabar, mandar un mensaje específico al cliente para notificarle si se ha relizado correctamente o ha dado algún problema.

Hay comandos que requieren acciones adicionales más específicas para llevar a cabo su funcionalidad de forma correcta una vez se ha comprobado que el comando está bien formado. Entre ellas, están los controles de errores por si el usuario no tiene permisos para realizar un comando, con lo cual, simplemente se le comunica con el mensaje oportuno según el protocolo IRC. Entre los permisos y controles más habituales, está el ver si el cliente necesita ser operador de un canal para ejecutar una acción, o si el usuario objetivo está “away”.

server.c

Aquí tenemos nuestro main , es decir, el proceso principal que tendremos en segundo plano
Su funcionalidad principal es crear un socket para escuchar peticiones de conexiones de clientes, y

en cuanto las reciba, abrir un socket para esos clientes, y les mandará un hilo donde tendrán una rutina threadRoutine que se dedicará a atender los comandos del cliente.
Dentro de threadRoutine se ejecutará la funcionalidad más interesante, que esencialmente se compone de 2 funciones connectAndRegister y retrieveMessage:
La primera, se encarga de procesar los comandos nick y user del principio de la conexión, hasta que no se hayan asignado un nick y user válido al cliente, no permite que se ejecute cualquier otro comando.
Una vez se haya creado un usuario correctamente con su nick y user, se procede a llamar a la función retrieveMessage, la cual es la que se encarga de procesar los comandos en general, está contenido en un bucle que espera comandos, y los identifica con su respectivo función comando de commands.c, al cual llama para que ejecute el comando de forma correcta. Este proceso acaba cuando el usuario decide cerrar la conexión.
Además de esto, tenemos un threadPing corriendo de fondo, que se encarga de mandar pings a todos los usuarios del servidor esperando recibir un pong.
El tiempo en un ping lo hemos puesto cada 300s, debido a que no queremos que interfiera con el r2d2, puesto que si recibe un ping en medio de una prueba, puede considerarla incorrecta. Y el tiempo para ver si ha respondido está en el doble, es decir, si en 600s no ha contestado con un PONG o simplemente ha ejecutado una acción, le cerramos la conexión a ese cliente.

• Funcionalidad IRC: a grandes rasgos, qué funciones del protocolo se han implementado
A grandes rasgos, lo que hemos implementado de forma general son los los comandos principales de uso de un servidor IRC, es decir, funciones que se encargan de parsear los comandos que le envía un cliente, aplicar una acción correspondiente a dicho comando, y formar un mensaje de respuesta para que el cliente sea informado de que dicho comando ha sido procesado.

Para realizar estas conexiones cliente-servidor, el cliente manda una petición de conexión por el socket en el que está escuchando el servidor, con lo cual, el servidor le crea un hilo individual a cada cliente, con su propio socket de comunicaciones. Y en ese hilo, es donde se procesan los diferentes comandos que ejecuta el usuario.

• Conclusiones técnicas: temas concretos de la asignatura que se han aprendido al realizar la práctica
De forma general hemos aprendido cómo funciona el protocolo IRC de chats, cómo interpreta los comandos que manda un cliente, y el formato que tiene la respuesta que le corresponde a dicho comando en base al protocolo, y más importante aún, hemos aprendido a buscar información en los RFC oficiales del IRC.
De forma más personal, hemos aprendido a programar en pareja con un proyecto con bastante más volumen que los que hemos tenido hasta ahora, y a saber dividirnos el trabajo y a la vez colaborar para integrar nuestras aportaciones para que todo funcione al final.
También nos hemos familiarizado con bugzilla, una buena herramienta para reportar bugs, y hemos encontrado unos cuantos que han solucionado para que pudieramos completar la entrega.

• Conclusiones personales: a qué se ha dedicado más esfuerzo, comentarios generales
Lo que más esfuerzo ha llevado probablemente ha sido empezar la práctica, puesto que para poder hacerlo de forma correcta, primero había que entender qué tenemos que codificar exactamente, además de leerse toda la documentación que se nos ha proporcionado, y ser capaces de encontrar entre todas las fuentes de información que se nos proporcionan lo que necesitamos para hacer cada una de las funcionalidades específicas del servidor.
Una vez se ha conseguido empezar y ver que las cosas funcionan tal y como esperamos, saber empezar a codificar no era tan difícil, pero el siguiente problema es la cantidad de código a programar, de cara a la entrega, pues requiere aún sabiendo hacerlo, bastantes horas de codificación.

Pero en conclusión, ha sido una práctica en la que hemos trabajado mucho, y aprendido mucho sobre este protocolo, y sobre servidores en general.