Práctica 1
Creación de un servidor IRC

Introducción

El objetivo de esta primera práctica es diseñar en C un servidor que implemente el protocolo IRC (Internet Relay Chat) para el intercambio de mensajes de texto, siguiendo lo descrito en el RFC-2812.
Este servidor debe ser capaz de atender múltiples clientes, los cuales estarán activos mientras el protocolo PING PONG no indique lo contrario, en cuyo caso el socket debe ser cerrado. Por cada petición de clientes que se realice al servidor, se creará un hilo que la atienda (con un máximo de hilos simultáneos).
El servidor deberá ejecutarse en modo daemon, para que este se ejecute en segundo plano y no esté asociado a una terminal.
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, y se creará un makefile que empaquete correctamente todos los ficheros y cree los ejecutables de la práctica para probarlos después con r2d2.

Diseño

El programa contiene los siguientes módulos:
● server.c: Este es el modulo principal de nuestro programa, en el cual se recogen los parámetros de ejecución, se “daemoniza” el servidor y se asigna cada comando a su posición correspondiente (dada por la función IRC_CommandQuery) dentro del array de funciones que hemos decidido utilizar para facilitarnos la recepción de comandos del cliente.
Después de llamar a las funciones openSocket() y bindSocket(), definidas en socket.c, el programa se va a meter en un bucle infinito en el que por cada iteración se va a aceptar una petición y se va a crear un hilo para atender a dicha petición, haciendo pthread_detach para devolver los recursos al sistema una vez se termine de procesar la petición.Hemos preferido gestionar las peticiones mediante hilos ya que los procesos
acaparan muchos más recursos, a pesar de que estos sean más fáciles de manejar.
● Socket.c: En este módulo se encuentran las funciones que utilizamos para gestionar los sockets (apertura y conexión) y las funciones que se encargan de establecer la conexión con los clientes y aceptar las peticiones que estos envíen.
Para procesar los comandos enviados por el cliente hemos decidido hacer un array de punteros a funciones en el cual, las funciones de los comandos tendrán la misma cabecera, que incluye el comando correspondiente, el socket y el Nick, que hemos decidido incluirlo en la cabecera para evitar tener que obtener todos los datos del cliente desde dentro de las funciones de comandos, lo cual sería bastante más ineficiente.
● Command.c: En este módulo se encuentran las funciones internas que se encargan de procesar cada comando, haciendo los cambios correspondientes en el servidor mediante las funciones del TAD.
Hemos incluido al principio una función de gestión de errores básicos para hacer más vistosos y eficientes los controles de errores de cada comando.

Funcionalidad IRC

Funciones relacionadas con los sockets:

● OpenSocket: Usa la función socket() usando el parámetro SOCK_STREAM para utilizar el protocolo TCP.
● BindSocket: Rellena los campos de la estructura addr, se llama a bind() para informar al SO y espera hasta recibir peticiones de clientes.
● AcceptClient: Llama a la función accept() para aceptar las conexiones de los clientes.
● AttendClientSocket: una vez aceptada la conexión, recibe una petición de un cliente. En primer lugar inicia la conexión (función beginConnection()), a continuación recibes el/los comandos del cliente, los separa y según el comando accede a su posición correspondiente dentro del array de punteros a funciones para gestionar dicho comando.
● ConnectClientSocket : conecta el servidor a una IP y un puerto especifico, rellenando la estructura addr. Esta función no se ha usado durante la práctica debido a que está pensada para el cliente.
● BeginConnection: esta función gestiona la conexión de un cliente. Para ello el cliente debe enviar una pass (opcional), un nick y un user. Esta rutina funciona como una máquina de estados, que tiene en cuenta si se recibe o no una pass y se encarga de que si hay cualquier fallo en alguno de los pasos se vuelva al estado inicial (no se ha recibido ninguno de los tres comandos). Si laautenticación se ha realizado con éxito, terminará enviando un rplWelcome y ya podrá esperar a que el cliente envíe otros comandos.
● Daemonizar: pasa al servidor a modo daemon, para lo cual se cierran todos los ficheros, se crea un proceso hijo el cual haremos el líder de la sesión cambiando su ppid y establece el directorio raíz como directorio de trabajo, abriendo syslog para posterior notificación de mensajes.

Funciones relacionadas con comandos:

● sendErrMsg: dado un código de error crea el mensaje de error correspondiente (familia IRCMsg_Err) y lo envía al socket.
● FuncDefault: Comprueba si un comando recibido es incorrecto e informa al cliente.
● userCommand: ejecuta el comando USER
● NickCommand: busca al usuario y sustituye su Nick por el pasado por argumento.
● modeCommand: dependiendo del número de parámetros que nos pasen, devolverá el modo del canal, cambiará su modo o cambiará la contraseña del canal.
● quitCommand: elimina a un usuario de un canal.
● namesCommand: si se especifican uno o varios canales, devuelve la lista de usuarios de cada uno de ellos, si no se especifican, devuelve la lista de nombres de todos los canales del servidor (termina siempre con un mensaje EndOfNames).
● listCommand: si se especifican uno o varios canales, devuelve la información de estos canales (nombre, modo, topic...), siempre entre mensajes listStart y listEnd. Si no se ha especificado ningún canal, se devolverá la información de todos los canales existentes. Se tiene en cuenta si el canal es secreto y si el

usuario no está en el canal.
● privmsgCommand: Comprobamos por los parámetros pasados si el mensaje se quiere enviar a un usuario o a algún canal. Una vez sabemos esto usamos la función auxiliar privToUser, que se encarga de conseguir el socker destino, de crear el mensaje y de enviarlo tanto a un usuario (comprobando también que no esté en away) como a un canal.
● motdCommand: devuelve el "Message of the day" del servidor dado, si no se especifica servidor devuelve el mensaje del servidor en el que se encuentra el usuario.
● joinCommand: creamos un CompexUser y lo unimos a uno o varios canales según los argumentos del comando, una vez se haya unido a los canales se devuelve al usuario la información de cada uno de ellos. Si el canal no existía se crea.
● whoisCommand: devuelve la información de los usuarios cuyo Nick se pasa por argumento, terminado con un mensaje EndOfWHoIs
● pingCommand: ejecuta el protocolo PING-PONG
● topicCommand: si no hay argumentos, devuelve el topic del canal en el que se encuentra el usuario, si no se elimina o se modifica el topic del canal (puedevenir por argumento) si el usuario tiene permiso para ello.
● partCommand: el usuario abandona el canal o canales especificados por argumento, si hay un mensaje se mostrará en el canal, si no se mostrará un mensaje predeterminado.
● kickCommand: expulsa a otro usuario de un canal si tiene permiso para ello y con un mensaje que puede o no especificarlo por argumento.
● awayCommand: si se especifica un mensaje por argumento, el usuario pasará a estar en modo away con ese mensaje, si no se especifica mensaje el usuario dejará de estar en modo away.

Protocolo PingPong

Para la implementacion del protocolo ping pong hemos creado un array global en donde cada usuario escribe en la posicion que le toca según el id. De esta manera conseguimos que no haya solapamiento en memoria compartida y nos libramos de tener que utilizar semaforos.

Se han implementado 3 funciones:
Ping envía ping a todos los usuarios activos en el servidor (siendo un maximo de MAXCHANNEL usuarios).
La segunda funcion, **checkConnection**,comprueba si los usuarios han respondido pong, si esto no ha sido asi, se procede a su desconexion del servidor, cerrando su socket.
La tercera funcion se ejecuta en un hilo aparte, se llama **pingpong** y como su nombre indica se dedica a la realizacion de este protocolo. La funcion es un bucle infinito compuesto de: un sleep(120), un llamada a ping, sleep(15) y llamada a **checkConnection.**

NOTA: se ha decidido poner un sleep(120) para que no interfiera con la ejecucion de los correctores R2D2 o C3PO.

Conclusiones técnicas

La gran dificultad de esta práctica ha sido enfrentarnos a un problema teniendo únicamente las indicaciones de varios RFC y de las funciones de una biblioteca externa, recogidas en metis. Esto ha hecho que nos hayamos podido enfrentar a un entorno de desarrollo real, en el que además de tener que implementar el servidor hemos tenido que estar pendientes de bugs que pudiesen surgir en las funciones de la biblioteca y de las actualizaciones constantes tanto a esta como al corrector.
El corrector r2d2 ha sido de mucha utilidad, ya que además de servirnos para obtener un feedback constante de nuestro trabajo nos ha servido como guía para saber el orden correcto de implementación de funcionalidades del servidor.
Cabe comentar que para esta entrega no hemos podido implementar el protocolo PING-PONG, el cual estamos pensando en implementar con alarmas y esperamos que en la entrega final esté correctamente funcional.

Conclusiones personales

El comienzo de la práctica ha sido sin duda lo más complicado por el hecho de no saber por dónde coger los RFC ni la documentación dada.
Uno de los aspectos en los que nos hemos fijado en cuanto a la implementación es que ciertas funcionalidades son mucho más sencillas que otras. Principalmente la parte de inicio de conexión (pass, nick, user) ha sido una de las más tediosas, ya que al tratarse de una máquina de estados estuvimos bastante tiempo pensando cómo implementarla debido a que no estamos acostumbrados a este tipo de estructuras.
En cuanto a los comandos, salvo algunos que nos han llevado algo más de trabajo como JOIN o WHOIS, el problema general que hemos tenido han sido los bugs que llevaban inicialmente ciertas funciones (progresivamente se han ido solucionando), sobre todo de las familias Parse y TAD, que nos han hecho retrasarnos algo más de lo necesario, a pesar de que la mayoría de comandos no son muy complicados de implementar.