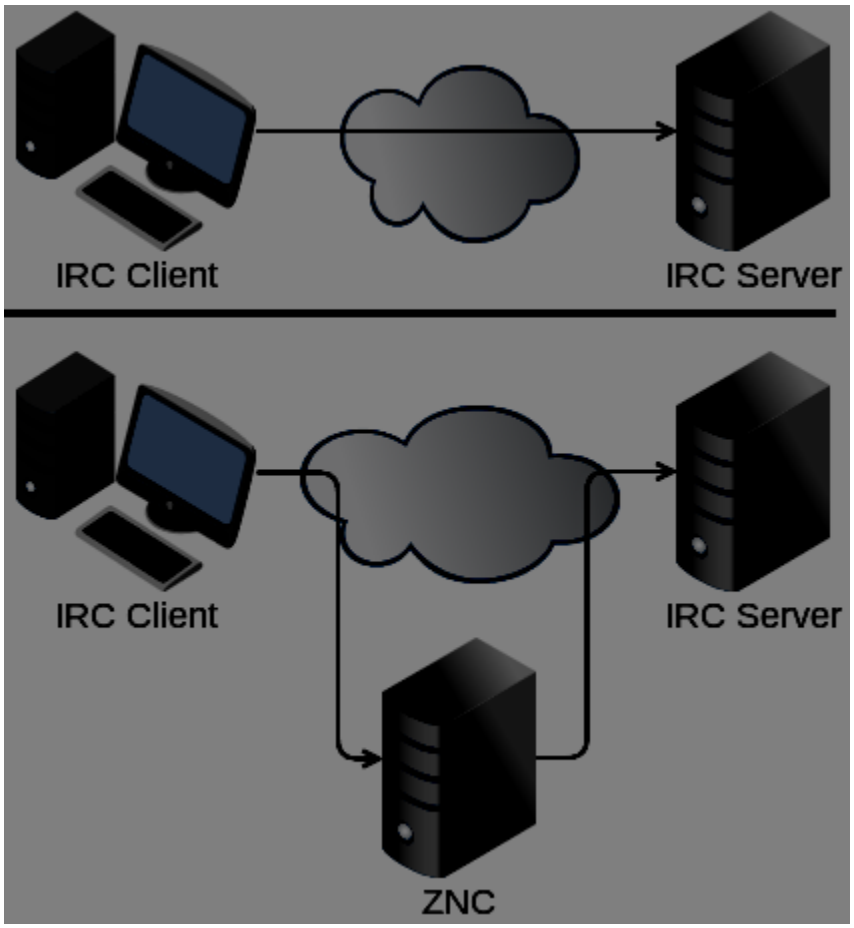
SERVIDOR IRC



Autores:
Pablo Marcos Manchón
Dionisio Pérez Alvear

ÍNDICE

INTRODUCCIÓN

En estas prácticas de la asignatura Redes de Computadoras II hemos desarrollado e implementado un servidor IRC completamente funcional. Lo primero que empezamos haciendo fue permitir el registro correcto de usuarios a nuestro servidor. Una vez implementado dicha funcionalidad, fuimos incorporando el reconocimiento de diversos comandos que ya se especificarán más adelante, lo cual fue añadiendo complejidad, a la par que completitud, a nuestro servidor.
Una vez dimos por terminada la parte del servidor, lo fuimos mejorando implementando, para empezar, el soporte de clientes, a lo que añadimos la transmisión tanto de ficheros como de audio en directo.
Para terminar, en las dos últimas semanas incorporamos al diseño del servidor el soporte de conexiones seguras por medio de SSL.

DISEÑO

A continuación expondremos las decisiones que tomamos a lo largo del desarrollo de la práctica. Lo dividiremos en diferentes partes correspondientes a cada una de las entregas parciales, pues creemos que es la manera natural de presentarlo, ya que muestra el proceso seguido a la hora de enfrentarnos a los problemas y buscar métodos para resolverlos.

Vamos primero con el desarrollo del servidor.
Para la realización consideramos básicamente dos alternativas de diseño: lanzar un hilo por usuario, y lanzar un hilo por mensaje.
La primera consideración, lanzar un hilo por usuario, hacía la tarea más sencilla desde el punto de vista del programador. Al tener que guardar toda la estructura del usuario etc. en un mismo hilo, el acceso y manejo de los datos era mucho más sencillo. Por contra, hacía que el rendimiento del servidor fuese mucho más bajo en el momento en el que el número de usuarios se veía incrementado drásticamente en un corto espacio de tiempo. Esto fue, entre otras cosas, lo que nos hizo decantarnos por la segunda opción.
Nuestro servidor está implementado con la alternativa de lanzar un hilo por mensaje. El proceso que seguimos para lidiar con los usuarios es el siguiente: para evitar la espera activa, y que esté en un bucle constante sin realizar ninguna tarea más que comprobar en todo momento si hay alguien a

quien atender, utilizamos la siguiente funcionalidad del select, que se desbloquea al recibir una señal que le es enviada. Cuando un descriptor es procesado por IRC lo quita del conjunto de descriptores; y una vez acaba de procesarlo lo añade al conjunto. A su vez, manda una señal que desbloquea al select y que, de otra manera, no haría nada. De esta manera, el select vuelve a quedarse esperando, pero con el descriptor ya añadido. Debemos mencionar que nos decantamos por este método, en lugar de por usar semáforos no bloqueantes, pues estos últimos no evitaban la espera activa.

Para procesar los clientes que se conectaban al servidor, decidimos crearnos una estructura temporal (desarrollada en el archivo conexion_temp.c) que almacenase el nick, el socket y la IP del nuevo cliente, así como el usuario previo y posterior, pues lo diseñamos como una lista doblemente enlazada para facilitar su acceso. En esta estructura almacenamos temporalmente los clientes desde que nos mandan el comando NICK hasta que recibimos el comando USER, momento en el que ya disponemos de los datos suficientes como para almacenar dicho usuario en la estructura proporcionada por Eloy.

Además, hacemos uso de dos semáforos, para controlar el acceso tanto al conjunto de descriptores como a la estructura de usuarios temporales, y así evitar condiciones de carrera.

También, cada vez que un comando es ejecutado por un cliente registrado en la base de datos, actualizamos el tiempo de su última acción (mediante la función IRCTADUser_SetActionTS). Mediante la rutina ping-pong, que utilizamos para saber si los clientes conectados siguen activos, el servidor les envía un PING cuando ve que su tiempo de última acción excede cierto tiempo previamente establecido, y espera su respuesta mediante un PONG. Si el pong es recibido, el tiempo de última acción del cliente es actualizado. Una vez que el tiempo de última acción ha excedido cierta cantidad de tiempo prefijada superior a la anterior, el cliente es expulsado del servidor. Esta última funcionalidad del servidor no la hemos implementado, pues las funciones de la librería de Eloy no actualizaban correctamente el tiempo de última acción. Con esto y todo, si al lanzar el servidor lo lanzas con la flag que activa la rutina ping-pong, cada 20 segundos se crea un hilo de atención a dicha alarma, a pesar de que no hace nada.

En cuanto al desarrollo del cliente, exponemos lo siguiente.
Por lo primero que empezamos fue por implementar los comandos del fichero xchat2.c, omitiendo aquellos que tenían que ver con el envío de ficheros o de audio, los cuales dejamos para el final.

Acerca del procesamiento de los comandos de usuario, destacar que acorde con el RFC, hacer un join 0 equivale a salir de todos los canales, lo cual utilizamos para implementar el comando partall.

Para el envío, tanto de ficheros como de audio, decidimos crearnos una estructura de cliente para poder almacenar una serie de campos que considerábamos imprescindibles; entre ellos: el socket, el puerto, la interfaz, y un par de variables que nos permitían ver en qué momento de la transferencia se encontraba el cliente. Además, decidimos no soportar el envío múltiple simultáneo de ficheros (el de audio estaba ya descartado), lo que hace que el cliente tenga que esperar a que se termine de enviar un fichero para poder enviar el siguiente.

Para el envío de ficheros, lo primero que hacemos es obtener un puerto en el que vamos a recibir, así como nuestra dirección IP, y se lo enviamos por mensaje privado al destinatario, con un código de envío de ficheros (\001FSEND). A la vez que sucede esto se crea un hilo con un temporizador, que limita la espera por parte del usuario que envía. Si el cliente no acepta el envío un FCANCEL es enviado, que provoca el cierre del puerto elegido para la transmisión, así como la muerte del hilo con el temporizador. Lo mismo ocurre si se agota el tiempo de espera, el hilo se muere y se cierra el puerto. En ambos casos, se avisa al cliente de la operación fallida. Antes de finalizar el envío, se comprueba que el FCANCEL proviene del mismo usuario a quien queríamos

enviar el fichero, para evitar pérdidas cruzadas de ficheros. Esto lo implementamos pensando en un posible envío múltiple de ficheros, cosa que al final no implementamos. En caso de aceptar el receptor el envío del archivo, vamos enviando el fichero en paquetes fragmentados de tamaño máximo 8192, hasta que todo el fichero ha sido enviado. En ese momento cerramos descriptores, puertos etc., que se hubieran estado usando, y actualizamos el estado del cliente (que había sido modificado en el momento que lanzamos el hilo del timeout) para que pueda volver a recibir.

Para recibir los ficheros, lanzamos un hilo que se encarga de esperar a que la otra parte nos acepte la conexión. Además, antes de empezar a recibir el fichero, se comprueba que el cliente no tenga ya una conexión activa con otro cliente (comprobando el status del cliente). Si el cliente se encuentra en disposición de recibir, y ha sido aceptado por el emisor, vamos recibiendo el archivo y lo volcamos en un fichero. Una vez terminado, cambiamos el status del cliente; cerramos descriptores, puertos, etc., y matamos el hilo de recepción.

Como curiosidad práctica, con dos ordenadores conectados por wifi al mismo servidor fuimos capaces de transmitirnos un archivo de 1.4 Gb. También fuimos capaces de transmitir archivos con otros clientes que también habían implementado la transmisión de ficheros de manera similar a la descrita en el FAQ, utilizando FSEND y 001 como carácter de control.

A continuación explicaré la decisión que tomamos para el envío de audio entre dos clientes. Lo primero es que si se desea tener una comunicación bidireccional ambos clientes deben solicitar la apertura de audiochat al otro usuario. Por otra parte, el cliente que solicita la apertura del audiochat será el cliente que escuche, no el que emita. Para establecer una conexión de audio, el receptor abre un puerto UDP que será el que se utilizará en el hilo de recepción de dicho cliente. Lanza también el hilo de recepción, el cual escucha en el puerto que se le ha pasado al emisor, y en el que se reproducirá el audio. A continuación manda un mensaje que empieza con AUDIOCHAT indicando el puerto e IP privadas donde espera recibir. Si todo ha ido bien, y el emisor acepta la transmisión de audio, lanza (el emisor) un hilo de envío, y abre un socket UDP para poder enviar al puerto recibido lo grabado. Si se desea una comunicación bidireccional, en la que ambos clientes se comuniquen a través de voz, los papeles deben turnarse, y el proceso ha de ser repetido.

A la hora de ejecutar el cliente, obtenemos la interfaz que utiliza por defecto accediendo al archivo /proc/net/route. Sin embargo, esto solo es válido en sistemas con distribución UNIX, por lo que damos la opción, mediante la flag -i, de especificar la interfaz que quiere usarse.

En cuanto a la inclusión de SSL a nuestra práctica, pocas decisiones tuvimos que hacer al respecto, puesto que el guion de la práctica ya marcaba muy claramente el camino a seguir. Mencionar, eso sí, que decidimos mantener el contexto SSL creado como variable global, al igual que una estructura donde guardamos para cada socket su correspondiente estructura SSL asociada. También declaramos una variable global que determinase si el cliente tenía activada la opción de conexiones SSL.

FUNCIONALIDAD IRC

Para el desarrollo del servidor hemos desarrollado e implementado las siguientes funciones del protocolo:

Primero implementamos los comandos NICK y USER, para poder registrar usuarios en nuestro servidor. A continuación implementamos los comandos JOIN y LIST, con los que nuestro servidor empezaba a poder hacer grandes cosas ya.

Lo que hicimos después fue implementar el resto de comandos requeridos para pasar el r2d2 en la primera entrega parcial. Como añadidura, mencionar que también implementamos el procesamiento de los comandos Rpl_Whoisuser y Rpl_Whoisserver a la hora de procesar el comando whois. Además, también implementamos el comando WHO a pesar de que tampoco se pedía, para así poder ejecutarlo con el hexchat.

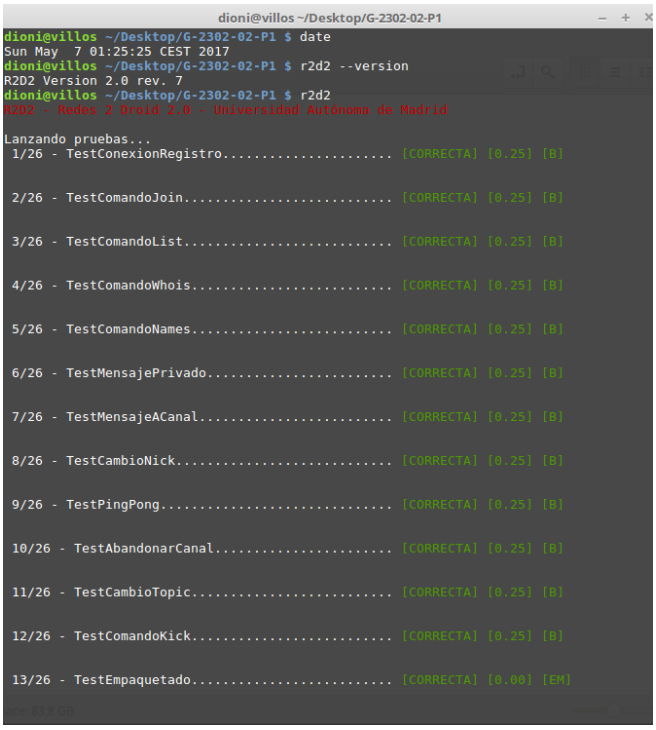
Otro comando que merece mención aparte es el comando QUIT. Sabemos que al hacer un QUIT hay que enviarle un mensaje a todos los usuarios que de alguna manera tengan relación con quién se va, esto es, que compartan algún canal en común. Sin embargo, al incluir esas líneas de código en nuestro programa se cerraba la conexión repentinamente y se caía el servidor. Tras probar numerosas cosas logramos aislar el error a un caso de error de la función que envía mensaje

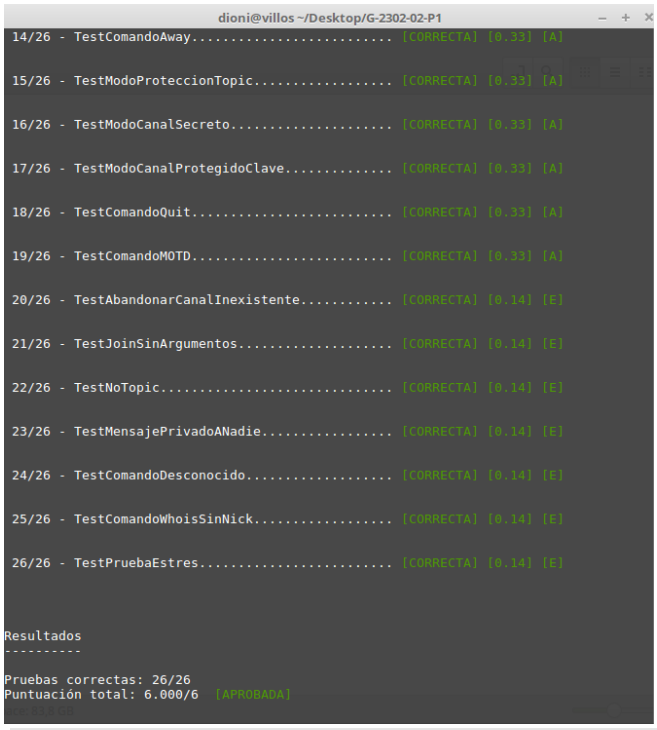
privado. Además, creemos que el fallo es debido a las librerías de sockets, que son ajenas a nuestro control. Por ello, en vez de comentar dicha parte del código hemos incluido una flag en el archivo makefile (inicialmente comentada) para que cuando ésta este activada se compile también esa parte del código, por si en un futuro ese error desapareciera.

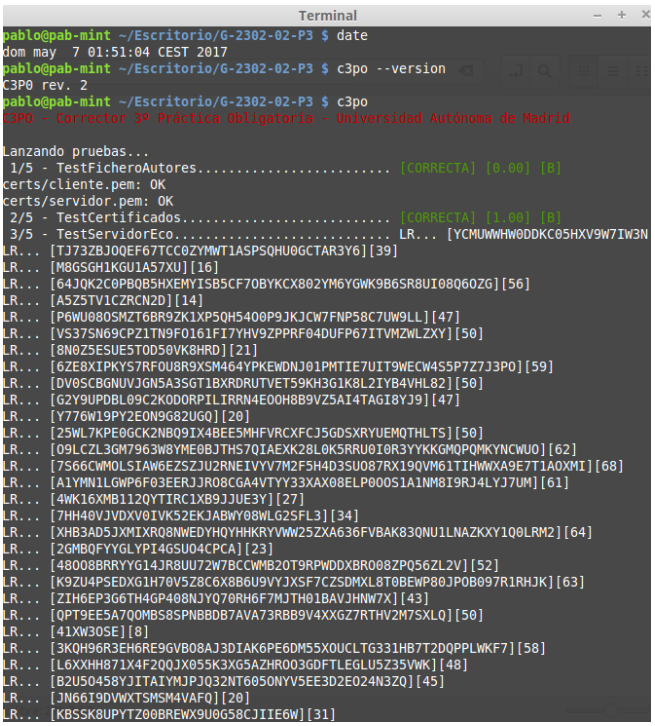
A la hora de desarrollar el cliente, implementamos todos y cada uno de los comandos que así se indicaba tanto en el guion como en metis. Empezamos por las funciones de la interfaz del xchat2, de las cuales ya venía el prototipo de las mismas. De la mayoría de los comandos del cliente, de los mensajes que recibimos del servidor, en caso de estar bien formados, eran devueltos prácticamente tal cual. Destacamos un par de comandos especiales, que el cliente puede ejecutar sobre la consola, como son el comando cycle, el part y el partall.
En cuanto a las funcionalidades del SSL, para hacerlo compatible con el resto de la práctica tan solo tuvimos que modificar las funciones de envío en los archivos red_servidor.c red_cliente.c, puesto que gracias a la librería que habíamos desarrollado de sockets TCP no utilizábamos nunca directamente la función send.

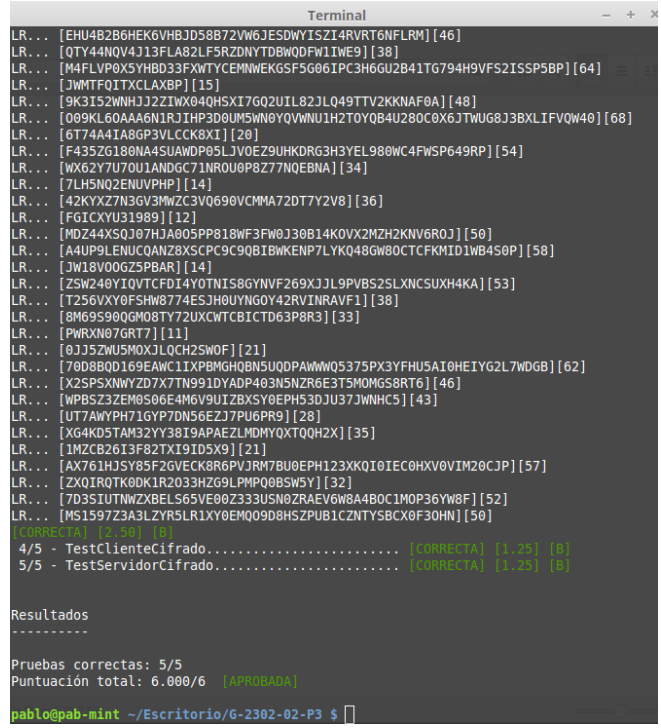
DEMOSTRACIONES PRÁCTICAS

A continuación presentamos 4 capturas de pantalla que demuestran que nuestro proyecto pasa satisfactoriamente ambos programas correctores: el r2d2 y el c3po.









CONCLUSIONES TÉCNICAS

La realización de esta práctica, si bien no iba muy en consonancia con lo visto en teoría, sí que comparte algunos temas cruciales, y que ha servido para consolidar.

Quizá lo más destacable, y que más nos haya servido y ayudado, sea el tema de aplicaciones de red, comunicación entre procesos y programación de sockets. Sin esto el desarrollo de la práctica habría sido tarea imposible, pues es fundamental para poder llevarla a cabo desde el día 1. Además, el tema de la programación de sockets se extendía hasta la adición del SSL a nuestra práctica, con lo que al final de la misma podemos decir que es la funcionalidad que más reforzada hemos visto.

Junto con lo anterior, lo segundo que más hemos trabajado de la asignatura ha sido el tema de seguridad en la red. A la hora de desarrollar los certificados de confianza y de establecer las conexiones seguras la parte vista en teoría nos ayudó a entender al principio cómo funcionaba. Pero sin duda el tener que implementar toda la funcionalidad de SSL, y juntarlo con la implementación de sockets, y que no funcionase ni a la primera, ni a la segunda, y pelearnos con ello nos ayudó a entender mejor la teoría.

En definitiva, sin ser unas prácticas que sigan el desarrollo lineal de lo visto en teoría y ayuden a reforzar lo trabajado en clase, las partes en que ambos, teoría y prácticas, coinciden han servido para reforzar de manera notable dichos temas.

CONCLUSIONES PERSONALES

En lo referente a esta práctica, su realización a supuesto un reto por varios motivos.

En primer lugar, la manera de organizarnos es bastante diferente a lo que hemos estado acostumbrados. En esta ocasión hemos dispuesto de un organigrama orientativo que no teníamos por qué seguir, y de tres guiones que te indicaban un poco lo que tenías que hacer, pero dejaba muchas decisiones al criterio del programador. Esto, unido al hecho de que en este caso no se trataba de realizar varias prácticas de 2-3 semanas, sino de una de 4 meses, ha provocado que hubiese momentos de relajación, y otros de gran estrés debido a la dificultad que tiene el medir el tiempo que nos podía llevar el desarrollar las diversas partes de la práctica.

Otra gran dificultad que nos hemos encontrado es la falta de información a la hora de llevar a cabo la práctica. Sí que es verdad que en la página de metis se nos indicaba donde podíamos o debíamos informarnos de cada una de las funciones o de las dudas en general que nos surgieran a lo largo del desarrollo de la práctica, pero eso no significa que fuera fácil encontrarla. Más de una vez nos hemos encontrado atascados durante la implementación de la práctica por no saber cómo realizarlo. Sin embargo esto también tiene sus ventajas, pues otra manera de sacar información era 'preguntárselo' al servidor de metis, al que podíamos tomar

como modelo a seguir. De esta manera, cualquier duda podíamos recrearla y estudiar su comportamiento.

CONCLUSIÓN

En definitiva, esta práctica ha sido diferente a cualquier otra práctica que hayamos tenido, por las razones que hemos comentado en puntos anteriores. Las prácticas diferían en gran medida de lo expuesto en teoría, y se trataba de una única práctica de larga duración, en vez de varias prácticas pero más cortas, que era a lo que estábamos acostumbrados. También, estábamos mucho menos guiados que en prácticas anteriores, pues los guiones que venían para cada una de las entregas parciales eran meras directrices. Con esto y todo, creo que la manera de enfocar estas prácticas, mucho más realista y enfocada al mundo laboral y a lo que nos vamos a encontrar, nos ha sido bastante útil, y nos ha servido para aprender muchas cosas aparte de lo meramente educativo.