Práctica1
Creación de un servidor IRC

Eloy Anguiano

Rodrigo Castro
Ignacio Fernández

Fecha de inicio: 29 de enero de 2014 (Grupos de los miércoles). 7 de febrero de 2014 (Grupos de los viernes)
Fecha de entrega: 5 de marzo de 2014 (Grupos de los miércoles). 14 de marzo de 2014 (Grupos de los viernes)

1. Diseño de un servidor IRC

Esta práctica consiste en el diseño de un servidor IRC.El protocolo IRC (*Internet Relay Chat*) es un protocolo de aplicación ideado para el intercambio de mensajes de texto (ver detalles en el RFC 1459[1]). Este protocolo utiliza conexiones *TCP*[2] para realizar las comunicaciones entre los distintos agentes que intervienen en el intercambio de mensajes.

La Figura 1 muestra de forma esquemática el proceso completo de encapsulamiento de un mensaje enviado por el usuario. El alumno debe tener en cuenta que al trabajar con *socket* no tendrá que implementar todo este proceso. El texto escrito por el usuario (en el ejemplo "hola") se encapsula dentro de un mensaje *IRC* (capa de aplicación) que se pasa a la capa de transporte. En la capa de transporte el mensaje *IRC* se encapsula en un mensaje *TCP* que se pasa a la capa de red. De la misma forma, en la capa de red el mensaje *TCP* se encapsula en un mensaje *IP* que es el que se transmite por el canal de comunicaciones. Una vez el mensaje llega a su destino, el proceso es inverso, de forma que el mensaje recibido va subiendo por la pila TCP/IP hasta que llega al nivel de aplicación. En esta práctica el alumno se centrará en las comunicaciones a nivel de capa de aplicación. Para ello deberá trabajar con *socket* TCP que, una vez creado el mensaje *IRC*, se encargarán de enviarlo a su destino de **forma transparente** para el desarrollador.

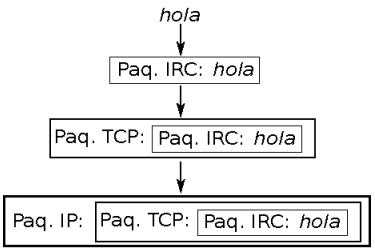

Figura 1: Diagrama de encapsulamiento.

Como todo servidor debe ejecutarse en linux como un `daemon`.

1.1. Nomenclatura de fuentes y directorios

Antes de describir estas funciones es importante indicar las condiciones de trabajo en cuando a directorios y ficheros. Son las siguientes:

- Debe crearse un directorio con el siguiente nombre G-CCCC-NN-PX donde CCCC es el número de la clase de prácticas y NN es el número de del grupo **siempre con dos dígitos** y X es el número de práctica.
- Dentro de ese directorio sólo estará el *makefile*, la documentación y los directorios que se indican a continuación. Sobre él se crearán los ejecutables necesarios.

[1] RFC 1459: http://tools.ietf.org/html/rfc1459
[2] *Transmission Control Protocol*, RFC 793: http://tools.ietf.org/html/rfc793

- Los nombres de todos los ficheros: *makefile*, fuentes, librerías, ejecutables, documentación o fichero comprimido para la entrega deberán empezar por G-CCCC-NN-PX según la indicación anterior. El resto del nombre es a elección del estudiante salvo en los siguientes casos:
  - El *makefile* que, obligatoriamente deberá llamarse G-CCCC-NN-PX-makefile.
  - La documentación que deberá llamarse G-CCCC-NN-PX-doc.pdf. Este nombre implica, por supuesto que toda la documentación debe entregarse en formato PDF.
  - Los ficheros para el comando `man` deben llevar los nombres adecuados para que funcionen correctamente.
  - La librería deberá llevar un nombre propio de una librería y es aconsejable que sea una única librería para todas las prácticas.
- Dentro del directorio anterior se crearán los siguientes directorios:
  1.– Un directorio `src` donde se almacenarán los fuentes.
  2.– Un directorio `srclib` donde se almacenarán los fuentes que se vayan a acompilar para una librería.
  3.– Un directorio `includes` donde se guardarán los includes (.h)
  4.– Un directorio `lib` donde se almacenarán las librerías (.a).
  5.– Un directorio `obj` donde se almacenarán los objetos (.o) que deberá estar vacío en el comprimido.
  6.– Un directorio `man` donde se almacenarán los ficheros para el comando `man`.
- El directorio G-CCCC-NN-PX es el que deberá ser comprimido en formato `.tgz` (mirar el comando `tar` con el comando `man`). Es importante comprobar que si se descomprime en cualquier directorio este fichero, aparece el directorio base de la práctica.
- Bajo ningún concepto deberán incluirse ejecutables en el archivo comprimido, de tal forma el *makefile* deberá borrar los ejecutables cuando se solicite la compresión.

Dado que las funciones se almacenan en una librería y que están documentadas con `man` es aconsejable también tener disponibles los `man` en el directorio adecuado para que sean accesibles y la librería en un directorio de sistema accesible a la hora de compilar. Si están bien diseñadas y documentadas estas funciones pueden ser de mucha utilidad en el futuro.

El incumplimiento de alguna de estas condiciones supondrá una reducción de nota considerable. Es por tanto conveniente revisar que se cumplen todas desde el principio.

1.2. Librerías, makefile

Todas las funciones deben integrarse en una librería para compilar en C. Para ello es necesario crear un *makefile* adecuado que elimine de la librería la función anteriormente introducida (si es necesario) e introduzca la nueva. Sólo el programa principal que use las funciones que se van a realizar no deberá estar en la librería pero también deberá compilarse el `daemon` si ha sido modificado o ha cambiado la librería. Para crear la librería y modificarla usar el comando `ar` propio de la creación de librerías de linux. También deberá crear el fichero comprimido si solicitas como parámetro del comando `make`. Así mismo podrá crear los ejecutables de pruebas si solicita con distintos parámetros.

En la sección de la documentación en la que se explique el fichero de *makefile* deberán indicarse también los parámetros éste que se hayan diseñado.

2. Función para "daemonizar"

Cualquier servidor debe estar en forma de `daemon`. Existen una serie de tareas que deben realizarse antes de que se inicie el servidor. En las siguientes líneas se van a describir las acciones que se deben realizar pero sin indicar la función en C que debe utilizarse. Después se aportará una serie de funciones en C útiles para desarrollar estas acciones. Es trabajo del alumno informarse sobre el uso de las funciones. Para ello se aconseja el uso de la función `man` en una terminal. A lo largo de todo el proceso debe informarse al sistema de `log` de unix de las tareas que se van realizando porque, como se verá, al perder el control de las terminales es imposible enviar mensajes a la terminal.

En la función de daemonizar debe recibir un literal (cadena de caracteres) que permitirá la identificación de mensajes en el `log` y devolverá un código de error o cero en el caso de que no exista ningún error. Ésta función deberá realizar las siguientes tareas:

1.– Se capturan las señales `SIGTTOU`, `SIGTTIN` y `SIGTSTP` que provienen de la terminal con la función de C `signal` que enviará la captura a la función de tratamiento de señales predefinida `SIG_IGN`.
2.– Se crea un proceso hijo.
3.– Se cierra el proceso padre.
4.– Se crea una nueva sesión de tal forma que el proceso se convierte en el líder de la sesión.
5.– Se pierde el control por parte del `tty` capturando la señal `SIGHUP` con la función `SIG_IGN`.
6.– Los ficheros creados por el servidor deben ser accesibles a todo el mundo. Para ello es necesario cambiar la máscara de creación de ficheros.
7.– Por seguridad se puede cambiar el directorio de trabajo a, por ejemplo, en raíz.
8.– Es necesario cerrar todos los ficheros abiertos previamente.
9.– Se capturan las señales `SIGCHLD` y `SIGPWR` con una función que debe crearse, que espera a que termine el demonio debido a estas señales y cierra el `log`.

Las funciones de C a utilizar en las funciones anteriores son ordenadas alfabéticamente son: `closelog`, `fork`, `getdtablesize`, `openlog`, `setsid`, `signal`, `syslog`, `umask`, `wait`, `waitpid`.

Se deben realizar pruebas para comprobar el funcionamiento correcto de la función de daemonizar así como de su comportamiento.

La función para "daemonizar" devolverá un código de error en caso de que sea necesario. Los errores serán todos negativos salvo si no hay error, en cuyo caso se devolverá un cero.

Esta función o, si es necesario, funciones, también deberán estar documentadas en forma de `man`.

En la sección de la documentación donde se explique el funcionamiento de esta función debe no sólo explicarse el diseño de ésta sino también las pruebas realizadas para comprobar que es correctamente funcional

3. Funciones de apertura de puerto y espera de conexión

Debido a que en una práctica posterior se añadirá la seguridad SSL es necesario realizar un diseño de dos niveles de las funciones necesarias para realizar esta tarea.

En este estadio de desarrollo sólo habrá tres funciones de nivel superior.

1.– Función que iniciará servidor, es decir, abrirá el *socket*, realizará las asignación del *socket* al puerto deseado y abrirá una cola de escucha de una longitud determinada. Esta función devolverá un código de error y tendrá dos parámetros:
   1.1.– El número de puerto en el que estará esperando recibir las comunicaciones.
   1.2.– La longitud máxima de la cola de espera de conexiones.
2.– Función que pondrá al servidor a "escuchar" peticiones de conexión. Devolverá un código de error si la conexión no se ha realizado.
3.– Función que cierra la comunicación. Tendrá como parámetro el *handler* de la conexión a cerrar y devolverá un código de error.

Las funciones de bajo nivel serán :

- Función que abre un `socket` de servidor TCP. No tiene parámetros. Devolverá un código de error o el *handler* del `socket` según corresponda.
- Función que asigna un puerto al `socket`. Tendrá dos parámetros que serán el *handler* del `socket` y el número de puerto. Devolverá un código de error.
- Función que determina la longitud de la cola. Tendrá dos parámetros que serán el *handler* del `socket` y la longitud de la cola. Devolverá un código de error.

Todas las funciones devolverán un código de error en caso de que sea necesario. Los errores serán todos negativos salvo si no hay error, en cuyo caso se devolverá un cero o el valor deseado.

Todas estas funciones utilizarán el sistema de `log` para almacenar la información que se considere adecuada. Dado que algunas de estas funciones se repetirán en el caso del cliente que se realizará en otra práctica y dado que las funciones del servidor utilizan el sistema de `log` es necesario que en el nombre de las funciones se distinga que su uso se realizará en el servidor.

Las funciones de C que deben mirarse para la implementación de estas funciones son: `accept`, `bind`, `listen`, `socket` y `waitpid`.

Estas funciones también deberán estar documentadas en forma de `man`.

En la sección de la documentación donde se explique el funcionamiento de estas funciones debe no sólo explicarse el diseño éstas sino también las pruebas realizadas para comprobar que son correctamente funcionales.

4. Funciones de recepción y envío de mensajes

Para enviar y recibir hay que utilizar las funciones de C `send` y `recv`. Como ambas funciones sólo pueden enviar como máximo el tamaño de un segmento TCP será necesario construir dos funciones de más alto nivel para enviar y recibir grandes cantidades de datos.

La función que envíe los datos tendrá como parámetros un puntero a los datos (que será de tipo void *) y la longitud en bytes a enviar. Devolverá la longitud en bytes enviada o un código de error (negativo). Enviará los datos en paquetes del tamaño máximo de un segmento salvo el último que sólo enviará los datos restantes.

La función que recibe los datos tendrá como parámetro un puntero a un puntero a los datos (que será de tipo void **). Devolverá la longitud en bytes recibida o un código de error (negativo). Esta función hará sitio en la memoria para un buffer del tamaño adecuado a esa longitud (cuidado con la terminación de cadena de caracteres) y rellenará el buffer con los datos. Cuidado, cuando se llame a esta función será necesario recordar que el buffer será necesario liberarlo.

Todas estas funciones utilizarán el sistema de `log` para almacenar la información que se considere adecuada. Dado que algunas de estas funciones se repetirán en el caso del cliente que se realizará en otra práctica y dado que las funciones del servidor uilizan el sistema de `log` es necesario que en el nombre de las funciones se distinga que su uso se realizará en el servidor.

Estas funciones también deberán estar documentadas en forma de `man`.

En la sección de la documentación donde se explique el funcionamiento de estas funciones debe no sólo explicarse el diseño éstas sino también las pruebas realizadas para comprobar que es correctamente funcional

5. Funciones de procesamiento de los mensajes

Para procesar los mensajes es necesario crear una función que tenga como parámetros el mensaje recibido y un apuntador a un puntero de caracteres. Esta función identificará la instrucción en el mensaje, podrá el puntero a carácter en el primer carácter después de la instrucción o NULL si no hay más mensaje y devolverá un entero que indicará la instrucción solicitada.

Se recomienda el uso de `strncmp`. El valor entero deberá estar indicado en forma de `enum` de C. Así mismo se puede utilizar un array predefinido con las cadenas de texto correspondientes al comando. En clase de prácticas se darán algunas pistas de cómo realizar esta función.

De igual forma, para los comandos que un usuario puede introducir en un mensaje se utilizará una función similar en la que se utilizará `strstr` para encontrar dicho comando y se creará otra función que identifique en un texto los comandos de usuario.

Se creará una función para el procesamiento de cada instrucción IRC.

Todas estas funciones utilizarán el sistema de `log` para almacenar la información que se considere adecuada. Dado que algunas de estas funciones se repetirán en el caso del cliente que se realizará en otra práctica y dado que las funciones del servidor uilizan el sistema de `log` es necesario que en el nombre de las funciones se distinga que su uso se realizará en el servidor.

Es necesario realizar pruebas que comprueben de forma completa el correcto funcionamiento de las funciones así como de la devolución de errores de éstas. En la sección de la documentación correspondiente se indicarán las pruebas realizadas y los parámetros del *makefile* necesarios para compilar dichas pruebas. Las pruebas siempre deberán realizarse con las funciones introducidas en la librería.

Estas funciones también deberán estar documentadas en forma de `man`.

En la sección de la documentación donde se explique el funcionamiento de estas funciones debe no sólo explicarse el diseño éstas sino también las pruebas realizadas para comprobar que son

correctamente funcionales.

6. Creación del "daemon"

Con las funciones creadas anteriormente deberá crearse el servidor en forma de *daemon*. Para ello se "daemonizará" el proceso y se abrirá la conexión del lado del servidor quedándose a la espera de petición de conexión.

Tras aceptar la conexión el *daemon* deberá realizar todos los comandos de conexión del IRC y después lanzará un hilo o un proceso (según desee el alumno) que atenderá a esa conexión y al resto de peticiones IRC. Así el *daemon* volverá a esperar a una nueva petición de conexión.

Para probar que el servidor funciona correctamente se pueden utilizar múltiples clientes de chat disponibles para linux como, por ejemplo `xchat`.

6.1. Algunas observaciones

Como los datos de conexión de los múltiples clientes deberán ser accesibles desde múltiples clientes es aconsejable que se defina una colección de estructuras con estos datos y que sean accesibles desde todos los hilos o procesos que atienden a la conexión. En el caso de hilos la solución más sencilla es que esta colección sea accesible globalmente. En el caso de usar procesos, esta colección deberá definirse en una zona de memoria compartida.

Para la creación de hilos es aconsejable el uso de la librería `pthread`. Es importante recordar que, salvo las variables que se definen dentro de la función que va a atender al hilo, el resto de variables es común a todos los hilos, incluso las variables que se pasan por apuntador al propio hilo por lo que es posible que sea necesario hacer una copia interna al hilo nada más entrar en él.

6.2. Funciones de IRC

En el RFC correspondiente al protocolo IRC antes indicado se encuentra definida la funcionalidad completa del protocolo IRC. El alumno deberá interpretar qué funcionalidad es esencial, cuál es importante y cuál una funcionalidad poco usada. Con ello establecerá un orden de diseño de funcionalidad que irá implementando y probando incrementalmente. Buena parte de la evaluación de esta práctica estará relacionada con la cantidad de protocolo implementada y, sobre todo de qué parte del protocolo está implementada. Así, para aprobar no es necesario hacer más allá de la funcionalidad básica. En ningún momento los profesores indicarán cuál es la funcionalidad básica puesto que esto es parte de la tarea del alumno: saber analizar el IRC y discriminar la importancia de cada funcionalidad es parte de los objetivos de aprendizaje de esta práctica.

Se intentará que, salvo casos especiales, cada funcionalidad de IRC sea realizada por una función de C.

7. Entregables y documentación

Sólo debe entregarse el archivo comprimido a través de moodle.

En la documentación además de todo lo indicado a lo largo de este guión organizado de forma correcta y coherente deberá añadirse una introducción que indique lo que se pretende realizar en la práctica y al final dos secciones de conclusiones, una de conclusiones técnicas y otras de conclusiones personales en la que se indicará lo que se ha aprendido y lo que no se tiene claro (si es que hay algo que no se tenga claro).

8. Consideraciones de diseño

Es importante realizar el diseño de tal forma que sea fácil añadir funcionalidad o algún comando IRC nuevo que no esté en el RFC. Esto es debido a que en una práctica se va a realizar una ampliación de protocolo con comunicación por voz y en otra se va a introducir la comunicación segura con SSL.