# Práctica 1
# Creación de un servidor IRC

**Eloy Anguiano**

**Oscar Delgado**

**Carlos García**

**Antonio Calleja**

**Fecha de entrega optativa**: 27 de febrero a las 9:00.
**Fecha de entrega obligatoria**: lunes 8 de mayo a las 9:00.

# 1 Objetivos

Uno de los objetivos de esta asignatura es desarrollar la capacidad de interpretar e implementar un protocolo según los RFCs correspondientes, y por tanto no se va a describir el protocolo IRC (aunque los profesores de prácticas ayudarán a cada alumno en las dudas que tengan acerca de los protocolos). Sin embargo, para facilitar la implementación se aporta información a continuación.

Esta práctica consiste en el diseño de un servidor IRC. El protocolo IRC (*Internet Relay Chat*) es un protocolo de nivel de aplicación ideado para el intercambio de mensajes de texto. Los detalles de este protocolo pueden verse en los RFCs ("Request For Comments") indicados en la página web: `http://metis.ii.uam.es/redes2`

En estos RFCs se describe cómo se deben realizar las comunicaciones entre los distintos agentes que intervienen en el intercambio de mensajes. Sin embargo se puede encontrar información adicional de varias formas. Una de ellas consiste en utilizar un cliente de GNU/linux llamado `xchat` (está disponible en los repositorios oficiales de casi todas las distribuciones). Se aconseja utilizar este cliente antes incluso de leer los RFCs para tener una idea somera del funcionamiento del protocolo IRC. Así mismo, una de las opciones de este programa permite guardar las comunicaciones realizadas a un fichero de texto plano legible por cualquier editor. Se recomienda tener abierta la ventana de "registro plano" del xchat para ver esta información en tiempo real y así entender el protocolo.

Existen también multitud de fuentes de información que puede buscar el alumno. Aun así es importante diferenciar entre las comunicaciones IRC y los comandos de usuario. Los comandos de usuario son *los que introduce el usuario en los clientes IRC (/join, /mode, etc) y que deben ser traducidos al protocolo IRC* aunque muchas veces haya una correspondencia directa entre el comando de usuario y el mensaje IRC correspondiente.

El objetivo de esta práctica es implementar un servidor para el protocolo IRC. Dicho servidor deberá ser capaz de atender las peticiones de múltiples clientes según lo descrito en el RFC 2812[1] (no se va a exigir la comunicación servidor-servidor contemplada en el RFC 2813[2]).

El servidor a implementar deberá ejecutarse en modo *daemon* y quedar configurado a la espera de conexiones TCP. Cuando un cliente se conecte, el servidor atenderá la conexión. Todas las conexiones se mantendrán abiertas mientras no falle el protocolo PING-PONG de IRC. Cada vez que se reciba una petición de un cliente se bloqueará la recepción de ese cliente y se lanzará un proceso o un hilo para procesar la petición. Una vez procesada la petición se desbloqueará la recepción de ese cliente. Evidentemente, aunque esté bloqueada la recepción de un cliente, el resto de las recepciones de clientes no estarán bloqueadas.

Es importante que el servidor sea capaz de comunicarse correctamente con otros clientes como XChat[3] o hexchat[4]. Para ello será necesario tratar no sólo los comandos IRC sino también los posibles retornos de

[1] RFC 2812: `http://tools.ietf.org/html/rfc2812`
[2] RFC 2813: `http://tools.ietf.org/html/rfc2813`
[3] XChat, multiplatform chat program: `http://xchat.org/`
[4] `https://hexchat.github.io/`

cada uno, ya sean respuestas (p. ej. `RPL_LIST`) o errores (p. ej. `ERR_NEEDMOREPARAMS`).

## 1.1 Ejecución en modo *daemon*

El servidor a desarollar en esta práctica deberá ejecutar en modo *daemon*, es decir, como proceso no interactivo corriendo en segundo plano (o *background*) y desacoplado de cualquier terminal. En esta sección se enumeran las acciones que un programa debe llevar a cabo para iniciar como *daemon*, junto con una lista de funciones `C` a utilizar. Es tarea del alumno consultar la documentación de las mismas para comprender su funcionamiento.

En particular, se debe desarrollar una función `daemonizar()` que reciba una cadena de caracteres que identifique al servicio y que devuelva cero en caso de ejecución correcta o un entero negativo en caso de error. Dicha función deberá:

1.– Crear un proceso hijo y terminar el proceso padre
2.– Crear una nueva sesión de tal forma que el proceso pase a ser el líder de sesión
3.– Cambiar la máscara de modo de ficheros para que sean accesibles a cualquiera
4.– Establecer el directorio raíz `/` como directorio de trabajo
5.– Cerrar todos los descriptores de fichero que pueda haber abiertos incluidos `stdin, stdout, stderr`
6.– Abrir el log del sistema para su uso posterior

Es importante notar que un proceso *daemon* no está asociado a una terminal, y por tanto no debe utilizar funciones como `printf()` ni leer o escribir de entrada o salida estándar. Para notificar mensajes, este tipo de procesos utiliza el log del sistema, que típicamente se puede encontrar en `/var/log/syslog`.

Funciones a utilizar:

```
chdir, close, fork, getdtablesize, open, openlog, setsid, syslog, umask
```

Consultar en el manual de la función `open` el significado de los flags `O_RDONLY` y `O_RDWR`.

## 1.2 Diseño del servidor

Un **servidor iterativo** procesa conexiones entrantes desde un puerto y las atiende en orden de llegada. La petición de cada cliente es procesada completamente antes de servir al siguiente cliente. Esto puede afectar al tiempo de respuesta si el servidor recibe un gran número de peticiones o si ha de desempeñar una gran carga de trabajo por petición.

Por ese motivo se desarrollará un sistema no iterativo. Este sistema debe admitir todas las conexiones y las mantenerlas abiertas mientras el protocolo PING-PONG no determine que el cliente ha desaparecido. El protocolo PING-PONG debe tener asociado un hilo que se ejecute cada 30s para realizar los envíos necesarios. Si en el siguiente envío no se ha recibido respuesta de un determinado servidor, el socket de este servidor debe ser cerrado. En lugar de en forma de hilo puede implementarse como proceso o como una atención a la señal “alarm”. En la documentación deberá justificarse la elección de diseño realizada.

Por otro lado, cada vez que se reciba una petición de uno de los clientes deberá lanzarse un hilo o proceso que la atienda. Evidentemente para no recibir más peticiones del mismo cliente antes de que se

procese la petición que está siendo atendida es importante bloquear las recepciones de ese cliente. Para orquestar las distintas peticiones utilizaremos la función `select`. Una llamada a `select()` bloquea el proceso hasta que sucede un evento sobre algún conjunto de descriptores de fichero. En nuestro caso nos permitirá escuchar de varios sockets a la vez para que, según vayan llegando peticiones, el servidor se active, las lea, y atienda al cliente correspondiente.

Los parámetros de `select` son los siguientes:

**nfds :** Valor entero que indica el número de descriptores que han de ser comprobados. Debe ser el mayor número de descriptor + 1.

**readfds :** Puntero a estructura de tipo `fd_set`. Determina los sockets que hay que comprobar para saber si están listos para lectura.

**writefds :** Puntero a estructura de tipo `fd_set`. Determina los sockets que hay que comprobar para saber si están listos para escritura.

**exceptfds :** Puntero a estructura de tipo `fd_set`. Determina los sockets que tienen pendientes condiciones excepcionales.

**timeout :** Puntero a estructura de tipo `timeval`. Con él se indica la cantidad de tiempo que se debe esperar hasta que haya un cambio en algún descriptor antes desbloquear la llamada y retornar.

Para manipular los conjuntos de descriptores de fichero han de emplearse las macros `FD_ZERO()`, `FD_SET()`, `FD_CLR()` y `FD_ISSET()`. Tanto en el manual de la función `select` como en el libro de Stevens o en la red pueden encontrarse numerosos ejemplos de su uso.

Otra posibilidad para organizar las peticiones de cada cliente es utilizar semáforos de forma adecuada. Tanto el método basado en select como el de los semáforos son válidos pero se deberá justificar de forma técnica la elección realizada.

Como implementación avanzada se sugiere el uso de un “pool” mínimo de hilos o procesos que puede crecer hasta un máximo o decrecer hasta un mínimo. Este máximo y mínimo pueden estar definidos en programa o por un archivo de configuración.

# 2 Creación de librerías propias

Con el fin de utilizar técnicas estándar de desarrollo, los alumnos deberán crear una librería con funciones de alto nivel para la conexión, desconexión y envío y recepción de mensajes de servidor en TCP. Las funciones deben ser de alto nivel, agrupando varias funciones de bajo nivel, resolviendo los errores en lo posible. Así mismo en esta librería se encontrará también la función de “daemonización”.

Se aconseja crear a la vez las funciones de TCP tanto para el servidor como para el cliente (es decir, tanto de escucha en un puerto, como de establecimiento de conexión a una determinada dirección). Si además se implementan las funciones de UDP, se facilitará y acelerará el desarrollo de futuras prácticas.

Esta librería será mantenida a lo largo de todas las prácticas. Todas las funciones de esta librería que sean susceptibles de ser usadas en otros programas deberán estar documentadas con su `man` correspondiente (no es necesario para las funciones de uso interno a la librería). Se aconseja el uso de doxygen para crear las páginas man.

Las prácticas deberán venir acompañadas del makefile correspondiente nombrado según la normativa correspondiente (ver en Moodle). Como mínimo, el makefile deberá crear la librería y el ejecutable correspondiente. Pero se aconseja que el makefile haga todas las tareas, incluida la compresión en el formato solicitado en la normativa. También es importante que si el makefile se ejecuta sin parámetros se creen los evaluables de la prácticas.

# 3 A tener en cuenta

Es importante que el servidor admita parámetros en ejecución al menos para poder activar la ejecución con SSL en la tercera práctica. En este caso deberá poder ejecutarse con la bandera -s. Cualquier otra opción de ejecución deberá documentarse en el código al igual que aparecer en la terminal si se usa la bandera -h (en cuyo caso se acabará la ejecución y no se ejecutará el demonio). También se pueden usar las banderas -ssl y -help usando la función getopt_long.

Para el uso de banderas es interesante aprender el uso de la función getopt y getopt_long. En moodle hay enlaces para aprender a usar esta función.

# 4 Documentación y entrega

Junto con el código del servidor debe entregarse una **memoria** que, brevemente, incluya al menos la siguiente información:

- Introducción: una descripción de lo que se pretende realizar en la práctica
- 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.)
- Funcionalidad IRC: a grandes rasgos, qué funciones del protocolo se han implementado
- Conclusiones técnicas: temas concretos de la asignatura que se han aprendido al realizar la práctica
- Conclusiones personales: a qué se ha dedicado más esfuerzo, comentarios generales

Se considera parte de la documentación también las páginas man arriba indicadas.

**La entrega se realizará en un único fichero a través de Moodle. Se recuerda que es obligatorio seguir el formato descrito en la normativa de entrega. Aquellas prácticas que no compilen correctamente con `make` o que de manera constante den fallos al ejecutar la funcionalidad básica se considerarán como no entregadas. Por ello es aconsejable probar con el autocorrector justo antes del envío.**

# 5 Evaluación

El diseño del servidor queda completamente a elección del alumno, pero se evaluará en función de las siguientes cualidades:

- Estabilidad: el servidor ejecuta correctamente y no se cierra de manera inesperada (fallos de segmentación, etc.)

- Modularidad: el programa está desacoplado en módulos que realizan funciones concretas (sockets, conexión/desconexión, etc.)
- Funcionalidad IRC: mensajes soportados, retorno correcto de respuestas y errores, gestión usuarios, canales y modos, etc.
- Diseño del servidor.
- Calidad de la memoria: sin necesidad de ser extensa, contiene los apartados enumerados anteriormente y describe concretamente las decisiones de diseño.

Así mismo, se utilizará un corrector automático que se proporciona a los alumnos *con toda su funcionalidad*. Este corrector aportará la **nota de ejecución**, que se encontrará entre **0 y 6**. Así mismo el profesor podrá utilizar XChat o hexchat como cliente de referencia para probar el servidor a la hora de evaluar la ejecución de la práctica con el fin de comprobar que esta práctica no ha sido diseñada exclusivamente para pasar las pruebas.

Los cuatro puntos restantes se reparten, de foma orientativa en: **2 puntos documentación y páginas man, 2 puntos código** (calidad, modularidad, coherencia en la nomenclatura, eficiencia ...).

*Importante: Si no se consigue un 3 en ejecución no se podrá sacar más de un 4.9 en la práctica, independientemente de la calidad de la documentación y el código.*