Práctica 2
Creación de un cliente IRC

Eloy Anguiano

Rodrigo Castro

Ignacio Fernández

Daniel Hernández

Fecha de inicio: 5 de marzo de 2014 (Grupos de los miércoles). 7 de marzo de 2014 (Grupos de los viernes)
Fecha de entrega: 2 de abril de 2014 (Grupos de los miércoles). 4 de abril de 2014 (Grupos de los viernes)

Cualquier archivo fuente que sea modificado por el usuario deberá cambiar de nombre y usar la nomenclatura propia del usuario

1. Diseño de un cliente IRC

Esta práctica consiste en el diseño de un clienter 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] y en el RFC 2812[2]). Este protocolo utiliza conexiones *TCP*[3] para realizar las comunicaciones entre los distintos agentes que intervienen en el intercambio de mensajes.

De la práctica 1 ya es conocido el uso de este protocolo y por tanto no se incidirá en su descripción.

Para poder implementar adecuadamente un cliente se proporciona un interfaz gráfico en C en el .tgz disponible en moodle. Se adjuntan tanto los objetos como los fuentes. Los objetos sirven para compilar directamente con ellos y los fuentes están disponibles para el caso en que se desee modificar su funcionamiento. En éste interfaz, parte de los comandos IRC se pueden realizar a través del interfaz y otros deberán implementarse como comandos en el texto introducido. Por ese motivo será necesario parsear adecuadamente este texto para interpretar los comandos. Se recomienda el uso tradicional de comandos[4].

Si se desea modificar el interfaz gráfico es necesario disponer de los paquetes de desarrollo de gnome (GTK)[5].

1.1. Funcionalidad aportada

Se aporta un makefile extremadamente simple para que el alumno pueda ver las opciones de compilación fácilmente.

Con el fin de simplificar la interacción del programador con el interfaz gráfico se han encapsulado dichas llamadas en funciones fácilmente utilizables y que se describen a continuación:

void publicText(char *user, char *text): Presenta en la ventana de chat el texto recibido de un determinado usuario enviado como mensaje público.

void privateText(char *user, char *text): Presenta en la ventana de chat el texto recibido de un determinado usuario enviado como mensaje privado.

void errorText(char *errormessage): Presenta en la ventana de chat un texto de error.

void messageText(char *message): Presenta en la ventana de chat los mensajes que desee el programador.

[1] RFC 1459: http://tools.ietf.org/html/rfc1459
[2] RFC 2812: http://tools.ietf.org/html/rfc2812
[3] *Transmission Control Protocol*, RFC 793: http://tools.ietf.org/html/rfc793
[4] http://personales.mundivia.es/papi/comirc.html#1
[5] libgtk2.0-dev o superior

char * getApodo(void): Obtiene el apodo definido por el usuario y devuelve un apuntador a éste. No se debe liberar el apuntador a la cadena devuelta.

char * getNombre(void): Obtiene el nombre definido por el usuario y devuelve un apuntador a éste. No se debe liberar el apuntador a la cadena devuelta.

char * getNombreReal(void): Obtiene el nombre real definido por el usuario y devuelve un apuntador a éste. No se debe liberar el apuntador a la cadena devuelta.

char * getServidor(void): Obtiene la url del servidor definido por el usuario y devuelve un apuntador a ésta. No se debe liberar el apuntador a la cadena devuelta.

int getPuerto(void): Obtiene el número de puerto definido por el usuario.

void setTopicProtect (gboolean state): Activa (TRUE) o desactiva (FALSE) la protección de tópico.

void setExternMsg (gboolean state): Activa (TRUE) o desactiva (FALSE) los mensajes externos.

void setSecret (gboolean state): Activa (TRUE) o desactiva (FALSE) el secreto del canal.

void setGuests (gboolean state): Activa (TRUE) o desactiva (FALSE) la admisión de invitados.

void setPrivate (gboolean state): Activa (TRUE) o desactiva (FALSE) la privacidad del canal.

void setModerated (gboolean state): Activa (TRUE) o desactiva (FALSE) la moderación del canal.

void errorWindow(char *msg): Permite presentar una ventana de diálogo de error con el mensaje deseado.

Es importante tener en cuenta que si se utilizan múltiples hilos o procesos, el acceso al interfaz gráfico deberá ser ordenado.

1.2. Implementación de funcionalidad

Deberán implementarse las funciones presentadas en `chat_funcs.c`. Si no se implementan es necesario dejar la implementación con la que se aportan originalmente. Si se desea implementar una mayor funcionalidad pueden añadirse en este u otros ficheros. Es necesario cambiar el nombre de este fuente al propio del grupo y por tanto es divisible en múltiples archivos fuente.

Las funciones a implementar son las siguientes:

void connectClient(void): Función que es llamada cuando se pulsa el botón de conexión.

void disconnectClient(void): Función que es llamada cuando se pulsa el botón de desconexión.

void topicProtect(gboolean state): Función que es llamada cuando se pulsa el botón de protección de tópico. En el parámetro se recibe el valor del botón nada más ser pulsado.

void externMsg(gboolean state): Función que es llamada cuando se pulsa el botón de mensajes externos. En el parámetro se recibe el valor del botón nada más ser pulsado.

void secret(gboolean state): Función que es llamada cuando se pulsa el botón de secreto. En el parámetro se recibe el valor del botón nada más ser pulsado.

void guests(gboolean state): Función que es llamada cuando se pulsa el botón de invitado. En el parámetro se recibe el valor del botón nada más ser pulsado.

void privated(gboolean state): Función que es llamada cuando se pulsa el botón de privado. En el parámetro se recibe el valor del botón nada más ser pulsado.

void moderated(gboolean state): Función que es llamada cuando se pulsa el botón de moderación. En el parámetro se recibe el valor del botón nada más ser pulsado.

void newText (const char *msg): Función que es llamada cuando se introduce un nuevo texto en la entrada de mensajes y se pulsa la tecla “enter”. En el parámetro se recibe el texto introducido.

2. Modificación multimedia de IRC

Se propone una modificación del protocolo IRC que consiste en la comunicación privada usando sonido. Para ello será necesario utilizar el protocolo RTP: RFC 3550[6]. Para que el alumno no tenga que realizar las funciones de sonido se aporta un objeto con la funcionalidad de sonido. Como la comunicación se realizará vía UDP será necesario encapsular nuevas funciones encapsuladas para la comunicación vía UDP.

Para elegir un puerto aleatorio que esté libre se deberá utilizar en el `bind` como número de puerto el 0 y después llamar a la función `getsockname` para obtener la información necesaria acerca de nuestra IP y el puerto que ha sido abierto.

Al protocolo de IRC se van a añadir los comandos \pcall, \paccept y \pclose. Desde el punto de vista de estos usuarios estas funciones sólo llevan un parámetro: el nick con el que se quiere realizar la acción. Desde el punto de vista de la comunicación todos ellos enviarán internamente un PRIVMSG en el que se indicará que es una llamada de voz. En el caso de \pcall y \paccept deberá incluir también el nick del usuario y su IP y el puerto donde espera recibir la comunicación. En el caso de \pclose se cerrará la conexión de voz y se enviará un PRIVMSG indicando que se ha cerrado la conexión. En ningún caso deberán presentarse estos mensajes al usuario.

Si se desea modificar la funcionalidad de sonido es necesario disponer del paquete de PulseAudio[7]. En realidad estas son funciones síncronas. Si se desean funciones asíncronas es necesario utilizar otras funciones de PulseAudio. Es más, a través de un valor para `pkg-config` distinto se puede integrar con el *main loop* de GTK. Aunque no es necesaria esta implementación, se tendrá en cuenta positivamente si se realiza.

2.1. Funcionalidad aportada

En el fuente `sound.c` y en su correspondiente `sound.h` se crean y prototipan respectivamente las siguientes funciones:

int openRecord(char *identificacion): Inicia la captura. Devuelve un error estándar de PulseAudio o 0 si no ha habido error. El parámetro de identificación es una cadena de caracteres cualquiera. Se suele utilizar el nombre del programa que usa esta función.

int openPlay(char *identificacion): Inicia la reproducción. Devuelve un error estándar de PulseAudio o 0 si no ha habido error. El parámetro de identificación es una cadena de caracteres cualquiera. Se suele utilizar el nombre del programa que usa esta función.

void closeRecord(void): Cierra la captura.

void closePlay(void): Cierra la reproducción.

int recordSound(char * buf, int size): Almacena una parte de la grabación en el buffer. Se aconseja el uso de buffers con tamaños relacionados con los paquetes RTP. Devuelve un error estándar de PulseAudio o 0 si no ha habido error. Los parámetros son el buffer en el que se almacenan los datos y el tamaño en bytes a leer. El buffer debe tener espacio reservado suficiente.

int playSound(char * buf, int size): Reproduce la parte de la grabación almacenada en el buffer. Se aconseja el uso de buffers con tamaños relacionados con los paquetes RTP. Devuelve un error estándar

[6] RFC 3550: http://tools.ietf.org/html/rfc3550

[7] libpulse-dev y http://freedesktop.org/software/pulseaudio/doxygen/glib-mainloop.html

de PulseAudio o 0 si no ha habido error. Los parámetros son el buffer en el que se almacenan los datos y el tamaño en bytes a leer.

int sampleFormat(enum pa_sample_format format, int nChanels): Tipo de muestreo de la señal. Devuelve el valor del "payload" de RTP correspondiente. El parámetro de entrada es un valor de los presentados en la tabla inferior. Si no se usa, el valor por defecto es PA_SAMPLE_S16BE. Así mismo, el segundo valor es el número de canales que se quiere muestrear. Este parámetro puede tomar el valor 1 en todos los casos y también 2 en el caso de PA_SAMPLE_S16BE. Las funciones internas de PulseAudio permiten otro tipo de muestreos pero ninguno compatible con un payload RTP. Devuelve -1 en caso de formato incorrecto.

Valor	Significado
PA_SAMPLE_ALAW	8 Bit a-Law 8 kHz
PA_SAMPLE_ULAW	8 Bit mu-Law 8 kHz
PA_SAMPLE_S16BE	16 Bit PCM con signo, big endian, 44.1 kHz

3. Estructura de desarrollo

Deberán seguirse usando los mismos directorios y la nomenclatura. De esta forma la librería seguirá siendo única y el makefile también. Por ese motivo en el entregable se incluirán ambas prácticas.

3.1. 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.
- 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.

3.2. 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.

4. 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).

5. Consideraciones de diseño

Es importante realizar el diseño de tal forma que sea fácil añadir funcionalidad. Esto es debido a que en una práctica se va a realizar la comunicación segura con SSL.