**Memoria. Parte cliente.**

Para el desarrollo del cliente tratamos de acercarnos lo máximo posible al comportamiento de Hexchat.

**Estructura y diseño básicos:**

Para implementar el cliente y que este tuviera un funcionamiento fluido utilizamos 2 hilos principales. El primero se encarga de mantener activa la interfaz del usuario, así como de enviar y procesar los comandos introducidos por él. El segundo hilo está a la espera de mensajes por parte del servidor y se encarga de recibirlos, procesarlos y actualizar la interfaz con la información recibida.

A estos dos hilos principales se suma uno más encargado de enviar un mensaje de ping al servidor cada 20 segundos si el usuario no ha enviado ningún mensaje. Este hilo, al que llamamos tkeep\_alive se encarga de mantener activa la conexión con el servidor para no ser desconectados por timeout.

Se lanzan más hilos en los casos de envío o recepción de ficheros y en el caso de chat de audio.

A partir de la biblioteca de funciones tcp que teníamos de la práctica anterior creamos una nueva llamada client\_tools con 2 objetivos principales. Primero controlar el acceso al socket de los hilos que compiten por él, de esta forma haciendo transparente el uso de hilos y posibles problemas de concurrencia en el resto del proyecto. Segundo, facilitar la futura implementación de ssl.

Al comienzo del desarrollo observamos una distinción entre los comandos introducidos por el usuario y los comandos recibidos del servidor. De esta manera creamos dos ficheros fuente: ucommands.c (de user commands) para los comandos de usuario, y rcommands.c (de reply commands) para los comandos recibidos del servidor.

**Comandos implementados**

Se han implementado los solo los comandos más importantes y esenciales para el funcionamiento básico del cliente. Consultando includes/ucommands.h y includes/rcommands.h se pueden ver los comandos implementados.

**Envío de ficheros y audiochat**

Para la implementación del envío de ficheros y llamadas tuvimos que crear nuevos mensajes. Estos mensajes se parsean como PRIVMSG, pero tienen como primer carácter /001.

La función que cumple cada uno de estos mensajes especiales se detalla en la documentación, en resumen, son:

FSEND. Se envía para ofrecer el envío de un fichero

REJECTED. Se envía al rechazar la oferta de transmisión de un archivo

AUDCHAT. Se envía para iniciar una llamada de voz.

AUDREPLY. Se envía cuando se recibe una llamada de voz para indicar ip y puerto.

ENDAUD. Se envía para terminar la llamada.

Tanto para el envío de ficheros como para el chat de audio se establece una conexión directa entre los clientes. Para ello se debe proporcionar la dirección IP y puerto propios donde queremos abrir esta nueva conexión. Para conseguir nuestra dirección IP se utiliza la función get\_ip() definida en cliente\_IRC.c. Esta función devuelve la dirección IP local de la interfaz prioritaria (la devuelta por hostname -I). Sin embargo, este comportamiento se puede modificar. Si se desea usar otra IP basta cambiar la variable global mi\_ip definida en cliente\_IRC.c (información más detallada en el propio fichero).

Nota: El cliente solo soporta 1 llamada al mismo tiempo. Si se intentan abrir varias, es probable que fallen todas.

**Problemas y bugs**

Hemos tenido algunos problemas con la librería del cliente y gtk pero creemos que se debe a que lo desarrollamos en nuestros ordenadores que tienen Linux Mint 18.1 Serena. No hemos ejecutado nuestro cliente muchas veces en los ordenadores de los laboratorios, pero cuando lo hemos hecho no ha habido problemas. En la sección de bugs detallaremos todos los bugs que nos encontramos y no pudimos reportar en Bugzilla.

A pesar de que tratamos de liberar toda la memoria que reservamos, en el envío de ficheros y llamadas es posible que haya perdidas y memoria no liberada. Decimos que es posible porque ha sido imposible utilizar valgrind para detectar estas pérdidas (y solucionarlas) ya que gtk tiene muchísimas pérdidas de memoria en muchos contextos y buscar entre todas estas las que podíamos haber generado nosotros era imposible. De cualquier forma, si hay pérdidas son menores e insignificantes al lado de las de la librería gráfica.

Por último, hay unos Warnings por usar la bandera m en la función sscanf cuando parsea los mensajes especiales. Esta bandera no está permitida en ISO C pero nosotros la usamos por comodidad y nos aseguramos de que no da ningún problema.

BUGS