26-3-2016

Aitor Portillo - Alejandro Celada Guerrero

UAH

Laboratorio de redes, sistemas y servicios

Práctica 1.2: Socket no bloqueantes en modo cliente-servidor y P2P

# Cliente-servidor

## Descripción del objetivo

El objetivo de esta parte de la práctica es crear un chat que sigue la infraestructura cliente-servidor, utilizando el lenguaje de programación “C”. Y de esta forma ver como es necesario la utilización de Sockets no bloqueantes mediante el Select.

## Problemas encontrados y dificultades

En cuantoa las dificultades encontradas, éstas están relacionadas mayormente con el uso del propio lenguaje de programación C. Puntos a tener en cuenta y sobre los que se han tomado precauciones a la hora de implementar el código son:

* Cerrar convenientemente todos sockets para que puedan reutilizarse.
* Detección de posibles errores en el envío y la recepción de mensajes y su correspondiente alerta al usuario.
* Conversión correcta de cadenas a enteros y viceversa para una correcta visualización y uso de direcciones y puertos.
* Detección del uso correcto de la aplicación, acotando los parámetros de entrada que el usuario puede introducir.
* ¿Cómo saber a qué cliente corresponde cada descriptor? Lo hemos solucionado creando una lista enlaza, en la que cada elemento es una estructura creada según nuestras necesidades. Lo que nos ha dado bastante potencial, porque en todo momento sabemos todos los datos que queramos cualquier cliente.
* ¿Nombre de los usuarios? ¿Distintos tipos de paquetes a mandar? Era bastante complicado hacerlo mandando solo un string, por lo que hemos creado un “protocolo casero”, en el que mandamos una estructura que nos ayuda a diferenciar entre distintos tipos de mensajes.
* Otro problema importante, ha sido “limpiar” las variables para que no imprimiésemos suciedad.
* La complejidad de que todo funcione correctamente, habiendo metido tantas ampliaciones, se hace un poco caótico tener tantas líneas de código.

## Explicación breve sobre las ampliaciones interesantes

La mayoría de las ampliaciones añadidas han supuesto diversos problemas a resolver a cambio de una mayor funcionalidad y robustez del programa. Entre ellos tenemos:

* Desde el servidor podemos movernos entre distintos modos de ejecución.
* Copia de seguridad de todo lo que se escribe por el chat y de los mensajes de control, a la cual solo se tiene acceso desde el servidor.
* Lista enlazada dinámica con la que controlamos todos los datos de los clientes, y podemos sacar estadísticos de estos.
* Cada vez que un cliente recibe un mensaje, en todo momento sabe el “Alias” de quien lo ha escrito, y la hora a la que lo escribió, pero en ningún momento puede saber desde que IP y puerto se ha escrito.
* Hacer una “protocolo casero” donde mediante una estructura podemos diferenciar entre distinto mensajes y en función de cada uno los clientes actuaran de distinta manera.
* Poder mandarse archivos de cualquier formato y tamaño, al servidor, y que este lo reenvie a todos los cliente conectados. Sin que haya bloqueo, pudiendo seguir usando el chat, sin necesidad de tener que abrir otra conexión.
* Asignación de colores a cada cliente. Por lo que si dos clientes no quieren identificarse y su Alias es “Anonimo” así se podrá identificar mejor. Aunque por comodidad solo hemos implementado cinco colores, asique si estos se acaban, se repiten.
* Tenemos la posibilidad de banear un cliente desde el servidor.

## Enfoque de servidor.c y del cliente.c

Vamos a dar un enfoque de cómo y qué es lo que hacen ambos programas.

En el servidor hace uso siempre de tres descriptores, uno para guardar lo que pasa mediante el servidor (“file\_copy”) en un fichero a modo de copia de seguridad. El otro es (“fd”), que es el que está asociado a la IP y puerto del servidor, además de que está incluido para que el select lo teste. Mediante el recibimos las peticiones de conexión. Y un tercero, que no hemos creado, que es característico de Unix es STDIN\_FILENO, que lo incluimos también par que lo teste el select. Tanto “fd” como “file\_copy” están incluidos en la lista enlaza dinámicamente.

El select puede activarse por tres causas:

* Petición de conexión. (fd)
* Recibimos mensaje que hay que mandar a todos los usuarios conectados.
* Recibimos datos del teclado. (STDIN\_FILENO)

Para el primer caso, cuando se recibe una petición de conexión se le hace el accept, de donde obtenemos el descriptor por el cual le identificaremos, la estructura sockaddr, que nos dará bastante información, y añadimos el nuevo descriptor para que el select lo teste. Toda esta información del cliente lo guardamos en la lista enlazada dinámicamente.

Para el segundo caso, recibimos un mensaje de algunos de los descriptores creados. Más tarde entraremos en detalle con los tipos de mensajes. Por el momento, es un mensaje normal, así que será reenviado a todos los usuarios conectados.

Gracias a las funciones que hemos creado con la lista enlazada dinámicamente, esta operación la hace una función, que recorre la lista enviándole dicho mensaje a todos los descriptores conectados, excepto a quien lo mando. Y además incluye ese mensaje que incluye el la hora, el alias, y los datos es introducido en el fichero que actúa como copia de seguridad.

Para el tercer caso, podemos recibir mensajes del teclado, donde se pueden diferencias tres modos dependiendo de que se escriba:

* “/help” se limpia la pantalla y se muestra un pequeña ayuda de las distintas opciones que hay en el programa.
* “/database” se muestra todos la lista enlaza dinámicamente. Donde aparece por cada cliente su Alias, IP, Puerto, descriptor asignado, y numero de mensajes.
* “/chat” en esta opción podemos ver todo lo que se están escribiendo los clientes, e incluso si conviene intervenir con el alias de “Admin”, y siempre en color rojo.
* “/banning ip:puerto”, desde el cual podemos hacer un baneo de un cliente.

En el cliente los parámetros son la IP, puerto, y nombre. Siendo este el nombre que vas a mantener durante esa sesión en el chat. Esta identificación es voluntaria, ya que si no se pone nada, el alias adoptado será de “Anonimo”.

El cliente crea un socket con el servidor, y mediante este se conectara al servidor. Por otro lado también hace uso de STDIN\_FILENO. Ambos los incluimos para que los teste el select.

Vamos a explicar los distintos tipos de mensaje para poder seguir con la explicación del cliente.

Gracias a la estructura que enviamos, en función de los parámetros que alberga, podemos diferencia entre cuatro tipos de mensajes:

* Mensajes chat. Donde la información es lo que ha escrito el usuario, a que hora y el alias de este.
* Mensajes de ficheros. Para el envió de fichero. En él la información son los propios datos a enviar, la hora, el alias de quien lo envía, y lo más importante el tamaño del segmento de fichero que lleva.

Y los otros dos mensajes, los podemos calificar como mensajes de control:

* Mensajes de apertura: En el avisamos de que vamos a iniciar la transferencia de un fichero. Y mandamos el nombre de esta, para que el cliente destino abra un socket para poner clonar el fichero.
* Mensaje de cierre: En el avisamos de que hemos acabado de mandar todos los fragmentos del fichero, y que el cliente destino cierre el socket.3

Tanto el mensaje de apertura como el de cierre, solo se mandan una vez durante la transferencia, por lo que si un cliente se conecta unas vez que se ha iniciado la transferencia de un archivo, los recibirá, pero los desechara.

Por lo que cuando el select del cliente se activa el descriptor fd, sabe que le ha llegado antes, mira los flag de la estructura recibida. Y en función de los cuatro tipos de mensajes que puede recibir actúa.

Por otro lado, si se activa STDIN\_FILENO, tenemos dos opciones. O es transferir un archivo o mandar un mensaje normal. Para transferir un archivo tenemos que poner “/ftp ruta”.

Cabe destacar que para mandar el archivo lo que hacemos es abrirlo mediante un socket, y según vamos sacando fragmentos del tamaño que limita el campo “datos” de la estructura (cabe hacer incapie que esa variable de char\* es la misma para mardar datos de un fragmento de fichero que del texto que se lee por pantalla), se envían al cliente.

Esta operación se realiza en un “for”, hasta que el fichero se acaba de leer y mandar al servidor por completo. Por lo que durante esos instante ese cliente sí que estará bloqueado. Pero no el resto de clientes.