Práctica 9. Desarrollo de un chat multiusuario

1. Introducción

Para esta práctica haremos uso de los conceptos vistos en los dos últimos temas: los *threads* y el manejo de red con el API facilitado por la librería Boost Asio de C++.

Ya hemos aprendido que, para poder manejar distintos hilos de ejecución en concurrencia, hemos de hacer uso de la librería de hilos o threads, que nos ofrece C++ (y, en general, casi cualquier lenguaje de programación). Estos hilos nos permiten que, una vez se lance nuestro *hilo* principal, o *main*, podemos desde éste lanzar otros hilos (proceso que se conoce como *fork* (pensemos en la forma que tiene un tenedor: un conducto principal del que parten distintos conductos...), a los que asignaremos una tarea de ejecución (o *task*).

Por otro lado, en el último tema hemos visto los conceptos básicos para el manejo de red desde nuestro programa. De este modo, podemos, entre otras cosas, establecer procesos de comunicación entre distintas entidades en red, que se denominan *endpoints*, y que están identificados por su dirección IP junto con el puerto que se use para dicha comunicación. Para poder realizar esa comunicación, debemos manejar el concepto abstracto de *socket*, que es la herramienta principal que se usa cuando tratamos de gestionar procesos en red.

2. Compilación de la librería *boost*

Junto a este enunciado se entrega el código completo de un ejemplo de cliente-servidor usando la librería Boost Asio. Ya hemos visto en clase cómo se instala esta librería, pero —quizás—, para una correcta compilación, puede que debas tratar algunos aspectos del path de acceso a esta librería, ya que normalmente no se instala en los directorios que emplea

normalmente el sistema Linux (e.g. `/usr/lib` o `/usr/local/lib`).

Siendo así, lo primero que debemos hacer es intentar compilar los códigos que se entregan, y ver si esta compilación nos vierte algún error. Para ello, ejecutaremos los siguientes comandos:

```
g++ tcp-server.cpp -o tcp-server -lboost_system
```

```
g++ tcp-client.cpp -o tcp-client -lboost_system
```

Si no nos aparece ningún error de compilación, perfecto, podemos proseguir con la ejecución como normalmente. En otro caso, debemos solucionar el error que nos aparezca. Por ejemplo, puede ocurrir que nos aparezca el siguiente error de compilación:

```
cannot find -lboost_system
```

En este caso, debemos indagar para saber dónde se ha instalado la librería `boost_system`. Para ello, lo buscaremos; el fichero que contiene esta librería se llamará, concretamente, `libboost_system.so`. Podemos buscar tal fichero usando el la herramienta `locate` o `whereis`, entre otras.

Una vez encontrado, que lo más seguro es que se haya instalado en la siguiente ruta:
`/usr/lib/x86_64-linux-gnu`
debemos indicárselo al gestor de librerías dinámicas (`LD`), incluyéndolo en su `LD_LIBRARY_PATH`. Para ello, lanzamos el siguiente comando:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/x86_64-linux-gnu
```

Cuidado con no sobreescribir ese path. Con el anterior comando se añade la ruta que queremos, incluyendo lo que ya tiene, que es lo ideal.

También añadimos este comando a nuestro `.bashrc`, para que cada vez que abra el `Terminal`, LD tenga ya incluido dicho path. Para ello, ejecutamos el siguiente comando:

```
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/x86_64-linux-gnu'
>> /home/julio/.bashrc
```

Finalmente, actualizamos `LD` para que *lea* el nuevo path. Ejecutamos lo siguiente:

```
sudo ldconfig
```

Una vez seguidos estos pasos, la compilación no debería verter ningún error.

Ejercicio básico (8 puntos)

Se pide que, partiendo de los códigos facilitados, y haciendo uso de *threads*, desarrollemos un chat de comunicación que permita la ejecución de distintos clientes. Para ello, el servidor debe permitir recibir peticiones de diferentes clientes, que quedarán registrados en una cola, y que atenderá mediante un *thread* independiente para cada uno. Los clientes se lanzarán desde distintos terminales para probar su correcta ejecución.

Ejercicio avanzado (2 puntos)

Además de permitir que cada cliente envíe un mensaje estático (escrito en el código), en esta parte avanzada se pide que, cada cliente pueda escribir en el terminal para enviar un mensaje de forma interactiva; esto es, permitiendo al usuario escribir usando el teclado para enviar mensajes, que irá recepcionando el servidor y reenviando a cada cliente registrado en el sistema.