13 Secuenciación UDP

Elaborado por: Ukranio Coronilla

Añadir un **timeout** para evitar un bloqueo permanente en la llamada a `recvfrom` es también una buena idea para resolver una situación de perdida de paquetes UDP. De este modo si el paquete se ha perdido, podemos tomar la decisión de volver a reenviarlo n veces hasta que recibamos una respuesta (siempre y cuando el servidor se encuentre activo).

Ejercicio 1: Elabore un cliente que en el caso de que no se reciba una respuesta, envíe hasta 7 solicitudes al servidor de la práctica 9 "*Servidor de dominio Internet tipo UDP*". En el caso de que no responda después de 7 solicitudes, el cliente deberá concluir que el servidor no se encuentra activo en este momento y que debe reintentar más tarde. ¿Por qué 7? porque hay 7 días de la semana, 7 colores en el arcoíris, 7 vidas tiene un gato, hay 7 maravillas en el mundo, 7 pecados capitales, etc ... ☺

A veces no solo es suficiente un temporizador, porque no sabremos si el mensaje se perdió al viajar hacia el servidor o se perdió al regresar al cliente. En algunas aplicaciones esto es crítico, por ejemplo cuando se tiene un servidor de transacciones y se transfiere dinero de la cuenta B a la cuenta C. En este caso si el mensaje que se ha perdido es el que regresa al cliente, entonces la segunda petición enviada por el cliente hará un segundo intento con la misma transacción lo cual de llevarse a cabo, provocará efectos indeseables.

Para resolver este importante problema agregaremos un número de intento al mensaje, de modo que el servidor pueda darse cuenta cuando la solicitud se ha repetido y entonces pueda descartarla cuando sea necesario.

Para hacer más segura la comunicación entre cliente y servidor, implementaremos un nuevo tipo de mensaje cuyo formato es el siguiente:

```
struct mensaje {
	uint32_t secuencia; //Numero de secuencia
	uint32_t solicitud[2]; //pareja de números enviados por el cliente para su suma
	uint32_t respuesta;   //respuesta enviada por el servidor
};
```

Para enviar este tipo de mensaje en el método `envia` solo es necesario hacer un cast de la variable apuntador `struct mensaje` a char *.

Ejercicio 2: Elabore un cliente y un servidor, donde el cliente envía n (donde n se pasa como parámetro en la línea de comandos) solicitudes distintas a un servidor, es decir que cada solicitud

consistirá de dos números aleatorios enteros. Cada solicitud (por ejemplo 93 +46) va con un numero de secuencia que inicia en 1, si alguna solicitud se pierde debe reenviarse (en este caso 93 + 46) y aumentar en uno el número de secuencia.

Si se han hecho 7 solicitudes y el servidor no responde entonces el cliente debe imprimir que el servidor no está disponible y terminar su ejecución.

El cliente debe recibir como parámetro el número de solicitudes distintas que se van a hacer al servidor, así como la IP y el puerto del servidor.

En cualquier caso el cliente deberá imprimir al terminar:

- El número de solicitudes distintas hechas al servidor.
- El número de solicitudes totales hechas al servidor (incluyendo los reenvíos).
- El número de solicitudes respondidas por el servidor.
- Porcentaje de datagramas perdidos

Ejercicio 3: Modifique el servidor para que reciba como parámetro un número entre 1 y 100 el cual será el porcentaje de datagramas que no serán respondidos. De este modo simularemos de manera “controlada” la perdida de mensajes para comprobar el ejercicio 2.