Haciendo confiable el protocolo Solicitud-Respuesta en UDP

Elaborado por: Ukranio Coronilla

Tal y como se menciona en la página 189 del libro de Coulouris, los timeouts se utilizan para compensar la pérdida de mensajes mediante la solicitud repetida de mensajes de solicitud. Vamos a hacer la implementación de los timeouts en nuestra clase Solicitud, y para ello también tendremos que agregar un nuevo método a la clase `SocketDatagrama`.

Sabemos que una llamada a `recvfrom` se va a bloquear hasta que no llegue un mensaje que generalmente se ha solicitado mediante una llamada `sendto` previa. En ciertas aplicaciones conviene sacar al proceso del bloqueo después de cierto tiempo transcurrido (timeout) donde no se ha recibido respuesta.

Para activar el temporizador se utiliza la función UNIX `setsockopt()` la cual permite manipular las opciones del socket. En este caso el tiempo se ajusta inicializando una estructura del tipo `timeval`, la cual contiene el miembro `tv_sec` para inicializar los segundos y `tv_usec` para los microsegundos. Por ejemplo para inicializar un tiempo de 2.5 segundos tendriamos:

```
#include <sys/time.h>
struct timeval timeout;
timeout.tv_sec = 2;
timeout.tv_usec = 500000;
```

Hecha la inicialización de la variable `struct timeval` podemos utilizarla en la función setsockopt como sigue (véase man 7 socket):

```
setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout));
```

La variable global ***errno*** tomará el valor de la macro `EWOULDBLOCK` si el temporizador se ha activado. De este modo después de ejecutar la función `recvfrom` podremos imprimir un mensaje de que han pasado 2.5 segundos y no llegó ningún mensaje como sigue:

```
#include <errno.h>
int n;
n = recvfrom(... )
if (n < 0) {
      if (errno == EWOULDBLOCK)
            fprintf(stderr, "Tiempo para recepción transcurrido\n");
      else
            fprintf(stderr, "Error en recvfrom\n");
}
```

Ejercicio 1: Añada una variable privada `struct timeval` y el siguiente método a la clase SocketDatagrama:

```
int recibeTimeout(PaqueteDatagrama & p, time_t segundos, suseconds_t microsegundos);
```

El cual realiza la misma función que el método recibe, pero además inicializa un temporizador con el valor de segundos y microsegundos que recibe como parámetros. Además debe devolver -1 en caso de que se haya agotado el timeout, así como imprimir el mensaje "`Tiempo de recepción transcurrido`".

Modifique ahora el método de la clase `doOperation` para que el cliente pueda reenviar la misma solicitud hasta 7 veces en caso de pérdida de mensajes. ¿Por qué 7 veces? Porque la semana tiene 7 días, hay 7 notas musicales, 7 pecados capitales, 7 colores del arcoíris, 7 vidas tiene el gato...

Si en 7 veces no se logra recibir una respuesta entonces el programa termina e imprime que el servidor no está disponible e intente más tarde.

Para probar nuestra clase vamos a elaborar una pequeña aplicación cliente servidor que simula un cajero automático. El servidor mantiene una nano base de datos que solo almacena la cuenta de un cliente en la variable entera nbd y cuyo valor inicial es cero. Este servidor ofrece dos tipos de operaciones, la operación lectura con `operationId` = 1 y la operación escritura con `operationId` = 2. La operación lectura provoca que el servidor devuelva el valor que tiene almacenada la variable nbd. La operación escritura provoca que el servidor actualice el valor de nbd con el valor que el cliente le envíe y devuelva el valor actualizado de nbd hacia el cliente.

Por otro lado tendremos un cliente que recibe en la línea de comandos un entero n, y va a ejecutar ese número de depósitos de $1 sobre su cuenta en el servidor. El cliente entonces deberá ejecutar un ciclo de n veces mediante operaciones consecutivas de lectura del saldo en la cuenta y posteriormente hacer el depósito de $1.

Como el cliente es muy cuidadoso con su dinero debe validar para cada operación de lectura o de escritura que el valor devuelto por el servidor sea el correcto. En caso de un error el programa cliente deberá terminar e imprimir la razón por la que está terminando.

Ejercicio 2: Pruebe la aplicación distribuida y verifique que existen inconsistencias cuando el valor de n crece y hay pérdidas de mensajes. Explique por qué sucede y ahora resuelva el problema como lo recomienda Coulouris en la misma página 189 del texto. Para determinar las razones de las inconsistencias dibuje en su cuaderno el diagrama de los mensajes enviados entre cliente y servidor y sus contenidos.