Protocolo Solicitud-Respuesta confiable parte 1

Elaborado por: Ukranio Coronilla

Tal y como se menciona en la página 189 del libro "Distributed Systems" de George 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 nuestro caso conviene sacar al proceso del bloqueo después de cierto tiempo transcurrido (timeout) donde no se ha recibido respuesta, con la intención de enviar nuevamente una solicitud, pues se presupone que se ha perdido el mensaje enviado o el que se iba a recibir.

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 tendríamos:

```
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: Retome el código de la práctica anterior, y 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". También, modifique el método **doOperation** de la clase **Solicitud** para que se utilice **recibeTimeout**, y 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 tienen los gatos... Si en 7 veces no se logra recibir una respuesta entonces el programa termina e imprime que el servidor no está disponible, y que intente más tarde. Para probar sus métodos, pruebe a detener el proceso servidor con CTRL-Z para que no conteste y reactívelo con %1.

Ejercicio 2: Pruebe su aplicación con el cliente y el servidor en distintas computadoras, y realice en un ciclo n solicitudes, donde n se recibe como parámetro en la línea de comandos. ¿Cuántas solicitudes por segundo puede atender su servidor? Posteriormente optimice el tamaño del mensaje al definir **TAM_MAX_DATA** con el valor mínimo necesario, y determine ¿Cuántas solicitudes por segundo puede atender su servidor? Observe el Histórico de red en su monitor de sistema para observar la cantidad de bytes enviados y recibidos por segundo. ¿Mejora el tiempo si se eliminan las impresiones a pantalla?