Multicast confiable básico

Elaborado por: Ukranio Coronilla

De la misma forma que los mensajes UDP unicast no son confiables, tampoco lo son los mensajes UDP multicast. En esta práctica agregaremos un modelo muy básico de confiabilidad a los mensajes multicast. Este modelo simple se explica en el subtema 8.4.1 (página 343) del libro "Sistemas Distribuidos" del autor Andrew S. Tanenbaum, y se ilustra en la figura 8-9. Asegúrese de entender cómo funciona para que realice la siguiente implementación:

Ejercicio 1

A la clase `SocketMulticast` agregue el método:

```
int enviaConfiable(PaqueteDatagrama & p, unsigned char ttl, int num_receptores);
```

El cual se encarga de hacer el envío confiable de un mensaje multicast. Este método es similar al método `envia()`, solo que se le ha agregado el número de receptores, el cual se debe conocer de antemano. Se sugiere enviar una estructura que contenga los datos y el identificador de mensaje que se explica en el libro de Tanenbaum.

Este método debe devolver –1 en el caso de que el mensaje no les haya llegado a **todos** los receptores.

Agregue también el método:

```
int recibeConfiable(PaqueteDatagrama & p);
```

Que se debe ejecutar para recibir mensajes multicast que implementan un protocolo confiable.

Ejercicio 2

Para probar el funcionamiento de su programa vamos a simular una nano base de datos que solo almacena la cuenta en pesos de un cliente en la variable entera **nbd**, y cuyo valor inicial es cero. Por seguridad, esta nano base de datos se encuentra replicada en tres servidores ubicados en tres computadoras independientes. En una cuarta computadora vamos a ejecutar un cliente que recibe en la línea de comandos un entero n, y va a ejecutar ese número de depósitos de una cantidad aleatoria de pesos comprendida entre $1 y $9 sobre la cuenta.

Para que la base de datos en los tres servidores se mantenga consistente en el monto, es necesario que el cliente realice los depósitos mediante envíos multicast confiables hacia las tres bases de datos.

Pruébelo varias veces y con números n grandes para comprobar que la base de datos se mantiene consistente en los tres servidores.

TIPS:

Recuerde de lo visto en teoría que un proceso no puede crear dos sockets con el mismo puerto. En nuestro caso una instrucción como:

```
socket_nuevo = new SocketDatagrama(puerto_fijo);
```

dentro de un método, no liberará el puerto aún si se ha terminado de ejecutar el método. Para lograrlo es necesario en este caso mandar llamar al destructor:

```
socket_nuevo->~SocketDatagrama();
```

antes de terminar dicho método.

Para probar la correcta creación de los sockets, pruebe a imprimir los valores devueltos por las funciones `socket( )` y `bind()`.