Protocolo Solicitud-Respuesta en C++

Elaborado por: Ukranio Coronilla

El protocolo solicitud respuesta se revisó en la lectura del tema 5.2 y se muestra en la figura 5.2 del libro “Distributed Systems” de George Coulouris.

Vamos a programar las tres operaciones del protocolo que se muestran en la figura 5.3 utilizando la estructura del mensaje que se observa en la figura 5.4.

La intención de este protocolo de alto nivel es ocultar al programador la programación de los sockets en el cliente y en el servidor, y utilizar métodos comunes a cualquier aplicación del tipo cliente-servidor.

Nuestra aplicación cliente-servidor va a ser la misma que la mostrada en los ejemplos vistos en el manual de “Programación de Sistemas Linux”, es decir, el cliente envía dos números enteros, el servidor los suma y devuelve el resultado de la suma hacia el cliente. Los dos enteros se le deben pasar al cliente como parámetros en la línea de comandos.

Para facilitar la elaboración de esta práctica se recomienda:

Tener el código del cliente y el servidor dentro de la misma carpeta para que ambos puedan utilizar las mismas clases, así si se modifica una clase no será necesario modificarla para el cliente y para el servidor.

Para compilar utilizar un archivo Makefile para el cliente y otro archivo para el servidor, en este caso solo es necesario utilizar la opción –f y a continuación el nombre del archivo Makefile. Por ejemplo, si el archivo make del cliente se llama MakefileCliente, entonces compilamos así:

```
make –f MakefileCliente
```

Para evitar confusiones, utilizar un solo archivo header (por ejemplo, `mensaje.h`) para definir el mensaje de la figura 5.4 como sigue:

```
#define TAM_MAX_DATA 4000

//Definicion de identificadores para operaciones permitidas
#define suma 1

// Definicion del formato de mensaje
struct mensaje{
   int messageType;                 //0= Solicitud, 1 = Respuesta
   unsigned int requestId;    //Identificador del mensaje
   int operationId;                 //Identificador de la operación
   char arguments[TAM_MAX_DATA];
};
```

El tamaño máximo de información se define en este caso como 4000, considerando que muchas aplicaciones distribuidas utilizan 4K Bytes como tamaño de mensaje UDP y descontando los otros datos miembros, sin embargo, se sugiere ajustarlo a las necesidades de la aplicación. También hemos eliminado los miembros IP y puerto, considerando que dicha información ya se encuentra implícita en el protocolo UDP.

Las operaciones de la figura 5.3 se pueden clasificar en operaciones que ejecuta quien hace la solicitud, y operaciones que ejecuta quien da la respuesta. Por esta razón incluiremos dos clases nuevas cuyos métodos son las operaciones de la figura 5.3, y que especificamos como sigue:

La clase **Solicitud** cuya interfaz básica se muestra a continuación:

```
class Solicitud{

public:

   Solicitud();

   char * doOperation(char *IP, int puerto, int operationId, char *arguments);

private:

   SocketDatagrama *socketlocal;

};
```

El objeto SocketDatagrama se deberá instanciar en el constructor como sigue:

```
socketlocal = new SocketDatagrama(0);
```

puesto que el usuario de esta clase es un cliente y su puerto será variable.

Los datos privados adicionales y la implementación del método **doOperation** son a completa satisfacción del programador.

La clase **Respuesta** incluye una interfaz básica que se muestra a continuación:

```
class Respuesta{
public:
   Respuesta(int pl);
   struct mensaje *getRequest(void);
   void sendReply(char *respuesta);
private:
   SocketDatagrama *socketlocal;
};
```

En esta clase el constructor recibe como parámetro el puerto al que estará escuchando solicitudes el servidor. El método **sendReply** no incluye IP ni puerto pues se presupone que se envía al proceso remoto que hizo la solicitud.

El código principal del servidor solo tendrá que incluir la librería

```
#include "Respuesta.h"
```

Y podrá hacer uso de la clase y sus métodos.

Observe que el método **getRequest** devuelve una estructura mensaje porque frecuentemente el código del servidor debe tener información al menos del **operationId** para saber qué operación de las varias que dispone el servidor le están solicitando.

Pruebe que funciona su aplicación en distintas computadoras, y asegúrese que en todo momento el cliente y el servidor se comunican a través de un mensaje UDP que contiene a la estructura `mensaje`, y no mediante una cadena simple.