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, tal y como lo hacía la RPC de Sun. De hecho en esta práctica estaremos construyendo nuestra propia RPC utilizando las clases `PaqueteDatagrama` y `SocketDatagrama`.

Nuestra RPC consiste en que el cliente le envíe al servidor una cadena de varias palabras, el servidor las invierte y regresa la cadena conteniendo las palabras en orden inverso (la última palabra ahora es la primera y así con todas las demás) . El cliente recibe como parámetro la IP y puerto del servidor y el servidor solo el puerto donde estará atendiendo las solicitudes.

Para facilitar la elaboración de esta práctica se requiere:

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 como se hizo en la práctica 14 del manual, para tener dos archivos Makefile con nombres distintos.

Utilizar un solo archivo header para definir el mensaje de la figura 5.4 como sigue:

mensaje.h

```
#define TAM_MAX_DATA 4000

struct mensaje{
   int messageType;           //0= Solicitud, 1 = Respuesta
   unsigned int requestId;    //Identificador del mensaje
   char IP[16];
   int puerto;
   int operationId;           //Identificador de la operación
   char arguments[TAM_MAX_DATA];
};
```

Observe que el tamaño máximo de información se define en este caso como 4000, considerando que muchas aplicaciones distribuidas utilizan 4KB (4096 Bytes) como tamaño de mensaje UDP y descontando los otros datos miembros.

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.

El código principal del cliente solo tendrá que incluir la librería

```
#include "Solicitud.h"
```

Y podrá hacer uso de la clase simplemente con instanciar un objeto y mandar llamar a su método `doOperation` como sigue:

```
Solicitud cliente;

printf("La respuesta del servidor es <%s>\n",cliente1.doOperation (argv[1], atoi(argv[2]),
ordenaCadena, cadenota));
```

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

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

En esta clase el constructor recibe como parámetro el puerto al que estará escuchando solicitudes el servidor.

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.

Después de programar cliente y servidor es posible elaborar una optimización en los tiempos de transferencia, ¿cuál sería esta?