Área de Ingeniería Telemática
Departamento de Ingeniería de Sistemas y Automática
Escuela Superior de Ingenieros
Universidad de Sevilla

LLAMADAS A PROCEDIMIENTO REMOTO

1. OBJETIVO

El objetivo es mostrar el funcionamiento de las llamadas a procedimiento remoto mediante la prueba de un ejemplo simple.

2. DESCRIPCIÓN

Las llamadas a procedimiento remoto permiten crear servidores de una forma fácil y cómoda. Para la especificación de un servicio en las RPCs se debe definir un conjunto de procedimientos dentro de un programa. Cada programa puede tener varias versiones. Esto permite cambiar la interfaz del servicio permitiendo que clientes que utilizan versiones anteriores sigan funcionando. Los programas, las versiones y procedimientos se identifican mediante números enteros. Un procedimiento queda totalmente identificado mediante la terna:

<programa, version, procedimiento>

Los números de versión se utilizan en las RPC para permitir que existan servidores similares (mismo programa) que exportan interfaces distintas (distintas versiones).

El primer paso que hay que dar para la creación de un servidor es especificar su interfaz, donde se indican los procedimientos que el servidor exporta. Esta especificación se escribe en un fichero cuya extensión es `.x`.

Por ejemplo, para un servidor con dos procedimientos remotos, sumar y restar, la interfaz es la siguiente:

```
program CALCULAR {
    version UNO {
        int sumar (int a, int b) = 1;
        int restar(int a, int b) = 2;
    }=1;
}=999999999;
```

calcular.x

En calcular.x se ha definido la versión `UNO` (con identificador `1`) del programa `CALCULAR` (con identificador `999999999`) que contiene el procedimiento `sumar` (con identificador `1`) y el procedimiento `restar` (con identificador `2`). No se puede utilizar como número de procedimiento el valor cero.

Una vez definida la interfaz, se utiliza el programa `rpcgen` para generar automáticamente el soporte (stub) del cliente y del servidor. Debido a que se ha utilizado más de un argumento en los procedimientos hay que indicárselo explícitamente a este programa con la opción `–N` (`-C` para generar código ANSI C):

```
rpcgen –N –C calcular.x
```

con esta orden se generarán automáticamente los siguientes ficheros:

`calcular.h`: fichero de cabecera a incluir en el cliente y en el servidor.
`calcular_svc.c`: soporte del servidor.
`calcular_clnt.c`: soporte del cliente.
`calcular_xdr.c`: funciones para la transformación de tipos.

A continuación el programador debe codificar el programa cliente y el programa servidor.

Los servidores deben implementar cada una de las funciones especificadas en el archivo de definición de interfaces. Estas funciones, que deben ser implementadas por el programador, tienen la siguiente forma:

```
tipo_resultado *procedimiento_V_svc(tipo_argumento arg, struct svc_req *sr);
```

Al nombre del procedimiento se le añade el número de versión seguido de `_svc` (porque pertenece al servidor). El segundo argumento permite acceder, aunque no se utiliza normalmente, a diferentes características como el número de versión o el número de procedimiento.

En el ejemplo, el programa servidor debe incluir la implementación de las funciones `sumar` y `restar` de la siguiente forma:

```
#include "calcular.h"
int * sumar_1_svc(int a, int b, struct svc_req *rqstp)
{
    static int r;
    r = a + b;
    return(&r);
}

int * restar_1_svc(int a, int b, struct svc_req *rqstp)
{
    static int r;
    r = a - b;
    return(&r);
}
```

calcular_servidor.c

Las variables que almacenan el resultado deben ser de tipo static (`static int r`), debido a que las funciones anteriores devuelven un puntero a entero.

El cliente, para poder ejecutar un procedimiento remoto debe en primer lugar establecer una conexión con el servidor, mediante el siguiente servicio:

```
CLIENT *clnt_create(char *host,u_long prognum,u_long versnum,char *protocol);
```

El primer argumento especifica el nombre de la máquina donde ejecuta el servidor. Los dos siguientes argumentos especifican el número de programa y número de versión del procedimiento remoto a ejecutar. El último argumento especifica el protocolo de transporte empleado (UDP o TCP). Devuelve un manejador de cliente (`CLIENT *`) que permite al cliente realizar llamadas al servidor. Si falla devuelve `NULL` y se puede usar `clnt_pcreateerror()` para imprimir la razón del error.

El prototipo que debe emplear el cliente para las llamadas a procedimientos remotos es:

```
tipo_resultado *procedimiento_V(tipo_argumento arg, CLIENT *cl);
```

donde el sufijo `V` representa el número de versión.

El programa cliente para el ejemplo del servidor para el programa `CALCULAR` es el siguiente:

```
#include "calcular.h"
int main (int argc, char **argv)
{
    char *host;
    CLIENT *sv;
    int *res;
    if (argc!=2)
        printf("Uso: %s  <host>\n", argv[0]);
    else
      {
        host = argv[1];
        sv = clnt_create(host, CALCULAR, UNO, "tcp");
        if (sv != NULL)
          {
            res = sumar_1(5, 2, sv);
            if (res != NULL)
            {
              printf("5 + 2 = %d\n", *res);
            }
            else
            {
               clnt_perror(sv, "error en RPC");
            }
            clnt_destroy(sv);
          }
        else
          {
            clnt_pcreateerror(host);
          }
      }
   return(0);
}
```

calcular_cliente.c

El programa cliente recibe el nombre de la máquina donde ejecuta el servidor en la línea de argumentos y la almacena en la variable host. Se ha utilizado la función clnt_destroy que permite romper la asociación entre el cliente y el servidor.

Ejercicios:

⇒ Añadir el procedimiento multiplicar (dos enteros) al programa calcular indicando una versión diferente. Realizar las modificaciones oportunas en el servidor. Crear un cliente que ofrezca la opción de sumar, restar o multiplicar, pidiendo los datos por teclado.

⇒ En la especificación de un servidor de ficheros se ha pensado en incluir un método para escribir en un fichero con 3 argumentos, correspondientes al descriptor de ficheros en el servidor, el número de bytes que se quieren escribir y el buffer donde están estos datos. Razonar que pasaría en el caso de que el cliente no obtuviera respuesta del servidor e intentara realizar la petición de nuevo. ¿Esta petición es idempotente?

⇒ En el caso anterior, el servidor debe incluir además un método para la apertura del fichero, previa a la escritura, para la obtención del descriptor del fichero en el servidor. También debe incluir otro método para el cierre del fichero cuando ya no se necesite más por parte del cliente. ¿Qué ocurre si el servidor cae y vuelve a arrancar? ¿Qué ocurre si el cliente cae?

⇒ Realizar un servidor de ficheros simplificado (sin estados), que permita leer una porción de un fichero remoto (procedimiento leer), permita añadir datos en un fichero remoto (procedimiento escribir), permita crear un fichero (procedimiento crear) y borrar un fichero (procedimiento borrar). La opción –a de rpcgen genera un ejemplo de cliente y servidor que pueden servir como base. La especificación de entrada a rpcgen de este servidor de ficheros sería la siguiente:

fs.x

```
const BUF_SIZE=1024;
const MAX_PATH=256;

typedef opaque BUF<BUF_SIZE>;
struct LEER_result {
  int cod_error;
  BUF datos;
  };

program FS {
   version UNO {
      LEER_result leer (string nom<MAX_PATH>, int offs, int nbytes)= 1;
      int escribir (string nom<MAX_PATH>, int offs, int nbytes, BUF buf)= 2;
      int crear (string nom<MAX_PATH>)= 3;
      int borrar (string nom<MAX_PATH>)= 4;
   } = 1;
} = 999999999;
```