Práctica 3. Llamadas a procedimientos remotos. Sun-RPC

SCS, 2010/11

16 de noviembre de 2010

Índice

1. Ejercicio 1: Compilación y evaluación de llamadas remotas

Se mostrará el uso del IDL de Sun-RPC (XDR), el funcionamiento del compilador de interfaces (**rpcgen**) y el desarrollo de clientes y servidores que hagan uso de las llamadas remotas. Se evaluarán también las diferencias de rendimiento entre llamadas locales y remotas.

- Información sobre XDR (*eXternal Data Representation*)
- RFC con la especificación XDR

Se parte de un código de ejemplo que implementa una *"calculadora remota"*.

Descarga: rpc1.tar.gz

El código de partida incluye:

- `calculadora.x`: Especificación XDR del interfaz remoto que exporta las cuatro operaciones básicas para enteros (suma, resta, multiplicación y división)
- `servidor.c`: Implementación, conforme al esquema esperado por el **skeleton** de Sun-RPC, de las 4 funciones exportadas
- `cliente.c`: Ejemplo de cliente donde se muestra la forma en que realizan las llamadas remotas a las funciones remotas exportadas.

1.1. Pasos a seguir

1. Descomprimir el paquete.

   ```
   $ tar xzvf rpc1.tar.gz
   $ cd rpc1
   ```

2. Comprobar el contenido del fichero `calculadora.x` con el interfaz remoto

   - Usa el esquema de parámetros *"clásico"* que sólo permite un argumento de entrada en los procedimientos remotos
   - Por esa razón se define la estructura `entrada` para contener los dos operandos (`arg1` y `arg2`) que se pasan a cada una de las cuatro operaciones

3. Compilar con **rpcgen** la definición del interfaz remoto

   ```
   $ rpcgen calculadora.x
   ```

   Se generan los siguientes ficheros.

   - `calculadora.h`: Fichero de cabecera con constantes y las definiciones de las estructuras usadas como parámetros de entrada y como valores de salida.
   - `calculadora_clnt.c`: Código C con la implementación del **stub**
   - `calculadora_svc.c`: Código C con la implementación del **skeleton**
   - `calculadora_xdr.c`: Código C con la implementación de las rutinas XDR para aplanar/desaplanar los argumentos y el valor de retorno

   Se peuden abrir con un editor y comprobar su estructura, las defincciones de tipos de datos, los prototipos de las funciones del **stub** y ver cómo son las llamadas realizadas en **stub** y **skeleton**.

4. Compilar y lanzar el servidor

   ```
   $ gcc -o servidor servidor.c calculadora_svc.c calculadora_xdr.c
   $ ./servidor &
   ```

   Se pueden comprobar los programas RPC registrados en el **portmapper** de una máquina con la orden **rpcinfo**.

   ```
   $ rpcinfo -p
   ```

   Muestra el núm. de programa (en decimal) junto su núm. de versión [ambos especificados en el fichero XDR `calculadora.x`] y sus puertos de escucha (tcp y udp).

5. Compilar y ejecutar el cliente de ejemplo (en un terminal distinto)

   ```
   $ gcc -o cliente cliente.c calculadora_clnt.c calculadora_xdr.c
   $ ./cliente localhost &
   ```

   Se puede probar a ejecutar las funciones remotas de otra máquina del laboratorio poniendo su dirección IP en lugar de "`localhost`"

1.2. Tareas a realizar

1. Comprobar las funciones, prototipos de funciones y estructuras de datos presentes en los ficheros generados por **rpcgen**
2. Generar con `rpcgen -a calculadora.x` los prototipos de cliente y servidor (`calculadora_server.c`, `calculadora_client.`
   y compararlos con las implementaciones aportadas.
3. Evaluar el rendimiento de las llamadas remotas comparándolo con el de las llamadas locales.

- Se incluye el fichero `test_local.c` que realiza 100.000 de llamadas a una función `sumar` local y evalúa el tiempo consumido.

```
$ gcc -o test_local text_local.c
$ ./text_local
```

- A partir de `test_local.c` y de `cliente.c` se pide:
  - Construir un fichero `test_remoto.c` equivalente que realice 100.000 de llamadas remotas a la función `sumar_1()` del servidor RPC.
    Compilación:

```
$ gcc -o test_remoto test_remoto.c calculadora_clnt.c calculadora_xdr.c
$ ./test_remoto localhost
```

  - Comprobar el tiempo invertido en las llamadas remotas a la propia máquina y en llamadas remotas a otra de las máquinas del laboratorio que también exporten el interfaz `calculadora.x` (**aviso:** es el caso de llamadas a otras máquinas, las 100.000 llamadas pueden ser excesivas).

2. Ejercicio 2: Uso de tipos complejos en interfaces remotos

Se parte de un código de ejemplo que implementa una serie de procedimientos remotos para el procesamiento de vectores:

- escalado de un vector (multiplicación por una constante)
- suma de 2 vectores

Descarga: rpc2.tar.gz

2.1. Pasos a seguir

1. Descomprimir el paquete.

```
$ tar xzvf rpc2.tar.gz
$ cd rpc2
```

2. Contenido
   - `vector.x`: Definición XDR del interfaz remoto.
     - La sintaxis de XDR es similar a la C, pero no es C.
     - En este caso se usa el tipo de dato XDR que implementa un *"vector de tamaño variable"* `t_vector`, que no existe en C.
     - Al generar el código C ese *"vector variable"* se definirá como un estrucutra con dos campos: `t_vector_len` con la longitud del vector y `t_vector_val` con un puntero al array de números reales en memoria.

```
typedef struct {
   u_int t_vector_len;
   float *t_vector_val;
} t_vector;
```

   - `servidor_vector.c`: Implementación de los procedimientos remotos definidos.
   - `cliente_vector.c`: Ejemplo de llamadas a los métodos remotos exportados.

   Revisar el fichero `vector.x` y sus declaraciones de datos.

3. Compilar con **rpcgen** la definición del interfaz remoto

```
$ rpcgen vector.x
```

Genera los siguientes ficheros.

- `vector.h`: Fichero de cabecera con constantes y las definiciones de las estructuras usadas como parámetros de entrada y como valores e salida.
  - Comprobar la definición de los tipos
- `vector_clnt.c`: Código C con la implementación del **stub**
- `vector_svc.c`: Código C con la implementación del **skeleton**
- `vector_xdr.c`: Código C con la implementación de las rutinas XDR para aplanar/desaplanar los argumentos y el valor de retorno

4. Generar los ejecutables del servidor y del cliente y ejecutarlos

```
$ gcc -o servidor_vector servidor_vector.c vector_svc.c vector_xdr.c
$ gcc -o cliente_vector cliente_vector.c vector_clnt.c vector_xdr.c
$ ./servidor_vector &
$ ./cliente_vector localhost
```

2.2. Tareas a realizar

Se tratará de extender el interfaz remoto con una nueva operación de vectores, el **producto escalar** de 2 vectores.

- El **producto escalar** de $\vec{a} = (2, 3, 4)$ y $\vec{b} = (2, 1, 2)$ es el número $\vec{a} \cdot \vec{b} = 13$, que es el resultado de calcular $2 * 2 + 3 * 1 + 4 * 2$
- Ver producto escalar

Pasos:

1. Modificar el interfaz XDR "`vector.x`" para incluir un nuevo procedimiento `producto_escalar`
   - recibirá como parámetro una estructura con 2 vectores (como en `suma_vectores`)
   - devolverá un número real (`float`)
2. Compilar el nuevo interfaz con **rpcgen**

   ```
   $ rpcgen vector.x
   ```

   Idea: Se puede usar `rpcgen -a vector.x` y comprobar como será el prototipo de la nueva función a implementar/llamar en `vector_server.c` y `vector_client.c`.
3. Incluir en `servidor_vector.c` la implementación del método `producto_escalar_1_svc()`

   ```
   float * producto_escalar_1_svc(entrada2 *argp, struct svc_req *rqstp) {
      static float  result;
      ...
      return &result;
   }
   ```

4. Incluir en `cliente_vector.c` la llamada al método `producto_escalar_1()`

   ```
   float * resultado2;
   ...
   resultado2 = producto_escalar_1(&args2, clnt);
   ...
   printf("Resultado: %f\n", *resultado2);
   ```

5. Compilar cliente y servidor y comprobar su funcionamiento

```
$ gcc -o servidor_vector servidor_vector.c vector_svc.c vector_xdr.c
$ gcc -o cliente_vector cliente_vector.c vector_clnt.c vector_xdr.c
$ ./servidor_vector &
$ ./cliente_vector localhost
```

3. Documentación a entregar

- Para el **ejercicio 1** incluir el fragmento de código donde se implementan las 100000 llamadas remotas, presentar los resultados obtenidos para las llamadas locales y para las remotas y comentar brevemente los valores resultantes.
- Para el **ejercicio 2** se entregará el framento de código fuente donde se implementa el producto escalar de vectores (función `producto_escalar_1_svc()`) y el fragmento donde realiza la llamada a `producto_escalar_1()` desde el cliente.

  Se deberá comentar también el contenido del código generado automáticamente por **rpcgen**: `vector_clnt.c`, `vector_svc.c`, `vector_xdr.c`, `vector.h`, describiendo brevemente la finalidad de las estructuras de datos y las funciones que aportan.