# Aplicación RPC cliente/servidor

## RPC

RCP es un estándar desarrollado por Sun Microsystems y usado por muchos distribuidores de sistemas UNIX.

El RPC es una interfaz de programación de aplicación (API) disponible para el desarrollo de aplicaciones distribuidas. Permite que los programas llamen a subrutinas que se ejecutan en un sistema remoto. El programa llamador (llamado cliente) envía un mensaje de llamada al proceso servidor y espera un mensaje de respuesta. La llamada incluye los parámetros del procedimiento y la respuesta los resultados

El RPC de Sun consta de las siguientes partes:

- RPCGEN: Un compilador que toma la definición de la interfaz de un procedimiento remoto, y genera los "stubs" del cliente y del servidor.

- XDR ("eXternal Data Representation"): Una forma estándar de codificar datos de modo que sean transportables entre distintos sistemas.

- Una librería "run-time".

El proceso de llamadas de RPC se puede ver en la siguiente figura

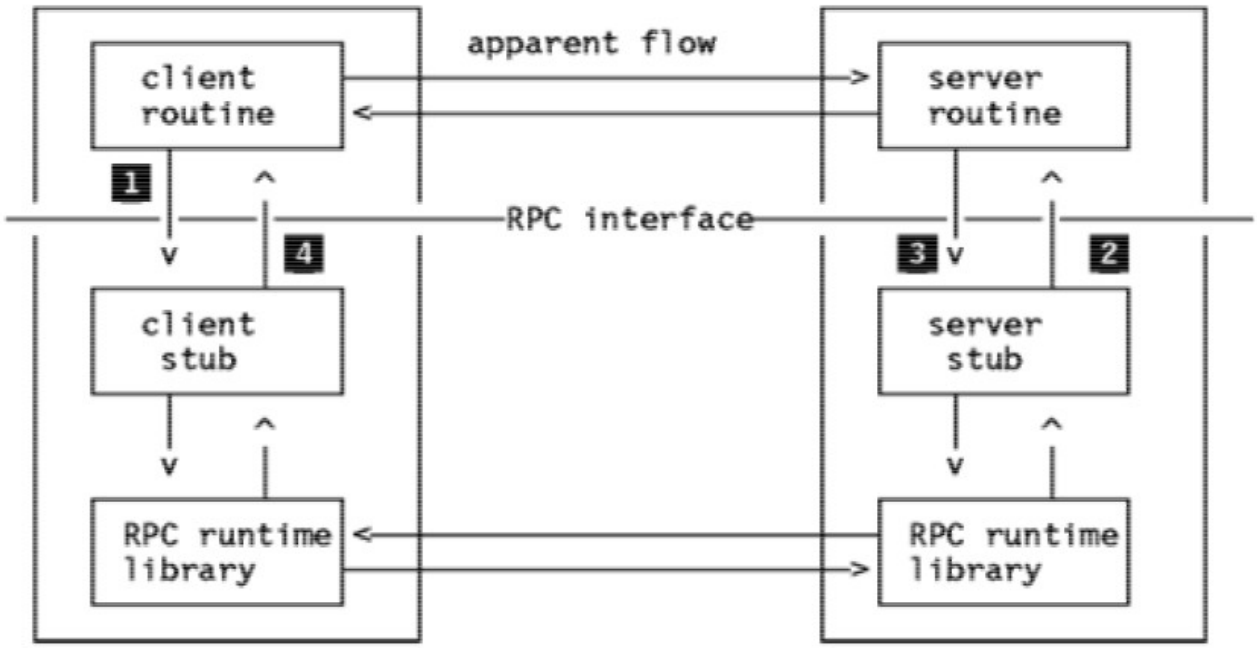


Figura 1. Esquema de comunicación RPC

Básicamente podemos describir el proceso de la siguiente manera:

1) El proceso llamador envía un mensaje de llamada y espera por la respuesta.

2) En el lado del servidor un proceso permanece dormido o no, dependiendo de la implementación, a la espera de mensajes de llamada. Cuando llega una llamada, el proceso servidor extrae los parámetros del procedimiento, calcula los resultados y los devuelve en un mensaje de respuesta

Un procedimiento remoto se identifica con el uso de tres campos:

- Número de programa remoto. Identifica un grupo funcional de procedimientos.
- Número de versión del programa remoto . A medida que el programa remoto sufre modificaciones se le va asignando el número de versión.
- Número del procedimiento remoto . Dentro del programa remoto cada procedimiento tiene un número único con el cual se le identifica.

Cada programa debe estar asociado a un puerto de los disponibles. El cliente llamador debe conocer dicho puerto.

Los números de programa son definidos de forma estándar:

- 0x00000000 - 0x1FFFFFFF: Definidos por Sun
- 0x20000000 - 0x3FFFFFFF: **Definidos por el usuario**
- 0x40000000 - 0x5FFFFFFF: Temporales
- 0x60000000 - 0xFFFFFFFF: Reservados

Existe una aplicación en el Servidor, `portmap`, encargada de mapear los puertos con los programas remotos. El funcionamiento es sencillo. `portmap` se encuentra escuchando en el puerto ***111*** y el cliente le pregunta por el puerto utilizado por un programa específico y `portmap` devuelve el puerto y así ya sabemos a que puerto conectarnos.

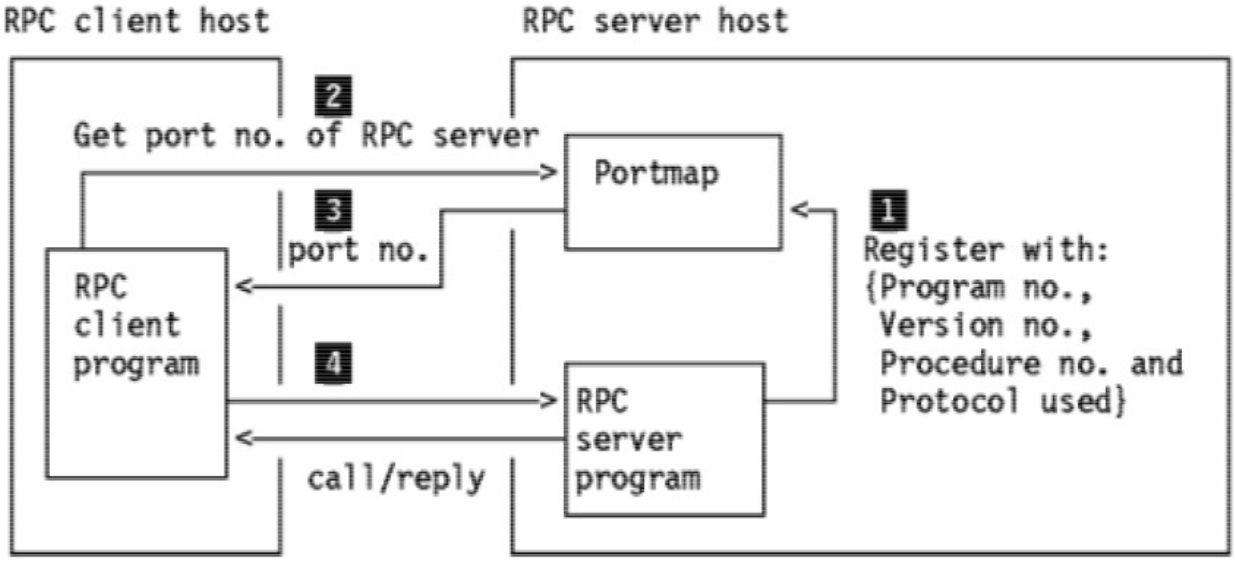


Figura 2. Funcionamiento de RPC

# RPCGEN

Se trata de una utilidad que genera código C para el protocolo RPC. Como parámetro le pasamos el nombre de un fichero escrito en un lenguaje parecido a C para definir los prototipos de las funciones que queremos que se publiquen. La extensión es `.x`. Si el argumento que le pasamos fuese `prueba.x` RPCGEN produce los siguientes archivos de salida.

- Un fichero cabecera llamado `prueba.h` que contiene definiciones comunes de constantes y macros.
- El código fuente del "stub" del cliente, `prueba_clnt.c`
- El código fuente del "stub" del servidor, `prueba_svc.c`
- El fichero fuente de rutinas XDR, `prueba_xdr.c`

Los pasos a realizar para crear una aplicación Cliente/Servidor con SunRPC
son los siguientes:

1) Creación del archivo `.x`
2) Generación de stubs, archivo XDR (conversión de tipos entre sistemas) y archivo `.h`
3) Creación del servidor
4) Creación del cliente
5) Compilación y ejecución

***1) Creación del archivo `.x`***

En este archivo especificamos la interfaz del objeto remoto, indicando el número de programa, número de versión del programa remoto y el número de los procedimientos remotos.

```
program SUMA_PROG {
    version SUMA_VERS {
        int SUMA(numeros) = 1; /* procedure number = 1 */
    } = 1;                     /* version number = 1 */
} = 0x20000001;                /* program number = 0x20000001 */
                              /* 0x20000000 - 0x3fffffff for users*/
```

Podemos definir constantes y estructuras como lo haríamos en C. SunRPC solamente permite el paso de un parámetro a los procedimientos remotos, por eso debemos crear estructuras con los parámetros que deseamos enviar. Es importante que el número del programa sea único para cada programa.

***2) Generación automática con RPCGEN***

Una vez hemos definido el prototipo de la funciones a publicar, con la herramienta `rpcgen` podemos generar, automáticamente, los stubs (clases proxys que son las que realmente permiten el acceso remoto), el archivos con las rutinas XDR y un archivo `.h` utilizado tanto por el cliente como por el servidor. Si utilizamos la opción `-a` obtendremos además clases de ejemplo para realizar la llamada.

```
$rpcgen -a prueba.x
```

Se generarán los siguientes archivos:

- `Makefile.prueba` → Archivo de compilación
- `prueba_clnt.c` → Stub cliente
- `prueba_server.c` → Programa servidor de ejemplo
- `prueba_xdr.c` → Como RPC permite llamadas de clientes a servidores que estén en máquinas distintas y, por tanto, puedan tener una arquitectura distinta, es necesario traducir los parámetros y resultados a un "código" universal, independiente de las máquinas. Si los parámetros son tipos básicos (int, float, char, etc), el sistema unix ya tiene unas funciones de conversión (xdr_int(), xdr_float(), etc). Si los parámetros, como en este caso, son estructuras definidas por nosotros, las funciones de conversión hay que hacerlas. rpcgen genera automáticamente dichas funciones y en nuestro caso, las ha metido en el fichero `prueba_xdr.c`
- `prueba_client.c` → Programa cliente de ejemplo
- `prueba.h` → Cabecera para el cliente y el servidor, contiene los prototipos de nuestras funciones y cualquier aplicación que quiera hacer uso de ellas deberá incluirlo.
- `prueba_svc.c` → Stub servidor

En este caso podemos modificar el cliente y el servidor e introducir la lógica que nosotros queramos.

***3) Creación del servidor***

En el caso anterior lo único que tendríamos que hacer es completar con nuestro código las funciones del servidor generado (`prueba_server.c`).

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "prueba.h"

int *
suma_1_svc(suma_num *argp, struct svc_req *rqstp)
{
	static int  result;

	/*
	 * insert server code here
	 */
```

```
        result = argp->a +argp->b;
        return &result;
}
```

*4) Creación del cliente*

El cliente que genera rpcgen tiene también todo hecho. Se conecta al servidor, llama a todas las funciones una por una y cierra la conexión. Obviamente aprovechamos el principio (la conexión) y el final (la desconexión). Las llamadas a las funciones deberíamos borrarlas y hacer que el cliente haga lo que nosotros queremos.

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "prueba.h"


void
suma_prog_1(char *host)
{
        CLIENT *clnt;
        int  *result_1;
        suma_num  suma_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, SUMA_PROG, SUMA_VERS, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = suma_1(&suma_1_arg, clnt);
        if (result_1 == (int *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
}


int
main (int argc, char *argv[])
{
```

```
        char *host;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        suma_prog_1 (host);
exit (0);
}
```

Debemos adaptar el cliente a lo que queramos hacer, pero nos servirá la forma de conectarnos y desconectarnos del servidor.

***5) Compilación y ejecución***

Una vez modificados el cliente y el servidor debemos ejecutar el Makefile para generar la aplicación.

```
$ make -f Makefile.prueba
```

Se generan los ejecutables `prueba_client` y `prueba_server`. A continuación ejecutaremos en la máquina servidora el programa servidor.

```
$ ./prueba_server &
```

Posteriormente ejecutaremos el cliente en la misma o diferente máquina indicando el host de la máquina servidora.

```
$ ./prueba_client localhost
```

Si se produce un error del tipo RPC: Program not registered posiblemente se deba a que `portmap` utiliza los ficheros de acceso `/etc/hosts.allow` y `/etc/hosts.deny`. Por defecto el `hosts.deny` prohibe el acceso a cualquier conexión. Deberíamos eliminar la linea que restringe el acceso y volver a lanzar los demonios `portmap` y `rpc.statd`.

```
$ portmap restart
$ rpc.statd restart
```

## Trabajo práctica.

1.- Realiza una aplicación cliente servidor que implemente una calculadora. El cliente pedirá dos números de un solo dígito y la operación a realizar. El cliente llamará al servidor para que realice la operación y devuelva el resultado.

2.- Realiza un programa que realice 100.000 llamadas a una función sumar local y nos dé el tiempo consumido.

3.- Realiza un programa que realice 100.000 llamadas a la función sumar del servidor instalado sobre el mismo ordenador que el cliente y nos dé el tiempo consumido.

4.- Igual que 3 pero con el servidor y el cliente en máquinas diferentes. En este caso las 100.000 llamadas pueden ser excesivas