RPC

Llamadas a Procedimientos Remotos

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Un sistema distribuido es un conjunto de computadoras conectadas en red que le dan la sensación al usuario de ser una sola computadora. Este tipo de sistema brinda una serie de bondades tales como: compartición de recursos, la concurrencia, alta escalabilidad, y tolerancia a fallos. A pesar que agregar complejidad al software y disminuir los niveles de seguridad, los sistemas de procesamiento distribuidos brindan una buena relación precio-desempeño y pueden aumentar su tamaño de manera gradual al aumentar la carga de trabajo.

RPC es una tecnología, tradicionalmente empleada en ambiente UNIX, que permite el desarrollo de sistemas de procesamiento distribuido basados en el paradigma procedimental. Con el advenimiento de implementaciones para plataforma Windows, así como para Java, es posible pensar en RPC como un mecanismo de integración de software heterogéneo.

Este documento presenta las características técnicas más importantes del mecanismo de Llamadas a Procedimientos Remotos (RPC, Remote Procedure Calls), especialmente los relacionados con la implementación ONC (Open Network Computing).

2. GENERALIDADES

Una llamada a un procedimiento (función o subrutina) es un método bien conocido para transferir el control de una parte del programa a otra, con un retorno del control a la primera. Asociado con la llamada a un procedimiento están el pase de argumentos y el retorno de uno o varios resultados. Cuando el código que invoca a un procedimiento y dicho procedimiento están en un mismo proceso en un computador dado, se dice que ha ocurrido una llamada a un procedimiento local.

Por el contrario, en una llamada a un procedimiento remoto (RPC, Remote Proceure Call) el sistema local invoca, a través de la red, a una función alojada en otro sistema. Lo que se pretende es hacerle parecer al programador que está ocurriendo una simple llamada local.

Se utiliza el término solicitud para referirse a la llamada que realiza el cliente al procedimiento remoto (servidor) y, el término respuesta, para describir el resultado devuelto por éste último.

La figura 1 ilustra el proceso de una llamada a un procedimiento remoto:

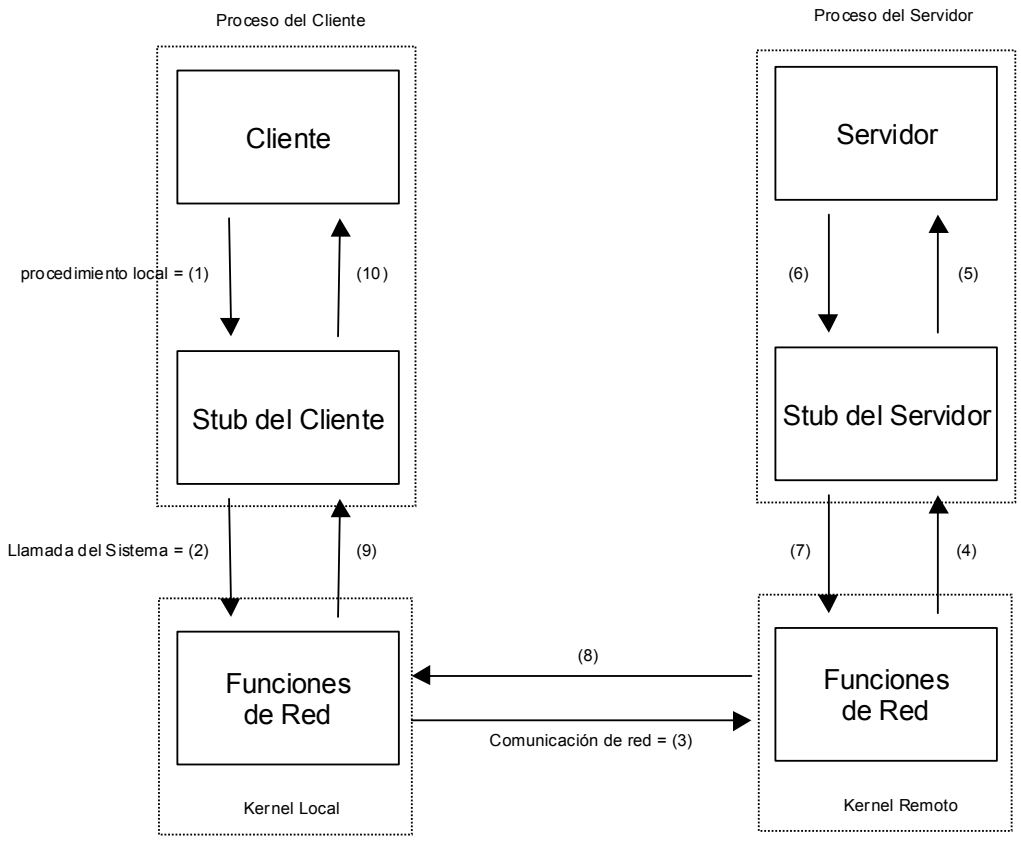

Figura 1. Modelo RPC

1. El cliente llama a un procedimiento local llamado “stub” del cliente, el cual aparenta ser el procedimiento servidor que el cliente desea llamar. El propósito del “stub” del cliente es empaquetar los argumentos del procedimiento remoto, adecuarlos a algún formato estándar y construir uno o varios mensajes de red. El empaquetamiento de los argumentos del procedimiento remoto en mensajes de red se conoce como “marshaling”.
2. Estos mensajes son enviados por el “stub” del cliente al sistema remoto, lo cual requiere una llamada del sistema.
3. Los mensajes son transferidos al sistema remoto empleando protocolos con o sin conexión.
4. Un procedimiento “stub” del servidor espera en el sistema remoto la solicitud del cliente. Desempaqueta los argumentos de los mensajes de red y si es necesario realiza alguna conversión.
5. El “stub” del servidor realiza la llamada al procedimiento local que realmente invoca la función del servidor y le pasa los argumentos transferidos a través de la red desde el “stub” del cliente.
6. Cuando el procedimiento del servidor termina, éste le regresa el control al “stub” del servidor devolviendo los resultados obtenidos.
7. El “stub” del servidor adecua el formato de tales resultados, si es necesario, y los empaqueta en mensajes de red para ser devueltos al “stub” del cliente.
8. Los mensajes son transmitidos al “stub” del cliente.
9. El “stub” del cliente lee los mensajes recibidos.
10. Luego de posiblemente convertir los valores de retorno, el “stub” del cliente retorna finalmente dichos resultados a la función del cliente haciendo parecer un retorno normal de función.

El concepto de llamada a procedimiento remoto permite ocultar en los “stubs” todos los detalles del código correspondiente a la comunicación a través de la red. Esto permite que los desarrolladores de programas de aplicación no se preocupen por detalles tales como “sockets” y ordenamiento de bytes. Uno de los objetivos de RPC es facilitar el desarrollo de aplicaciones distribuidas.

Según el modelo OSI, RPC cae en algún lado entre la capa de transporte y aplicación. Típicamente se considera parte de la capa de presentación. Debido a que RPC le "oculta" a la capa de aplicación los detalles de la red, usualmente incluye una especificación de algún formato estándar para el intercambio de argumentos y resultados entre el cliente y el servidor.

Las implementaciones de RPC más populares son:

- la desarrollada por Sun Microsystem denominada ONC-RCP (Open Network Computing, ONC-RCP), distribuida con casi todos los sistemas UNIX.
- la desarrollada por Microsoft en línea con el Ambiente de Computación Distribuida (DCE, Distributed Computing Enviroment) definido por la Fundación de Software Abierto (OSF, Open Software Foundation). Incluida en los sistemas operativos Windows.

Este documento describe mayormente la implementación ONC–RCP gracias a su sencillez y mayor difusión en entornos cliente/servidor, producto de su distribución junto a sistemas UNIX. Adicionalmente, se han identificado y realizado pruebas de interoperabilidad con implementaciones en lenguaje C para plataforma Windows y en Java, lo cual le abre una nueva dimensión a esta tecnología dándole una verdadera capacidad multiplataforma.

Un ejemplo de la aplicación de esta tecnología es el Sistema de Archivos en Red (NFS, Network File System), disponible en la mayoría de plataformas UNIX.

3. CARACTERÍSTICAS DE TRANSPARENCIA

Para lograr que a una aplicación le sea indistinto realizar una llamada a un procedimiento remoto o una llamada local, es necesario considerar los siguientes aspectos:

3.1. PASE DE PARÁMETROS

El pase de parámetros puede no ser transparente. Para el caso donde los parámetros son pasados por valor la implementación es simple. Sin embargo, se presentan problemas cuando se intenta pasar parámetros por referencia. Es por esto que típicamente las implementaciones sólo le permiten al cliente pasar argumentos por valor. Para cada procedimiento remoto se define específicamente los argumentos de entrada y los valores de retorno.

3.2. ENLACE

Esto se refiere a que el cliente contacte al sistema remoto apropiado para la ejecución de un procedimiento específico. Existen dos componentes a enlazar:

- el host remoto, y
- el proceso servidor deseado en dicho host.

Diferentes técnicas son empleadas para lograr este objetivo.

3.3. PROTOCOLO DE TRANSPORTE

Algunas implementaciones de RPC emplean un único protocolo de transporte mientras otras permiten su selección. La mayoría de las implementaciones dicen ser independientes del protocolo, sin embargo éstas sólo soportan uno o dos protocolos. Específicamente, las implementación ONC y DCE soportan TCP y UDP como protocolos de transporte.

3.4. MANEJO DE EXCEPCIONES

En un ambiente de computación distribuida la posibilidad de fallo aumenta, considerando que tanto los sistemas local y remoto como la red de datos son potenciales

fuentes de fallas. La implementación debe proveer mecanismos para manejar tales situaciones.

3.5. SEMÁNTICA DE LLAMADAS

Cuando se realiza una llamada a un procedimiento local, uno no se pregunta cuantas veces el procedimiento se ejecutó. Si un procedimiento retorna entonces se ejecutó exactamente una vez. Sin embargo, si se considera un procedimiento remoto del cual no se ha obtenido respuesta después de un intervalo de tiempo, no se puede tener certeza si se ha ejecutado. Si el servidor experimenta un error de ejecución, por ejemplo, antes de que el "stub" del lado del servidor se realice la llamada, entonces ésta no se habrá ejecutado. Si el servidor experimenta un error de ejecución después de haber devuelto su resultado al "stub", se habrá ejecutado una vez. Si el cliente retransmite una solicitud por no haber recibido una respuesta del servidor, se confunden aún más las cosas. Es posible que la primera solicitud haya sufrido un retraso en algún lugar de la red y finalmente, haya sido ejecutada al igual que la solicitud retransmitida.

Existe tres tipos de semánticas de RPC: exactamente una vez, cuando mucho una vez, y al menos una vez

3.6. REPRESENTACIÓN DE LOS DATOS

Empleando llamadas locales no existen problemas de incompatibilidad de la data puesto que el formato binario de todos los tipos de datos es el mismo. Pero cuando el cliente y el servidor se ejecutan en sistemas de arquitecturas diferentes es necesario una conversión de datos. Todas las implementaciones manejan este aspecto definiendo uno o más formatos estándar para los tipos de datos soportados por la implementación. La representación estandarizada de datos empleada por ONC-RPC es XDR, la cual impone un ordenamiento "big-endian" y un tamaño mínimo de 32 bits para cualquier campo. Dicha implementación utiliza tipificación implícita, es decir que sólo el valor de la variable es transmitida a través de la red.

3.7. DESEMPEÑO

Usualmente el desempeño empleando RPC disminuye entre 10 y 100 veces en comparación con un procedimiento local. Sin embargo, RPC debe ser visto como una herramienta que reduce el tiempo de desarrollo de aplicaciones distribuidas.

3.8. SEGURIDAD

Con RPC existen los mismos problemas de seguridad relacionados con la ejecución remota de comandos.

4. ONC-RPC (Open Network Computing - Remote Procedure Call)

Como se mencionó anteriormente, esta implementación fue desarrollada inicialmente por Sun Microsystem y está disponible en la gran mayoría de los sistemas UNIX. La especificación de ONC-RPC versión 2 está descrita en la RFC 1831 (Agosto, 1995). La RFC 1832 provee la descripción de XDR (Agosto, 1995). Ambas pueden ser obtenidas en el sitio http://www.rfc-editor.org.

ONC-RPC cuenta con los siguientes componentes:

- `rpcgen`, un compilador que toma la definición de la interfaz de un procedimiento remoto y genera el “stub” del cliente y el “stub” del servidor.
- XDR (eXternal Data Representation), un estándar para la descripción y codificación de datos que garantiza portabilidad entre sistemas de arquitecturas diferentes.
- Una librería que maneja todos los detalles.

4.1. XDR (EXTERNAL DATA REPRESENTATION)

XDR es un estándar para la descripción y codificación de datos que utiliza un lenguaje cuya sintaxis es similar a la del lenguaje de programación C. Es empleado principalmente en la transferencia de información entre diferentes arquitecturas computacionales y se ubica dentro de la capa de presentación del modelo ISO. Involucra un mecanismo de tipificado implícito (Implicit Typing), es decir que sólo viaja el valor de la variable por la red.

Es importante resaltar que XDR no es un lenguaje de programación, sino una especificación que incluye un lenguaje de descripción de datos el cual es extendido por ONC-RPC para la definición de procedimientos remotos.

Los tipos de datos de XDR presentan cierta similitud con los tipos de datos de C. Algunos de estos son:

- `int`
- `unsigned int`
- `enum`
- `bool`
- `hyper`
- `unsigned hyper`
- `float`
- `double`
- `quadruple` (punto flotante de cuádruple precisión)
- `opaque` (de longitud fija o variable)
- `string`
- `array` (de longitud fija o variable)
- `struct`
- `union` (uniones discriminadas)
- `void`
- `constant`

4.2. EXTENSIÓN DE XDR PARA LA DECLARACIÓN DE PROCEDIMIENTOS

ONC-RPC es un protocolo de mensajes especificado en XDR.

El lenguaje especificado por RPC es idéntico al lenguaje de XDR, excepto que agrega la definición de "programa", cuya gramática es mostrada a continuación:

```
program-def:
   "program" identifier "{"
      version-def
      version-def *
   "}" "=" constant ";"

version-def:
   "version" identifier "{"
      procedure-def
      procedure-def *
   "}" "=" constant ";"

procedure-def:
   type-specifier identifier
      "(" type-specifier("," type-specifier )* ")" "=" constant ";"
```

4.3. DESARROLLO DE APLICACIONES EMPLEADO ONC-RPC

A continuación se describen los pasos para el desarrollo de una simple aplicación cliente/servidor que devuelve la hora empleando ONC-RPC (Figura 2):

Desde un punto de vista muy general, se deben proveer tres archivos: el archivo de especificación RPC, los procedimientos del servidor que son llamados remotamente desde el cliente y la función `main` del cliente.

El compilador `rpcgen` toma el archivo date.x y genera dos archivos .c, el "stub" del cliente y el "stub" del servidor, junto a un archivo .h que es incluido por ambos archivos stubs. Posteriormente, el "stub" y la función main del cliente son compilados para generar el programa cliente ejecutable.

```
cc –o rdate rdate.c date_clnt.c –lrpclib
```

Igualmente, se genera el programa servidor compilando el "stub" del servidor (que contiene la función main del servidor) junto con nuestras funciones del servidor.

```
cc –o date_svc date_proc.c date_svc.c –lrpclib
```

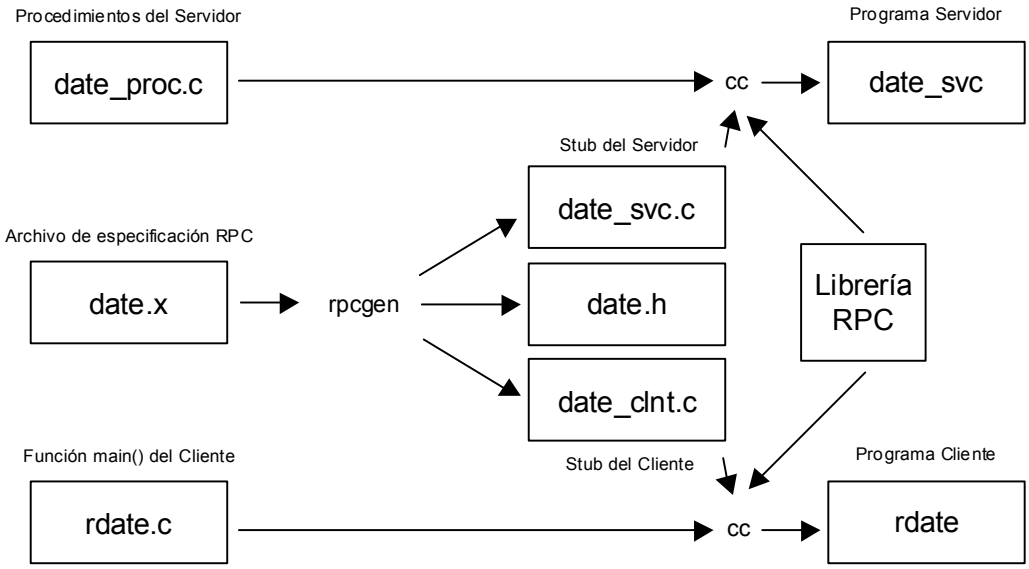

Figura 2. Archivos necesarios para el desarrollo de una aplicación cliente – servidor empleando ONC RPC.

A continuación se presenta el archivo de especificación el cual es la entrada al compilador `rpcgen`:

```
/*
 * date.x - Especificación de servicio de tiempo remoto
 */

/*
 * Define dos procedimientos
 *    bin_date_1(): devuelve la hora y fecha en formato
 *    binario
 *    str_date_1(): toma la hora y fecha en formato binario y
```

```
 *    lo transforma en un cadena de caracteres
 */

program DATE_PROG {
   version DATE_VERS {
      long BIN_DATE(void) =1;  /* procedimiento No. 1 */
      string STR_DATE(long) =2; /* procedimiento No. 2 */
   } = 1;   /* version No. 1 */
} = 0x31234567;   /* numero de programa */
```

Se han declarado ambos procedimientos y especificado los argumentos y el valor de retorno de cada uno. Se asignó un número a cada función (1 y 2), junto con un número de programa (0x31234567) y un número de versión (1). Los números de programas son enteros de 32 bits y son asignados como sigue:

```
0x00000000 – 0x1fffffff   definidos por Sun Microsystem
0x20000000 – 0x3fffffff definidos por el usuario
0x40000000 – 0x5fffffff transitorios
0x60000000 – 0xffffffff reservados
```

El compilador genera los verdaderos nombres de los procedimientos remotos convirtiendo `BIN_DATE` y `STR_DATE` en `bin_date_1` y `str_date_1`.

A pesar de que la gramática especificada en el apartado anterior sugiere lo contrario, ONC-RPC sólo permite un argumento y un resultado. Si más argumentos requieren ser especificados se debe emplear una estructura. De forma similar, si se desea devolver más de un valor se deben utilizar una estructura.

A continuación se muestra el contenido del archivo `date.h` que es generado por el compilador rpcgen:

```
#define DATE_PROG    ((u_long) 0x31234567)
#define DATE_VERS    ((u_long) 1)
#define BIN_DATE     ((u_long) 1)
extern  long       *bin_date_1();
#define STR_DATE     ((u_long) 2)
extern  char       **str_date_1();
```

El código del cliente:

```
/*
 *  rdate.c  - programa cliente >> servicio de tiempo
/*

#include  <stdio.h>
#include  <rpc/rpc.h>
#include  "date.h"

main(int argc, char *argv[]) {
   CLIENT *cl;     // RPC handle
   char  *server;
   long *lresult;
   char **sresult;

   if (argc ¡= 2) {
      fprintf(stderr, "Usage: %s hostname\n", argv[0]);
      exit(1);
   }
   server = argv[1];


// Creación del "handle" del cliente
   if (( cl=cln_create(server, DATE_PROG, DATE_VERS, "udp")) == NULL ) {
      // No se pudo establecer la conexión
      clnt_pcreateerror(server);
      exit(2);
   }

// Primero llamamos al procedimiento remoto "bin_ date".
   if (( lresult = bin_date_1(NULL, cl)) == NULL ) {
      clnt_perror(cl, server);
      exit(3);
   }

   printf("time on host %s = %ld\n",  server,  *lresult);

// Ahora llamamos al procedimiento remoto "str_ date".
   if (( sresult = str_date_1(lresult, cl)) == NULL ) {
      clnt_perror(cl, server);
      exit(4);
   }

   printf("time on host %s = %s",  server,  *sresult);

   clnt_destroy(cl); //  Destrucción del "handle" del cliente

   exit(0);
}
```

Inicialmente se ejecuta la llamada `clnt_create` para crear un "handle" RPC especificando el programa y su versión, el nombre del host y por último, el protocolo de

transporte (UDP). También es posible emplear TCP como protocolo de transporte especificando como último argumento “tcp”. Una vez que es creado el “handle”, es posible llamar a procedimientos remotos definidos para un programa y versión particular.

El código de las funciones del servidor:

```
/*
 *  dateproc.c  - procedimientos remotos
/*

#include  <rpc/rpc.h>
#include <time.h>
#include  "date.h"


// Devuelve la hora y fecha en formato binario
long *bin_date_1()
{
   static long timeval; // debe ser una variable estática

   timeval= time((long *) 0);
   return(&timeval);
}

// Devuelve la hora y fecha como cadena de caracteres
char **str_date_1(long *bintime)
{
   static char *ptr; // debe ser una variable estática

   ptr = ctime(bintime);
   return(&ptr);
}
```

La razón por la cual los valores de retorno de las funciones deben ser variables estáticas es que éstas deben devolver las direcciones de las variables. Si dichas variables no fueran estáticas o externas, sino que por el contrario fuesen automáticas, sus valores serían indefinidos luego que la instrucción `return` le pase el control al “stub” del servidor.

Para utilizar la aplicación antes escrita, primero se debe ejecutar el servidor. Luego desde otro o el mismo sistema se debe ejecutar el cliente. Este proceso, ilustrado en la figura 3, se describe a continuación:

1. Cuando se ejecuta el servidor en un sistema remoto, éste crea un socket UDP y lo enlaza a cualquier puerto local. Luego, llama a la función `svc_register()` de la librería RPC, para registrar el número y versión del programa. Esta función contacta al "Port Mapper" para registrase. El "Port Mapper" se encarga de manejar el número y versión de programa y el puerto. (El "Port Mapper" es usualmente iniciado como un demonio en el arranque del sistema). El servidor luego espera las solicitudes de los clientes. Es de notar que todas las acciones descritas en este paso son realizadas por el "stub" y la función `main` del servidor generados por el compilador `rpcgen`.
2. Cuando se ejecuta el programa cliente se llama a la función clnt_create(), la cual específica el sistema remoto, el número y versión de programa y el protocolo. Esta función contacta al "Port Mapper" del sistema remoto para encontrar el puerto UDP del servidor.
3. Luego, el cliente llama a la función `bin_date_1()`. Esta función está definida en el "stub" del cliente que fue generado por el compilador `rcpgen`. Dicha función envía el datagrama, utilizando el puerto UDP identificado en el paso anterior y espera la respuesta del servidor, retransmitiendo la solicitud un determinado número de veces si no se ha recibido respuesta. El datagrama es recibido por el "stub" del servidor y determina que el procedimiento ha ser llamado es la función `bin_date_1()`.
4. Cuando regresa cediéndole el control al "stub" del servidor, éste toma el valor retornado, lo convierte al formato XDR y lo incluye en un datagrama para transmitirlo de regreso al cliente. Cuando la respuesta es recibida por el "stub" del cliente, éste extrae el valor del datagrama, si es necesario realiza una conversión y lo retorna al programa cliente.

Posteriormente, el programa cliente llama a la función `str_date_1()` y se repite el mismo proceso excepto que esta función recibe una parámetro.

El "Port Mapper" del sistema remoto sólo necesita ser contactado si un nuevo "handle" de cliente es creado. Esto ocurre si un programa diferente o el mismo programa con diferente versión es llamado.

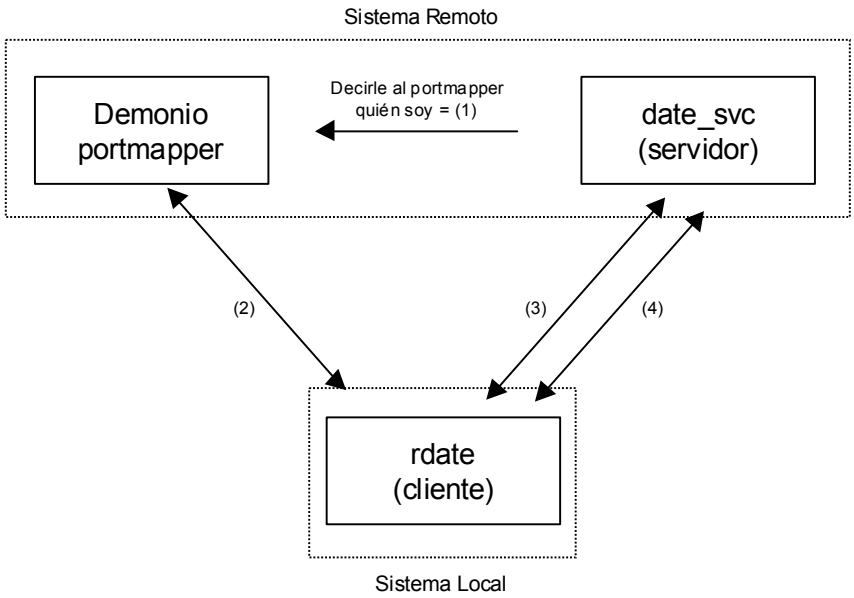

Figura 3. Sequencia de pasos de una llamada RPC

4.4. ASPECTOS DE TRANSPARENCIA DE ONC RPC

A continuación se presentan como ONC RPC maneja los aspectos relacionados con el ocultamiento de los detalles de bajo nivel.

4.1.1. Pase de Parámetros

Sólo es permitido el pase de un argumento y el retorno de un resultado. Múltiples argumentos y múltiples resultados deben estar contenidos en una estructura.

4.4.2. Enlace (Binding)

El "Port Mapper Daemon" del sistema remoto es contactado para ubicar el programa deseado. El cliente debe proveer el nombre del sistema remoto y el protocolo de transporte.

4.4.3. Protocolo de Transporte

ONC-RPC soporta tanto TCP como UDP.

4.4.4. Manejo de Excepciones

Cuando se emplea UDP, automáticamente se retransmite la solicitud cuando no se ha recibido respuesta después de un tiempo determinado. La llamada remota termina y devuelve un error al cliente luego de un número determinado de intentos fallidos. Adicionalmente, cuando se emplea TCP un código de error es devuelto al cliente cuando la conexión es terminada por el servidor.

4.4.5. Semántica de las Llamadas

La especificación de ONC-RPC establece que cada solicitud del cliente lleva consigo un identificador de transacción único. Este identificador es un entero de 32 bits denominado `xid`. Al servidor sólo le es permitido leer este valor para comprobar su igualdad. Las funciones del cliente inicializan este identificador con un valor aleatorio cuando se crea el "handle". Luego, este valor es cambiado cada vez que se realiza una solicitud. Las funciones del servidor devuelven el identificador que les fue enviado. Adicionalmente, las funciones del cliente chequean la correspondencia del identificador antes de retornar el resultado. Esto asegura que la respuesta recibida por el cliente corresponde a la solicitud realizada por éste.

Las funciones de los servidores UDP tienen una opción que les permite "recordar" las solicitudes recibidas de los clientes. Para lograr esto el servidor mantiene un "cache" de las respuestas indexadas por identificador de transacción, número de programa, versión de programa, número de procedimiento y dirección del cliente UDP. Cuando llega una solicitud, el servidor busca en su "cache" para determinar si no está duplicada. Si éste es el caso el procedimiento remoto no es llamado y la respuesta almacenada en el "cache" es retornada al cliente de nuevo. Se asume que la respuesta previa debe haberse perdido o

dañado. Esta técnica intenta implementar una semántica “al menos uno” para el protocolo UDP.

4.4.6. Representación de los Datos

El compilador `rpcgen` automáticamente genera el código requerido para referencia las funciones de conversión XDR.

4.4.7. Seguridad

ONC RPC soporta las siguientes formas de autenticación:

- Tipo UNIX
- DES (Data Encryption Standard)

Por defecto no se utiliza autenticación. La autenticación tipo UNIX requiere que los siguientes campos sean transmitidos con cada solicitud RPC:

- tiempo,
- nombre del sistema local,
- identificador efectivo de usuario,
- identificador efectivo de grupo, y
- una lista de otros identificadores de grupos a los cuales el usuario pertenece.

Luego, el servidor puede examinar estos campos y permitir o negar el procesamiento de la solicitud.

Una descripción general de la autenticación DES es provista en la RFC 1075.

5. CONCLUSIONES

Este documento ha presentado las características técnicas más importantes del mecanismo de Llamadas a Procedimientos Remotos (RPC, Remote Procedure Calls), específicamente de la implementación ONC.

RPC es una tecnología, tradicionalmente empleada en ambiente UNIX, que permite el desarrollo de sistemas de procesamiento distribuido basados en el paradigma procedimental. Con el advenimiento de implementaciones para plataforma Windows, así como para Java, se concibe a RPC como una tecnología de integración entre plataformas disimiles de hardware y software.

6. BIBLIOGRAFÍA

1. W. Richard Stevens. “UNIX Network Programming” . Prentice Hall. 1990.
2. John Bloomer. “Power Programming With RPC”. O'Reilly & Associates. March.1992.
3. Andrew S. Tanenbaum. "Sistemas Operativos Distribuidos". Prentice-Hall. 1996.
4. R. Srinivansan. “ XDR: External Data Representation Standard”. Sun Microsystems. August. 1995.
5. R. Srinivansan. “ RPC: Remote Procedure Call Specification Version 2”. Sun Microsystems. August. 1995.