Sistemas distribuidos

Tema 2

Comunicación entre procesos: RPC

Fco. Javier Ramírez
DIATEL-UPM,2003

Introducción

- RPC
  - Llamada a un procedimiento remoto (Remote Procedure Call).
  - Birrel y Nelson (1984)
- Objetivos
  - Que los programas puedan invocar procedimientos localizados en otras máquinas (proced. remotos).
  - Transparencia de acceso: invocar procedimiento remoto de forma similar que una llamada a procedimiento local.
  - Transparencia (parcial) de ubicación: el cliente ya no conoce todos los datos de ubicación del servidor.

Cuestiones que resuelve

- Transferencia de parámetros y resultados:
  - Automatiza generación del código de serialización/deserialización
- Localización (parcial) del proceso servidor.
- Procesos C y S en espacios de direcciones diferentes.
- Arquitecturas diferentes
  - Se basa en XDR
- Fiabilidad
  - Fallos en cliente o en servidor
  - Fallos en canal de comunicación
  - ¿servidor ha llegado a ejecutar la operación?
- Seguridad
  - Mayor en RPC seguras (Secure-RPC)

Plataformas RPC

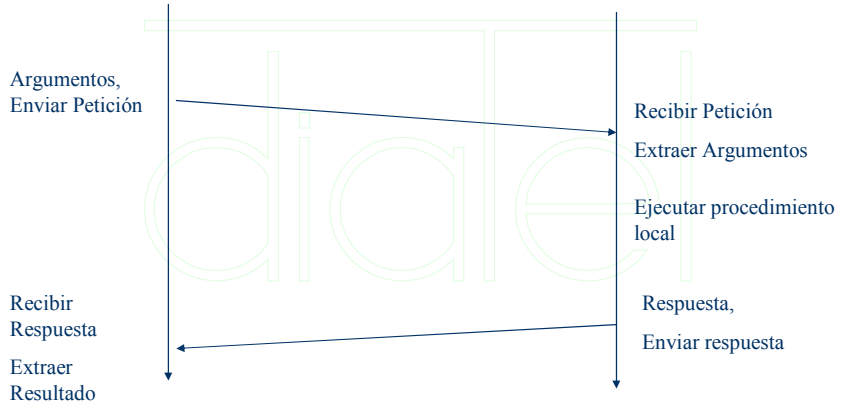
- ONC RPC
  - Antiguas Sun RPC/XDR
  - RFC 1831 (ONC RPC), RFC 1832 (XDR)
- OSF/DCE
  - Open Software Foundation/Distributed Computing Environment
  - Varios modelos de RPC y tipos de representación de datos y protocolos de red.
  - Servicios adicionales: directorio, tiempo, seguridad...
- Evolución:
  - Sirve de base a posteriores arquitecturas de objetos distribuidos: JAVA RMI, OMG CORBA.

ONC RPC

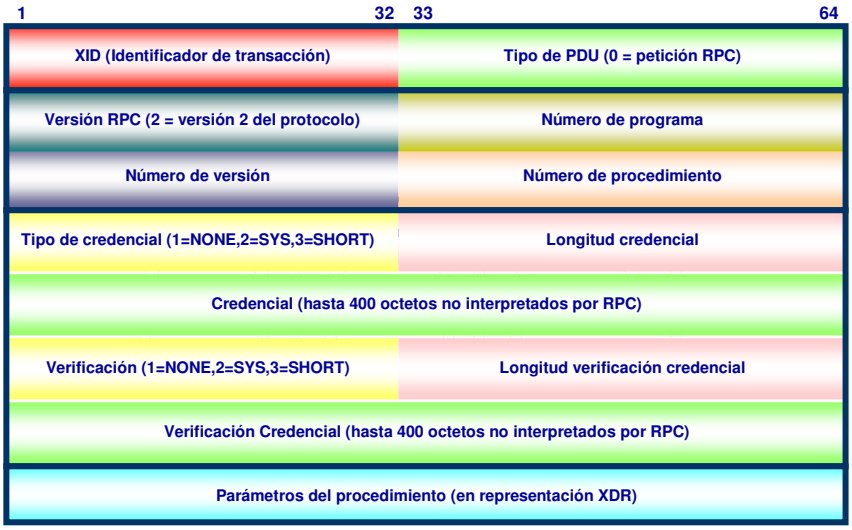
- Estudiamos esta implementación (la original).
  - Protocolo: descripción de pdus RPC
  - Representación externa de datos: XDR
  - Especificación de la interfaz del servicio (servicio.x)
  - Automatización de la generación de código a partir de la interfaz del servicio.
    - concepto de stub/skeleton
  - Ejemplo: servicio restador (restador.x)
  - Servicio de nombres (portmapper ó rpcbind)
  - Fiabilidad: comportamiento ante fallos
  - Seguridad: autenticación débil
  - Concurrencia: multihilo y multiproceso en el servidor.

Protocolo RPC

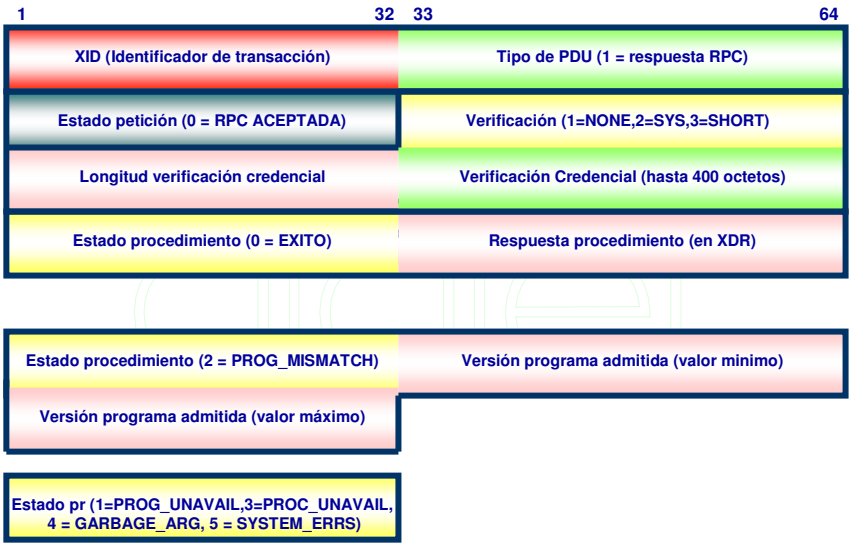
- Nos centramos en RPC versión 2
- Cliente Servidor



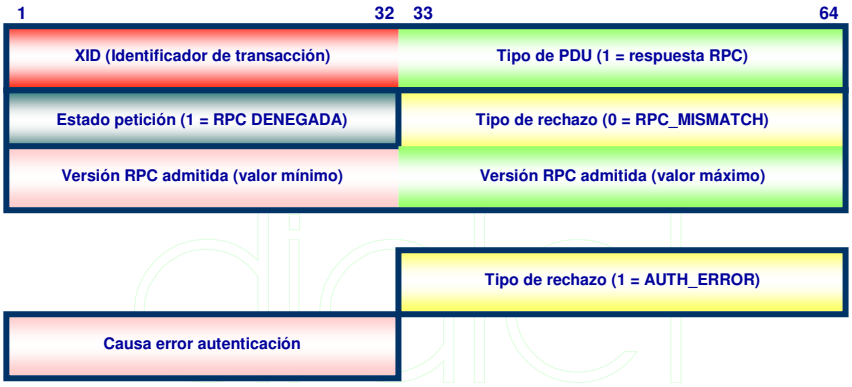
Protocolo: PDU petición RPC



Protocolo: PDU respuesta RPC



Protocolo: PDU respuesta RPC



Protocolo: marcado de mensajes

- Cuando RPC se apoya en protocolos de tipo flujo (ej. TCP), marca el mensaje.
  - Da estructura: separa un mensaje del siguiente.
- La petición/respuesta se envía en N fragmentos.
  - Cada fragmento lleva 32 bits por delante:
    - 1 bit: indica si es el último fragmento (1=ultimo)
    - 31 bits siguientes: indican la longitud del fragmento ($0..2^{31}-1$)

Representación externa de datos XDR

```
integer i; short s;  /* ENTERO CON SIGNO */
unsigned integer i;  /* ENTERO SIN SIGNO */
enum Color { BLANCO=0, NEGRO=1 };  Color c; /*ENUMERADO*/
bool b;  /* BOOLEANO: TRUE,FALSE */

(MSB)                   (LSB)
+------+------+------+------+
|byte 0|byte 1|byte 2|byte 3|
+------+------+------+------+
<------------32 bits-------->

hyper integer hi; /* ENTERO LARGO */
(MSB)                                                   (LSB)
+------+------+------+------+------+------+------+------+
|byte 0|byte 1|byte 2|byte 3|byte 4|byte 5|byte 6|byte 7|
+------+------+------+------+------+------+------+------+
<----------------------------64 bits---------------------->
```


Representación externa de datos

```
float f; /* NUM. EN COMA FLOTANTE, PRECISIÓN SIMPLE */
+-------+-------+-------+-------+
|byte 0 |byte 1 |byte 2 |byte 3 |
S|   E   |       F               |
+-------+-------+-------+-------+
1|<- 8 ->|<-------23 bits------>|
<------------32 bits------------>
string s<200>;  /* STRING de hasta 200 octetos */
opaque o<300>;  /* OPACO -sec. octetos- hasta 300 octetos*/
   0    1    2    3    4     5   ...
+----+----+----+----+-----+-----+...+-----+-----+...+-----+
|      length n     |byte0|byte1|...| n-1 |  0  |...|  0  |
+----+----+----+----+-----+-----+...+-----+-----+...+-----+
|<-----4 bytes----->|<------n bytes------>|<---r bytes--->|
                    |<----n+r (donde (n+r) mod 4 = 0)---->|
```

Representación externa de datos

```
struct Libro {
   tipo1 campo1;
   tipo2 campo2;
   ...
};
Libro l; /* Estructura */
   * Se representa como la suma de sus componentes

union Respuesta switch (int estado) {
case 1:
   tipo campo;
case 2:
   tipo campo;
default:
   tipo campo;
};
Respuesta r; /* Unión con discriminante*/
   * Se representa: discriminante (4 bytes) + campo seleccionado
```


Interfaz de un servicio

- Interfaz: contrato de acceso a un servicio
  - los clientes la usan, los servidores la ofrecen.
- Definición de la interfaz de un servicio
  - Cada servicio: tiene un número de programa único.
  - Cada programa: tiene versiones (1..n)
  - Cada versión: tiene procedimientos (1..n)
  - Cada procedimiento: un argumento y un resultado (definidos en nomenclatura XDR –similar a C--).
- Se define en un fichero con extensión ".x"

Ejemplo: interfaz servicio restador

- Fichero “resta.x”, define la interfaz del servicio:

```
/* Argumentos: XDR */
struct peticion {
   int op1;
   int op2;
};
/* Definición del programa */
program RESTA {
    version RESTAv1 {
            int restar(peticion) = 1;
    } = 1;
} = 0x20000001;
```


Otros detalles de la interfaz

- Rango de numeración de los programas

```
0 - 1fffffff definido por rpc@sun.com
20000000 - 3fffffff definido por usuario
40000000 - 5fffffff transient
60000000 - 7fffffff reserved
80000000 - 9fffffff reserved
a0000000 - bfffffff reserved
c0000000 - dfffffff reserved
e0000000 - ffffffff reserved
```

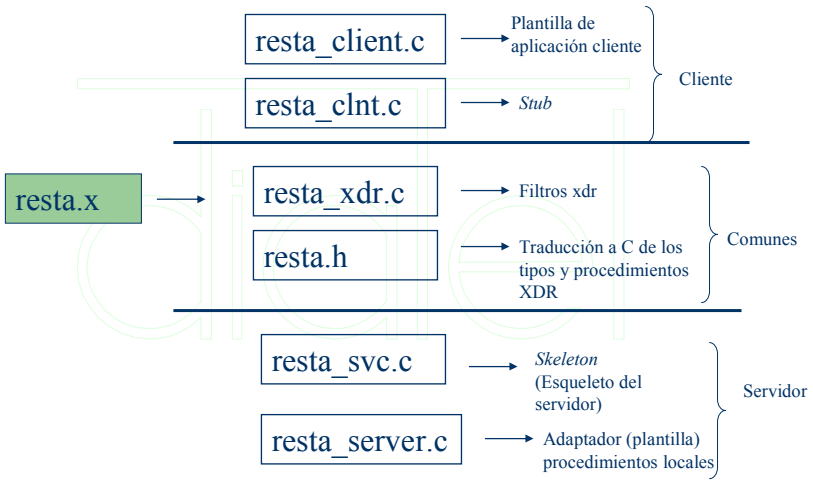
  - Grupo 1: números fijos (solicitar a rpc@sun.com)
  - Grupo 2: números temporales (desarrollo). Cuando pasa a ser fijo, debería solicitar número en grupo 1.
  - Grupo 3: números de programa generados automáticamente por herramientas.

Compilador rpcgen

- Compilador de interfaces de servicio RPC
- Genera:
  - *Stub* del cliente: código de serialización/deserialización XDR del cliente + uso adaptado de la biblioteca de funciones RPC.
  - *Skeleton* (esqueleto) del servidor: código de gestión de un servidor multiproceso/multihilo que atiende operaciones (serialización/deserialización + uso adaptado de la biblioteca de funciones RPC)
  - Filtros para la conversión de representación local a XDR y viceversa.
    - serialización (*marshalling*)
  - Ficheros de cabecera específicos para el servicio que permiten utilizar el código generado.

Salida del rpcgen

- rpcgen resta.x



resta.h

```
#ifndef _RESTA_H_RPCGEN
#define _RESTA_H_RPCGEN
#include <rpc/rpc.h>
struct peticion {
  int op1;
  int op2;
};
#define PROG_RESTA ((u_long)0x20000001)
#define VER_RESTAv1 ((u_long) 1)
#define PROC_RESTAR ((u_long) 1)
extern int * proc_restar_1(peticion *, CLIENT *);
extern int * proc_restar_1_svc(peticion *, struct
  svc_req *);
...

#endif /* _RESTA_H_RPCGEN */
```


resta_server.c

```
#include “resta.h"
int * proc_restar_1_svc(peticion *argp,
                        struct svc_req *rqstp)
{
  static int result;

  //codigo especifico del procedimiento
  result = (argp->op1 - argp->op2);

  return(&result);
}
```

resta_client.c

```
#include “resta.h"
int main( int argc, char* argv[] )
{
   CLIENT *clnt;
   int *res;
   peticion proc_restar_1_arg;
   char *host;

   if (argc < 2) {
       printf("uso: %s servidor\n", argv[0]); exit(1);
   }
   /* localiza al servidor */
   host = argv[1];
   clnt = clnt_create(host, PROG_RESTA,
                      VER_RESTAv1, "udp");
```


resta_client.c

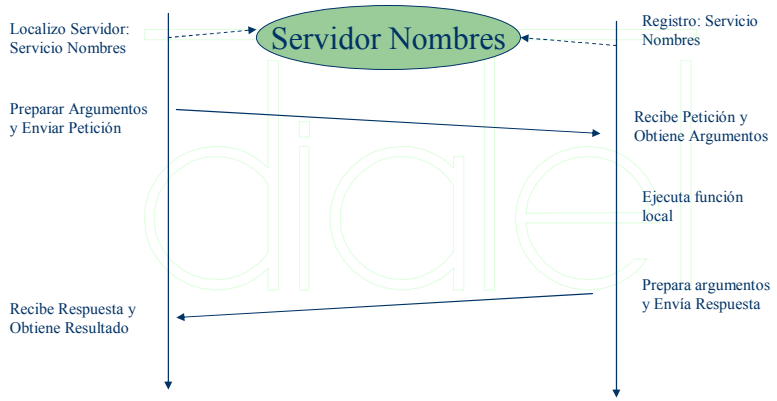
```
   if (clnt == NULL) {
       clnt_pcreateerror(host); exit(1);
   }
   proc_restar_1_arg.op1 = 5;
   proc_restar_1_arg.op2 = 2;
   res = proc_restar_1(&proc_restar_1_arg, clnt);
   if (res == NULL) {
       clnt_perror(clnt, “invocación RPC fallida:");
   }
   printf(“El resultado es %d\n", *res);
   clnt_destroy( clnt );
}
```

Servicio de nombres

- Logra transparencia (parcial) de ubicación del procedimiento remoto.
  - Cliente inicialmente no conoce parte de la ubicación del servidor.
- El Servicio de Nombres mantiene una tabla de traducción entre nombres de servicio y direcciones.
- Funciones:
  - Registrar un nombre de servicio.
  - Eliminar un nombre de servicio.
  - Entregar la dirección de un nombre de servicio.
- Localización del Servidor del servicio de nombres
  - El servicio de nombres es otro caso cliente-servidor
  - El servidor de un Servicio de Nombres suele tener una dirección conocida.

Servicio de nombres

- Cliente Servidor



Servicio de nombres

- En ONC RPC el servidor se ejecuta en un puerto registrado y conocido: 111 (TCP/UDP)
- Generalmente llamado “portmapper” o “rpcbind”
- Mantiene una tabla con:
  - **Nombre del servicio RPC** (Nº programa, Nº versión) junto a su **ubicación** (protocolo -UDP/TCP- y puerto).
  - La ubicación no incluye dirección IP
    - el servicio de nombres registra sólo los servicios RPC que se sirven en su misma máquina.
- El cliente sólo necesita conocer la máquina servidora.

Servicio de nombres

```
const PMAP_PORT = 111;     /* portmapper port number */
/*  * A mapping of (program, version, protocol) to port number
 */
struct pmap {
  unsigned int prog;
  unsigned int vers;
  unsigned int prot;
  unsigned int port;
};
/*  * Supported values for the "prot" field  */
const IPPROTO_TCP = 6; /* protocol number for TCP/IP */
const IPPROTO_UDP = 17; /* protocol number for UDP/IP */
/*  * A list of mappings  */
struct pmaplist {
  pmap map;
  pmaplist *next;
};
```

Servicio de nombres

```
/*  * Arguments to callit  */
struct call_args {
  unsigned int prog;
  unsigned int vers;
  unsigned int proc;
  opaque args<>; };
/* * Results of callit */
 struct call_result {    unsigned int port;    opaque res<>; };
/* * Port mapper procedures */
program PMAP_PROG {
  version PMAP_VERS {
      bool          PMAPPROC_SET(pmap) = 1;
      bool          PMAPPROC_UNSET(pmap) = 2;
      unsigned int  PMAPPROC_GETPORT(pmap) = 3;
      pmaplist      PMAPPROC_DUMP(void) = 4;
     call_result    PMAPPROC_CALLIT(call_args) = 5;
   } = 2;
 } = 100000;
```


Fiabilidad en las RPC

- Problemas que pueden plantearse con RPCs
  - No es posible localizar el servidor
    - Sol: Devolver error o excepción al solicitante del proc.
  - Pérdida del mensaje de petición
    - Sol: Retransmitirlo (cliente rpc establece temporizador)
  - Pérdida del mensaje de respuesta
    - Depende de si operaciones es idempotentes o no.
      - Operaciones idempotentes. Cuando si se repiten no ocasionan problemas.
    - Sol. idempotentes: retransmitir la petición.
    - Sol. no idempotentes: que el servidor descarte peticiones ya ejecutadas:
      - número de secuencia en cada petición
      - Una marca en cada petición que indique si es original o una retransmisión.

Fiabilidad en las RPC

- Se cae el servidor en el transcurso de una operación
  - Dos situaciones posibles:
    - A) El servidor no ha llegado a ejecutar la operación.
    - B) El servidor sí ha llegado a ejecutar la operación.
  - El cliente no sabe si A) o B). ¿Qué hace?
    - No garantiza nada.
    - Garantiza “al menos una vez”: retransmite la petición (no funciona para no idempotentes)
    - Garantiza “como mucho una vez”: no retransmite.
    - Garantiza “exactamente una vez”: lo deseable.
- Se cae el cliente en el transcurso de una operación.
  - El servidor sigue procesando: la operación puede haber sido realizada.
  - El cliente reinicia: cuidado al relanzar la RPC (números de secuencia, etc. -> posibles conflictos con RPC anteriores o en curso.)

Seguridad

- Dos campos en la petición RPC
  - Credencial
  - Verificación
- Un campo en la respuesta RPC
  - Verificación
- Son campos “opacos” para las RPC durante su transporte. La biblioteca de funciones RPC permite rellenarlos y verificarlos en cliente y en servidor.