Sistemas Distribuidos

Módulo 2

Comunicación en Sistemas Distribuidos

Agenda

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta: Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

Llamadas a Procedimiento Remoto (RPC)

El modelo RPC

Es similar al bien conocido y entendido modelo de llamadas a procedimientos usado para transferir control y datos.

El mecanismo de RPC es una extensión del anterior porque habilita a hacer una llamada a un procedimiento que no reside en el mismo espacio de direcciones.

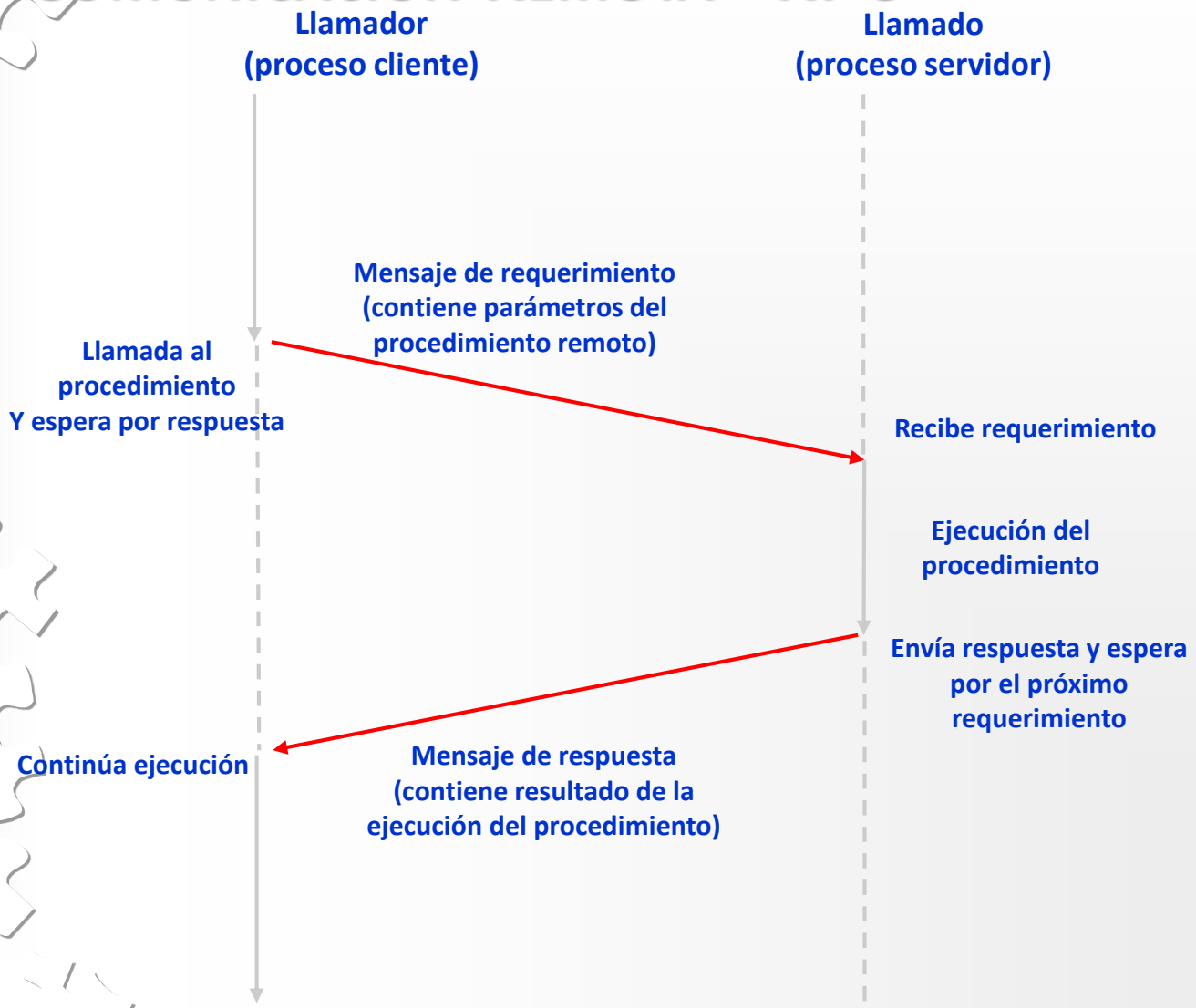
Comunicación Remota - RPC

La facilidad de RPC usa un esquema de pasaje de mensajes para intercambiar información entre los procesos ***llamador*** (proceso cliente) y ***llamado*** (proceso servidor).

Normalmente el proceso servidor *duerme*, esperando la llegada de un mensaje de requerimiento.

El proceso cliente se bloquea cuando envía el mensaje de requerimiento hasta recibir la respuesta.

Comunicación Remota - RPC



Comunicación Remota - RPC

Transparencia de RPC

Transparencia SINTÁCTICA: una llamada a procedimiento remoto debe tener la misma sintaxis que una llamada local.

Transparencia SEMÁNTICA: la semántica de un RPC es la misma que para una llamada local.

Diferencias:

a) espacio de direcciones

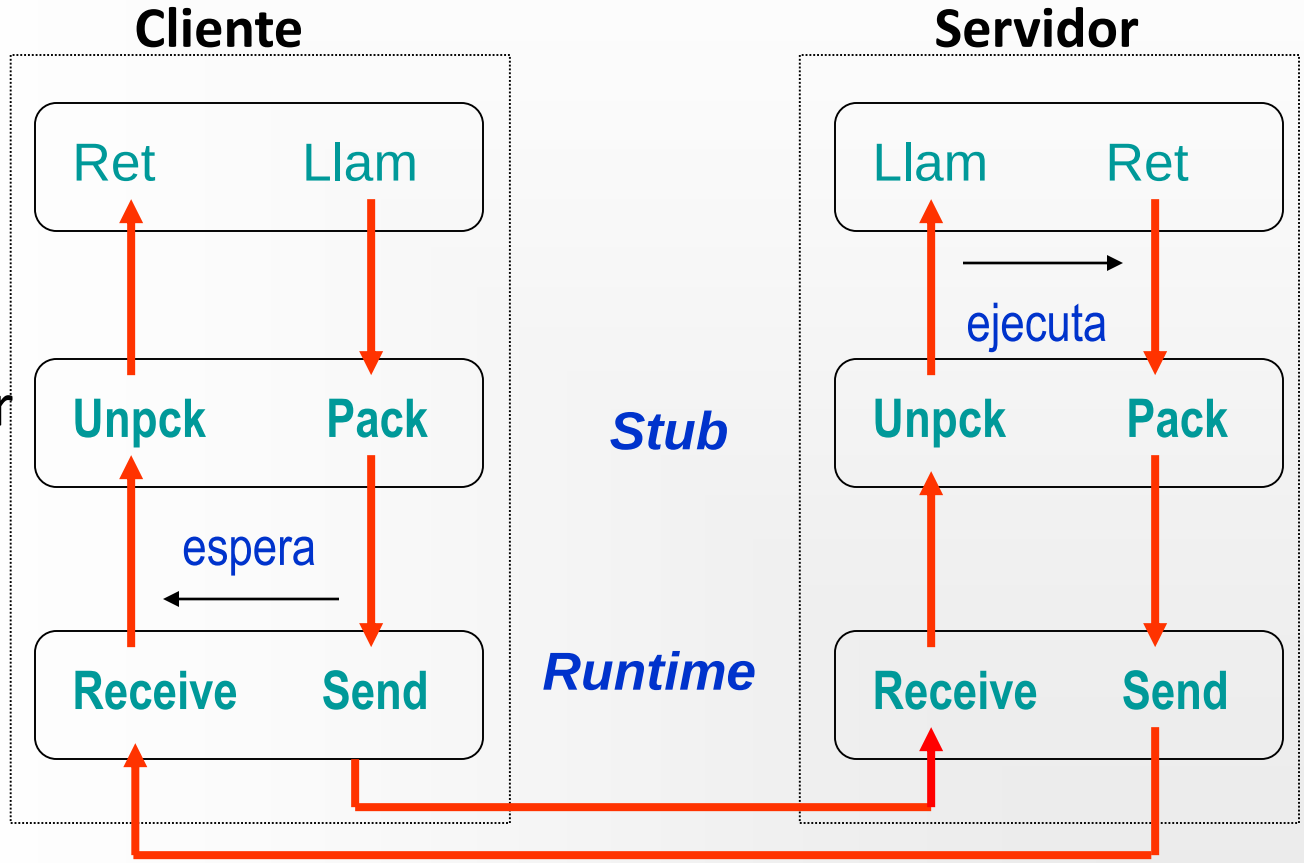
b) fallas

c) Consumen más tiempo

Comunicación Remota - RPC

Implementación del mecanismo de RPC

- El cliente
- El *stub* cliente
- El runtime RPC
- El *stub* servidor
- El servidor



Comunicación Remota - RPC

Cliente

Es el que inicia el RPC. Hace una llamada que invoca al *stub*.

Stub cliente

Realiza las siguientes tareas:

a) Empaqueta la especificación del procedimiento objetivo y sus argumentos en un mensaje y pide al *runtime* local que lo envíe al ***stub*** servidor

b) En la recepción de los resultados de la ejecución del proceso, desempaqueta los mismos y los pasa al cliente.

Comunicación Remota - RPC

Runtime RPC

Maneja la transmisión de mensajes a través de la red entre las máquinas cliente y servidor.

Stub servidor

Trabaja en forma simétrica a como lo hace el *stub* cliente.

Servidor

Cuando recibe un requerimiento de llamada del *stub* servidor, ejecuta el procedimiento apropiado y retorna el resultado de la misma al *stub* servidor.

Comunicación Remota - RPC

Mensajes de RPC

Mensaje de llamada

Mensaje ID	Mensaje tipo	Cliente Id	Procedimiento Remoto Id			Argumentos
			Prog. Nro.	Ver. Nro.	Proc.Nro.	

Mensaje de respuesta

Mensaje Id	Mensaje tipo	Respuesta Estado (éxito)	Resultado

Mensaje Id	Mensaje tipo	Respuesta Estado (no exitoso)	Razón de la falla

Comunicación Remota - RPC

Administración del Servidor

1.- Implementación

- Servidor con estado: mantiene información de un cliente de una llamada a procedimiento remoto a la próxima llamada.
- Servidor sin estado: no mantiene información de estado de un cliente.

¿cuál es mejor?

Comunicación Remota - RPC

Administración del Servidor

2.- Semántica de creación

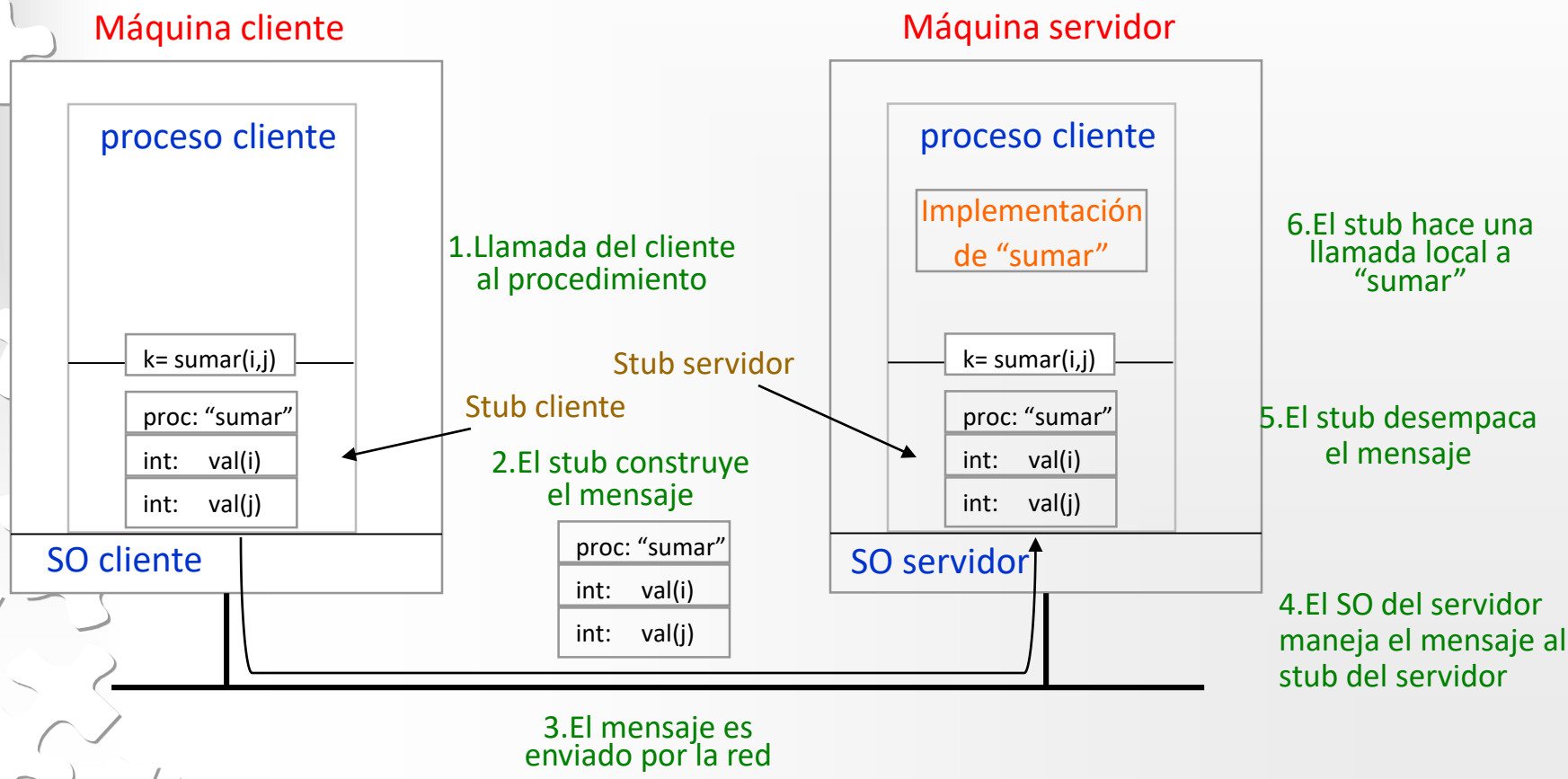
- Instancia por llamada al servidor
- Instancia por sesión con el servidor
- Persistente

3.- Pasaje de parámetros

- por valor
- por referencia

Comunicación Remota - RPC

Pasaje de Parámetros por Valor



Comunicación Remota - RPC

Semántica de la llamada

1.- Pudiera ser (may be)

2.- Al menos una vez (at-least-once)

3.- A lo sumo una vez (at-most-once)

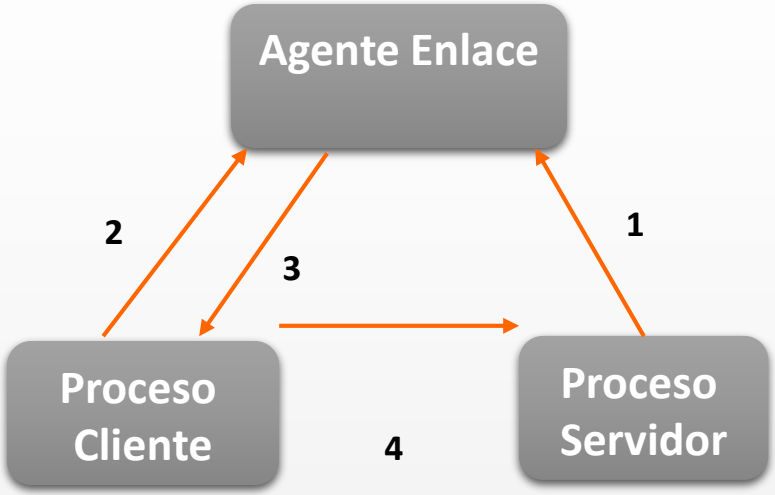
4.- Exactamente una vez

Enlace de un Cliente y un Servidor en RPC

1.- Nombre Servidor (Servicio)

2.- Ubicación Servidor

- Broadcasting
- Agente de enlace (binding)
  - Ubicación conocida
  - Ubicación desconocida



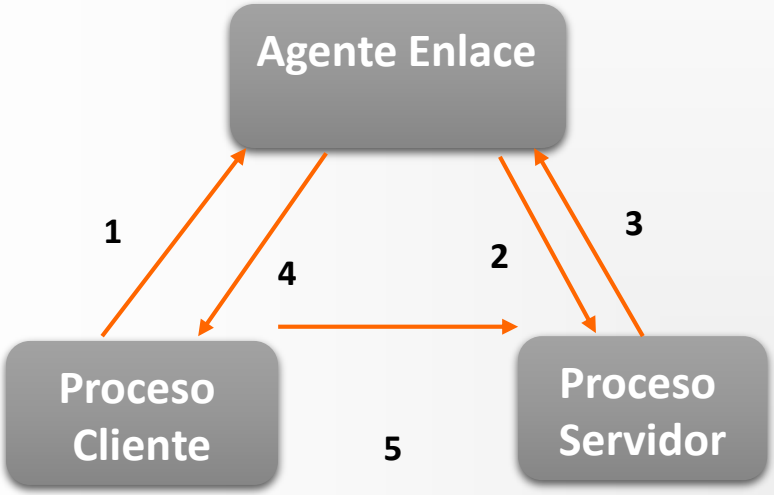
3.- Tiempo de enlace

- compilación.
- link (requiere un manejador o canal para conectarse el cliente con el servidor).
- llamada (ocurre en la primer llamada)

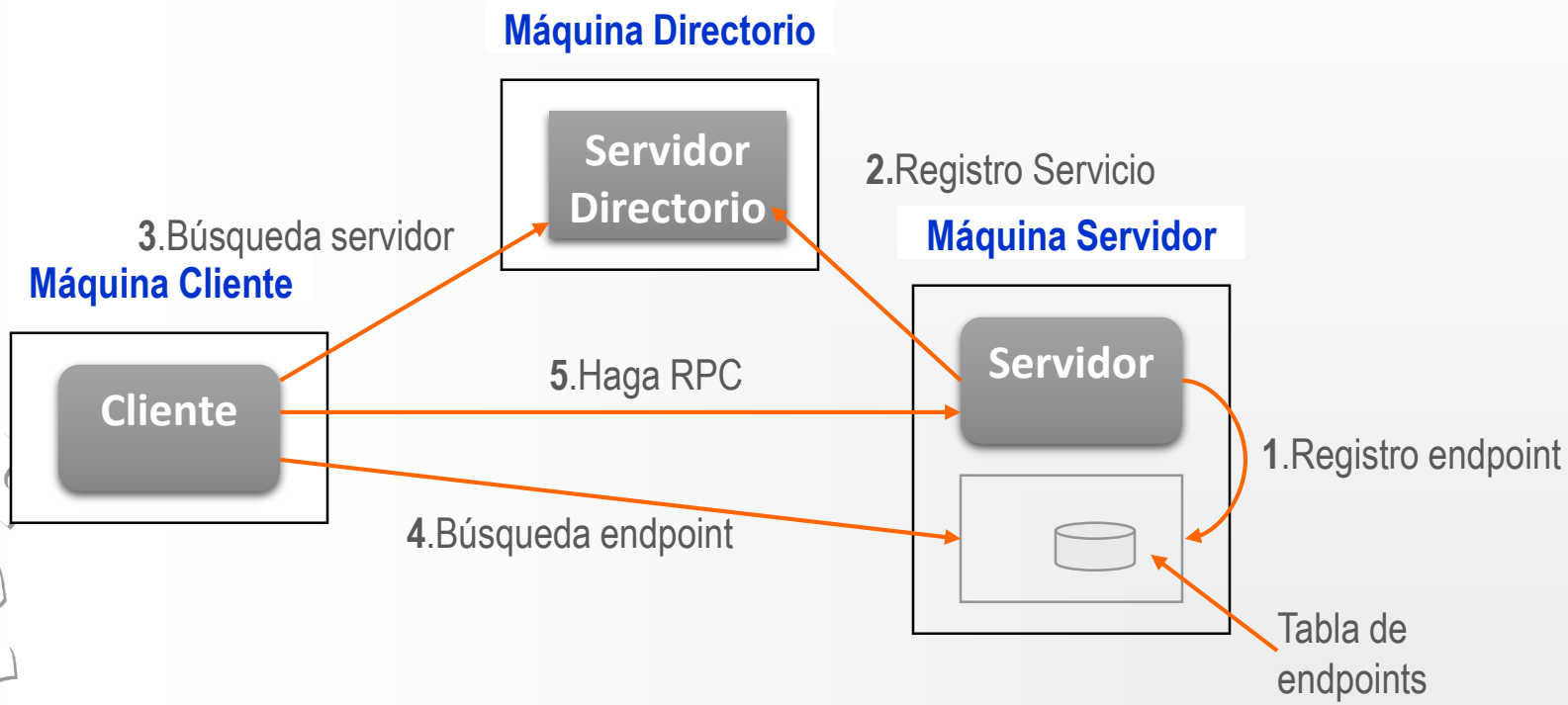
Enlace de un Cliente y un Servidor en RPC

Vinculación en el tiempo de una llamada.

Utilización del método de llamada indirecta



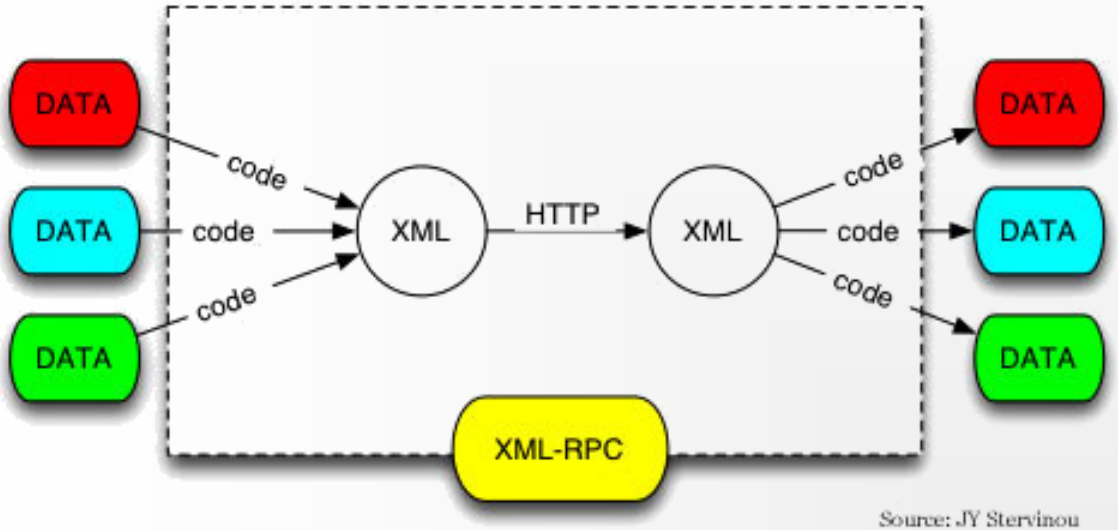
Enlace de un Cliente y un Servidor en RPC



Comunicación Remota - RPC

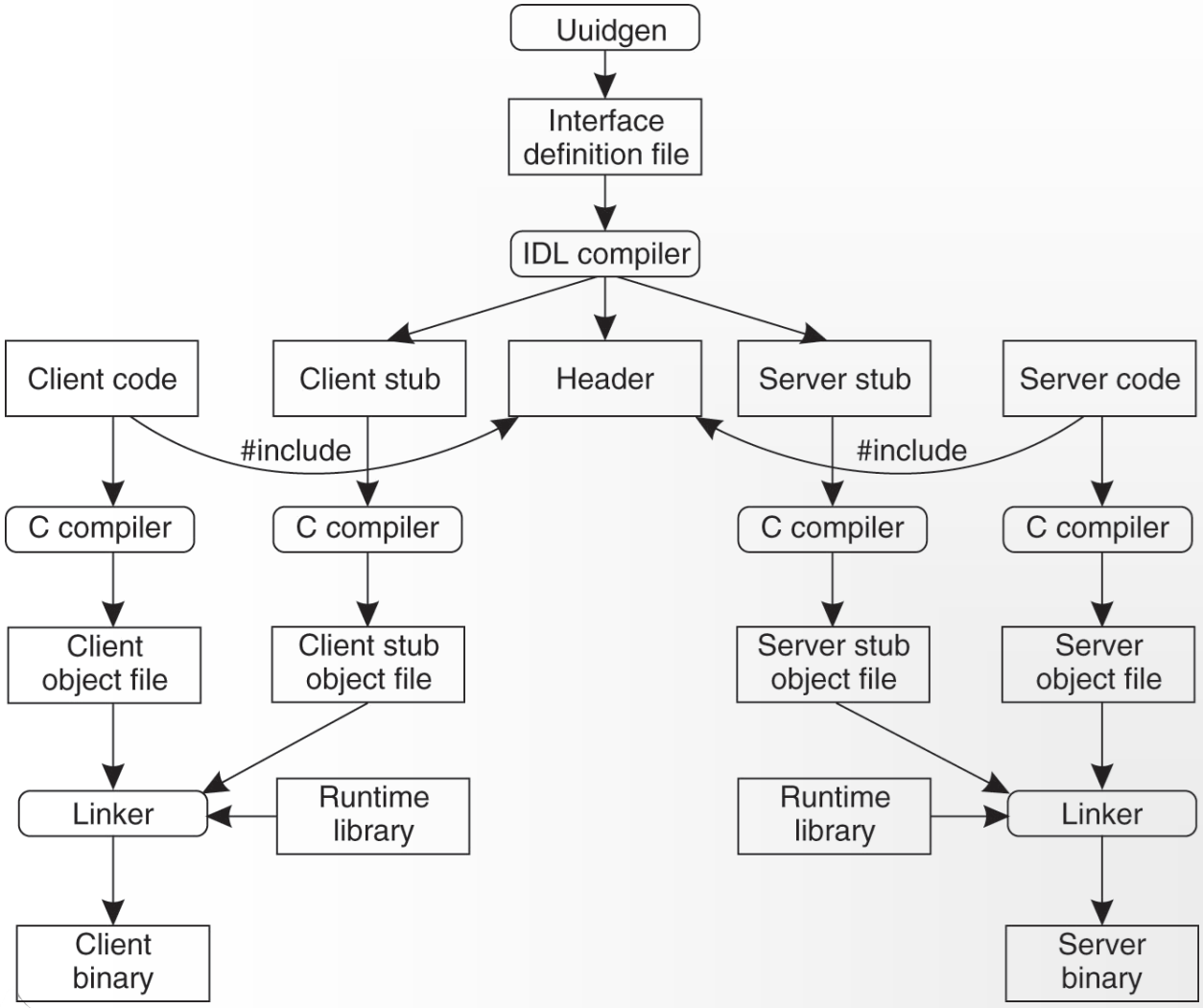
- ONC RPC – Sun RPC
- DCE RPC
- XML-RPC

Comunicación Remota - RPC



Source: JY Stervinou

Comunicación Remota - RPC

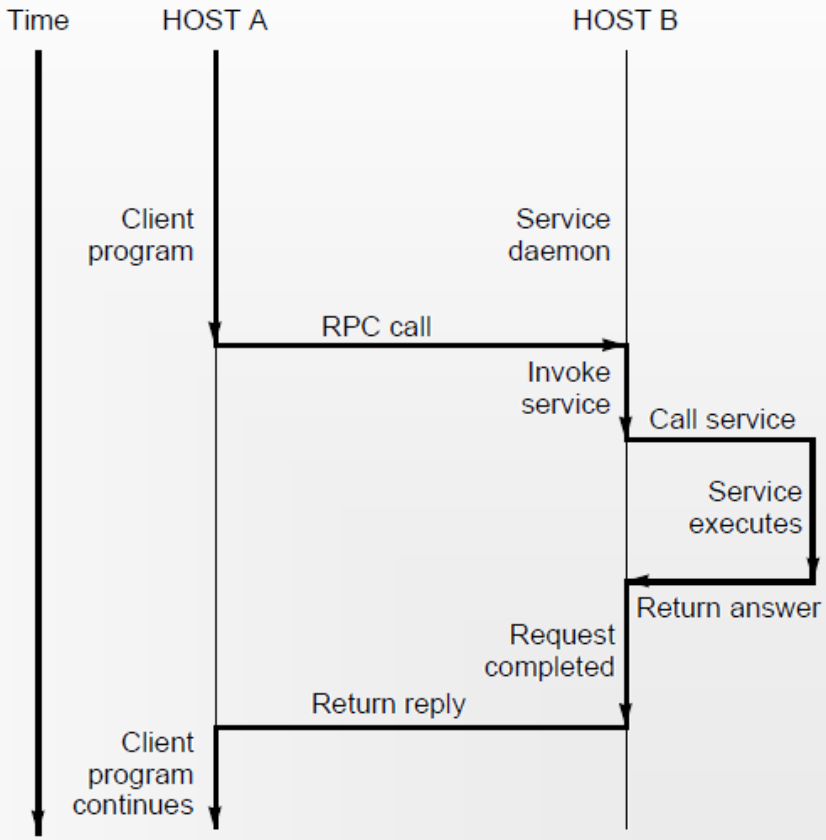


Sun RPC - Implementación

Comunicación Remota - Sun RPC

El RPC representa una llamada a una función.

Mantiene la *semántica* de una llamada local, lo que difiere es la *sintaxis*.



Comunicación Remota - Sun RPC

Pasos para desarrollar una aplicación RPC

- **Especificar el protocolo de comunicación del cliente y servidor.**
- **Desarrollar el programa del cliente**
- **Desarrollar el programa del servidor**

Los programas serán compilados en forma separada. El protocolo de comunicación es armado mediante *stubs* y estos stubs y rpc (y otras librerías) necesitarán ser linkeadas

COMUNICACIÓN REMOTA - Sun RPC

1.- Definir el protocolo

- utilizar un protocolo compiler como ***rpcgen***
- para el protocolo se debe especificar:
  - Nombre del procedimiento del servicio
  - Tipo de dato de los parámetros de E/S

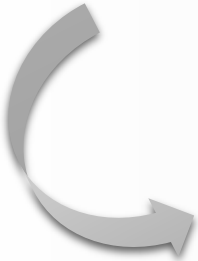
El protocolo compiler lee una definición y automáticamente genera los *stubs* del *cliente* y *servidor*

Comunicación Remota - Sun RPC

- ***rpcgen***: utiliza su propio lenguaje (RPC lenguaje o RPCL) el cual es similar a las directivas preprocesador.
- ***rpcgen*** existe como un compiler que lee archivos especiales que tienen extensión .x

Para compilar un RPCL:

rpcgen rpcprog.x



Genera cuatro posibles archivos

Comunicación Remota - Sun RPC

- **rpcprog_clnt.c** → el stub del cliente
- **rpcprog_svc.c** → el stub del servidor
- **rpcprog_xdr.c** → si es necesarioXDR (external data representation) filters
- **rpcprog.h** → el encabezado del archivo necesitado por cualquier XDR filters

Comunicación Remota - Sun RPC

2.- Definir el código de Aplicación del Cliente y Servidor

- El código del cliente se comunica vía procedimientos y tipos de datos especificados en el protocolo.
- El servidor tiene que ***registrar*** los procedimientos que van a ser invocados por el cliente y recibir y devolver cualquier dato requerido para el procesamiento.

El cliente y el servidor deben incluir el archivo *"rpcprog.h"*

Comunicación Remota - Sun RPC

3.- Compilar y Ejecutar la Aplicación

Compilar el código del Cliente:

gcc –c rpcprog.c

Compilar el stub del Cliente

gcc –c rpcprog_clnt.c

Compilar el XDR filtro

gcc –c rpcprog_xdr.c

Construir el ejecutable del Cliente

gcc –lnsl –o rpcprog rpcprog.o rpcprog_clnt.o rpcprog_xdr.o

Comunicación Remota - Sun RPC

3.- Compilar y Ejecutar la Aplicación – Cont.

Compilar los procedimientos del Servidor:

gcc –c rpcsvc.c

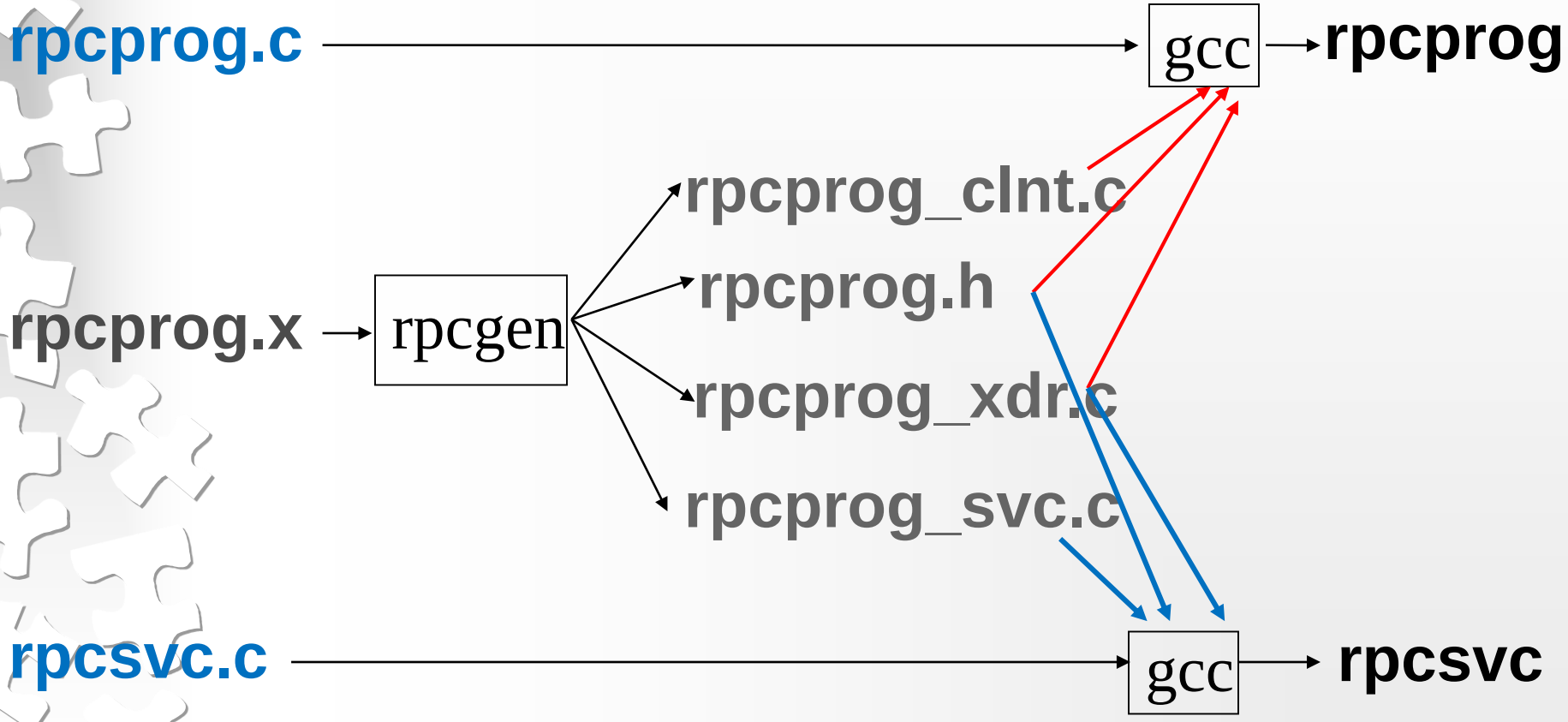
Compilar el stub del Servidor:

gcc –c rpcprog_svc.c

Construir el ejecutable del Servidor:

gcc –lnsl –o rpcsvc rpcsvc.o rpcprog_svc.o rpcprog_xdr.o

Comunicación Remota - Sun RPC



COMUNICACIÓN REMOTA - Sun RPC

Formato General Protocolo

/* **proto.x** protocolo para … */

Definición de constantes

…

Definición de tipos

…

program **NOMBREPROGRAMA**{

version **VERSIONPROGRAMA**{

tiporesultadofunción1 **FUNCION1**(tipoparámetro1) =1;

tiporesultadofunción2 **FUNCION2**(tipoparámetro2) =2;

tiporesultadofunción3 **FUNCION3**(tipoparámetro3) =3;

…

} = 1;

} = 0x20000121;

```
#define TAM_BLOQUE 512

Struct  atributos
{
   char nombre[512];
   int   modo;
};
```

Sun RPC – Ejemplo: Mensaje Remoto

Definición del protocolo

```
/* msg.x remote msg printing protocol */

program MESSAGEPROG{
    version PRINTMESSAGEVERS{
            int PRINTMESSAGE(string) =1;
    } = 1;
} = 0x20000001;
```

Identificación del programa

Identificación de la versión

Identificación del procedimiento

Los números de programa están definidos en forma standard:

- 0*x*00000000 - 0*x*1*FFFFFFF*: Defined by Sun
- 0*x*20000000 - 0*x*3*FFFFFFF*: User Defined
- 0*x*40000000 - 0*x*5*FFFFFFF*: Transient
- 0*x*60000000 - 0*xFFFFFFFF*: Reserved

Sun RPC – Ejemplo: Mensaje Remoto

Definición del Servicio

```
/* msg_proc.c: implementación del procedimiento remoto */
 #include <stdio.h>
#include "msg.h" /* msg.h generado por el rpcgen */
 int *printmessage_1(msg,req)
   char **msg;
   struct svc_req *req; /*detalle de la llamada */
 {
   static int  result; /*debe ser estático*/
   FILE  *f;
   f = fopen("/dev/console" ,"w");
   if (f == (FILE *)NULL) {
                result = 0;
                return(&result);
   }
   fprintf(f, "%s \n", *msg);
   fclose(f);
   result = 1;    return(&result); }
```

Definición de datos
Y funciones

Función o procedimiento remoto

En linux se denominará
***printmessage_1_svc**

Sun RPC – Ejemplo: Mensaje Remoto

Definición del Cliente

```
/* rprintmsg.c: versión remota de "printmsg.c" */
 #include <stdio.h>
#include "msg.h"  /* msg.h generado por rpcgen */
main (argc, argv)
   int argc;
   char *argv[];
{
   CLIENT  *clnt;
   int   *result;
   char  *server;
   char  *message;
   if (argc != 3) {
                fprintf(stderr, "usage: %s host message \n", argv[0]);
                exit(1);
   }
   server = argv[1];
   message = argv[2];
```

Variable utilizada para Establecer la comunicación

Nombre de la máquina

Sun RPC – Ejemplo: Mensaje Remoto

```
....
/* Create client "handle" used for calling MESSAGEPROG on the server */
   /* designated on the command line */
     clnt = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERS, "visible");
    if (clnt == (CLIENT *) NULL) {
                /* Couldn't establish connection with server. Print error message */
                /* and die */
                clnt_pcreateerror(server);
                exit(1);
   }
   /* Call the remote procedure "printmessage" on the server */
   result = printmessage_1(&message, clnt);
   if (result == (int *) NULL) {
                /* An error occurred while calling the server. Print error message */
                /* and die */
                clnt_perror (clnt, server);
                exit(1);
   }
```

Establecer conexión Con el servidor

Para seleccionar el protocolo de comunicación. TCP / UDP

Invocación al procedimiento remoto

Sun RPC – Ejemplo: Directorio Remoto

Definición del Protocolo

```
/* dir.x: protocolo para listar un directorio remoto. Este ejemplo muestra las funciones de rpcgen*/
const MAXLENNAME = 255; /* max longitud de la entrada del directorio */
typedef string nametype <MAXLENNAME>; /* directory entry */
typedef struct namenode *namelist; /* link in the listing */
struct namenode {
   nametype name; /* name of the directory entry */
   namelist next; /* next entry */
};
union readdir_res switch (int errno) {
   case 0:
                namelist list; /* no error: return directory listing */
   default:
                void; /* error ocurred: nothing else to return */
};
 program DIRPROG{
   version DIRVERS{
                readdir_res READDIR(nametype) = 1;
   } = 1;
} = 0x20000076;
```

SUN RPC – EJEMPLO: DIRECTORIO REMOTO

Dir.h
(contenido generado Por rpcgen en Solaris)

```
/* Please do not edit this file.  It was generated using rpcgen. */
#ifndef _DIR_H_RPCGEN
#define     _DIR_H_RPCGEN
#include <rpc/rpc.h>
#define     MAXLENNAME 255
typedef char *nametype;
typedef struct namenode *namelist;
struct namenode {
        nametype name;
        namelist next;
};
typedef struct namenode namenode;
struct readdir_res {
        int errno;
        union {
                namelist list;
        } readdir_res_u;
};
#define     DIRPROG ((unsigned long)(0x20000076))
#define     DIRVERS ((unsigned long)(1))
#define     READDIR ((unsigned long)(1))
extern  readdir_res * readdir_1();
extern int dirprog_1_freeresult();

/* the xdr functions */
extern bool_t xdr_nametype();
extern bool_t xdr_namelist();
extern bool_t xdr_namenode();
extern bool_t xdr_readdir_res();
#endif /* !_DIR_H_RPCGEN */
```

Sun RPC – Ejemplo: Directorio Remoto

Definición del Cliente

```
#include <stdio.h>
#include "dir.h"  /* generado por rpcgen */
#include <rpc/rpc.h>
extern int errno;
main(argc, argv)
   int argc;
   char *argv[];
{
   CLIENT  *clnt;
   char   *server;
   char   *dir;
   readdir_res   *result;
   namelist  nl;
      if (argc != 3) {
                fprintf(stderr, " %s host dir \n", argv[0]);
                exit(1);
   }
   server = argv[1];
   dir = argv[2];
```

Inclusión de la librería para RPC

Protocolo de comunicación

```
 clnt = clnt_create(server, DIRPROG, DIRVERS, "visible");
if (clnt == (CLIENT *) NULL) {
                clnt_pcreateerror(server);
                exit(1);
   }
   result = readdir_1(&dir, clnt);
   if (result == (readdir_res *) NULL) {
                clnt_perror(clnt, server);
                exit(1);
   }
```

Invocación al Procedimiento remoto

Sun RPC – Ejemplo: Directorio Remoto

Definición del Servicio

```
/* dir_proc.c: remote readdir implementation */
#include <dirent.h>
#include "dir.h" /* creado por rpcgen*/
#include <errno.h>
extern int errno;
extern char *malloc();
extern char *strdup();
readdir_res *readdir_1(dirname, req)
   nametype       *dirname;
   struct svc_req  *req;
{
   DIR  *dirp;
   struct dirent  *d;
   namelist  nl;
   namelist  *nlp;
   static readdir_res  res; /* debe ser estático */
   dirp = opendir(*dirname);
   ...
```

En linux se denominará

***readdir_1_svc**

SUN RPC – EJEMPLO: DIRECTORIO REMOTO

STUB DEL CLIENTE. Dir_clnt.c

```
#include "dir.h"
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
static struct timeval TIMEOUT = { 25, 0 };
readdir_res *
readdir_1(argp, clnt)
        nametype *argp;
        CLIENT *clnt;
{
        static readdir_res clnt_res;
        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, READDIR, (xdrproc_t) xdr_nametype,
                (caddr_t) argp,(xdrproc_t) xdr_readdir_res,
                (caddr_t) &clnt_res,    TIMEOUT) != RPC_SUCCESS) {
                return (NULL);          }
        return (&clnt_res);
}
```

Invocación al servicio remoto

Sun RPC – Ejemplo: Sumatoria de Enteros

```
// Definición del Protocolo. vadd.x
typedef int iarray<>;

program VADD_PROG {
  version VADD_VERSION {
    int VADD(iarray) = 1;
  } = 1;
} = 555575555;
```

Sun RPC – Ejemplo: Sumatoria de Enteros

Generado con Solaris 10

Rpcgen vadd.x

vadd.h

```
#ifndef _VADD_H_RPCGEN
#define  _VADD_H_RPCGEN
#include <rpc/rpc.h>

typedef struct {
        u_int iarray_len;
        int *iarray_val;
} iarray;
#define  VADD_PROG
        555575555
#define  VADD_VERSION    1
#define  VADD    1
extern  int * vadd_1();
extern int vadd_prog_1_freeresult();
/* the xdr functions */
extern bool_t xdr_iarray();
# endif /* !_VADD_H_RPCGEN */
```

Sun RPC – Ejemplo: Sumatoria de Enteros

Generado con Solaris 10 Rpcgen -C vadd.x

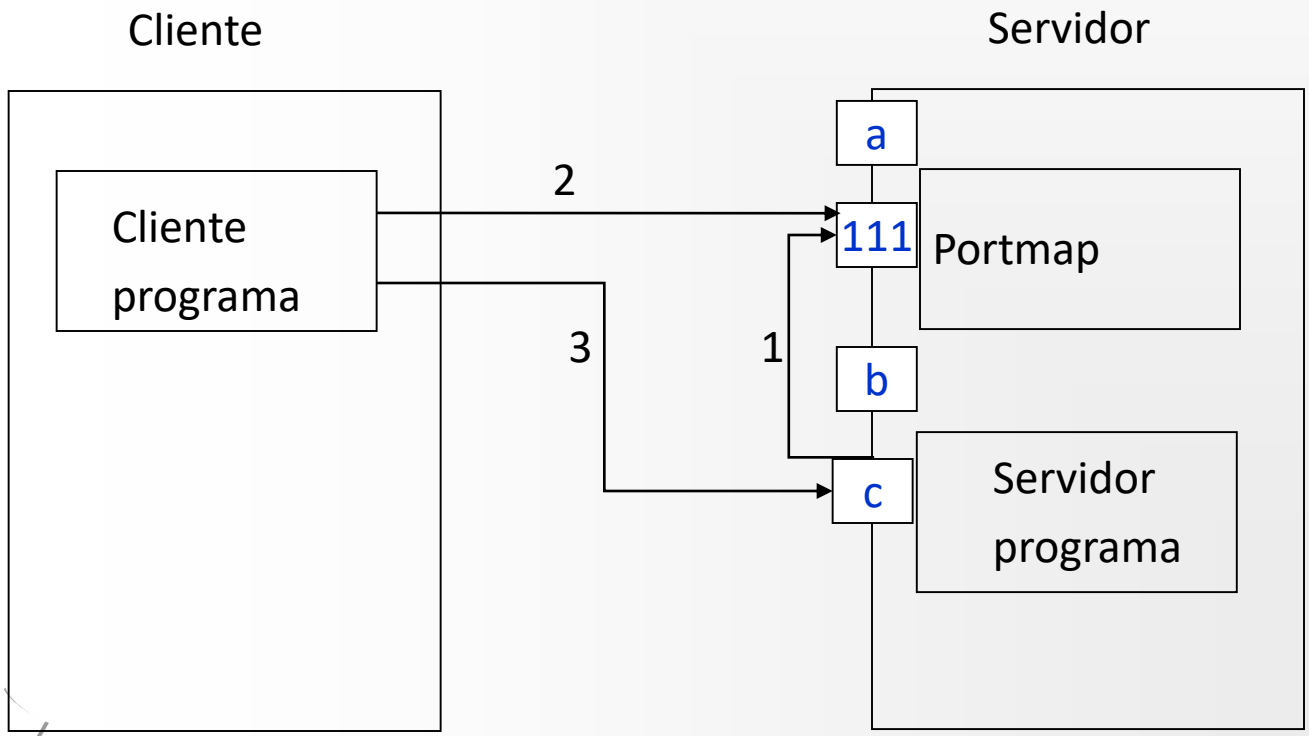
vadd.h

```
#ifndef _VADD_H_RPCGEN
#define _VADD_H_RPCGEN
#include <rpc/rpc.h>
#ifdef __cplusplus
extern "C" {
#endif
typedef struct {
        u_int iarray_len;
        int *iarray_val;
} iarray;
#define VADD_PROG 555575555
#define VADD_VERSION 1
.....
#define VADD 1
extern int * vadd_1(iarray *, CLIENT *);
extern int * vadd_1_svc(iarray *, struct svc_req *);
extern int vadd_prog_1_freeresult(SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define VADD 1
extern int * vadd_1();
extern int * vadd_1_svc();
extern int vadd_prog_1_freeresult();
#endif /* K&R C */
```

Comunicación Remota - Sun RPC

- **Utilidad del RPCBind (Portmap)**
  1. Servidor se registra en el portmap (rpcbind)
  2. Cliente obtiene el puerto del servidor desde el portmap (rpcbind)
  3. Cliente invoca al servidor



Bibliografía:

- Sinha, P. K.; “Distributed Operating Systems: Concepts and Design”, IEEE Press, 1997.
- Tanenbaum, A.S.; van Steen, Maarten; “Distributed Systems: Principles and Paradigms”. 2nd Edition, Prentice Hall, 2007 and 1st Edition 2002.
- van Steen, Maarten; Tanenbaum, A.S; “Distributed Systems”. 3rd Edition, Prentice Hall, 2017.