

SISTEMAS DISTRIBUIDOS

Introducción a RPC

PROGRAMA DE INGENIERIA DE SISTEMAS

ING. DANIEL EDUARDO PAZ PERAFÁN

Contenido

- ❖ Conceptos básicos
- ❖ Aspectos relacionados con las **Remote Procedure Call (RPC)**
- ❖ RPC de Sun Microsystems
- ❖ Ejemplo para desarrollar un programa de RPC
- ❖ Biblioteca de funciones de RPC
- ❖ Concurrencia
- ❖ Arquitectura de los nodos en RPC

Modelo cliente-servidor

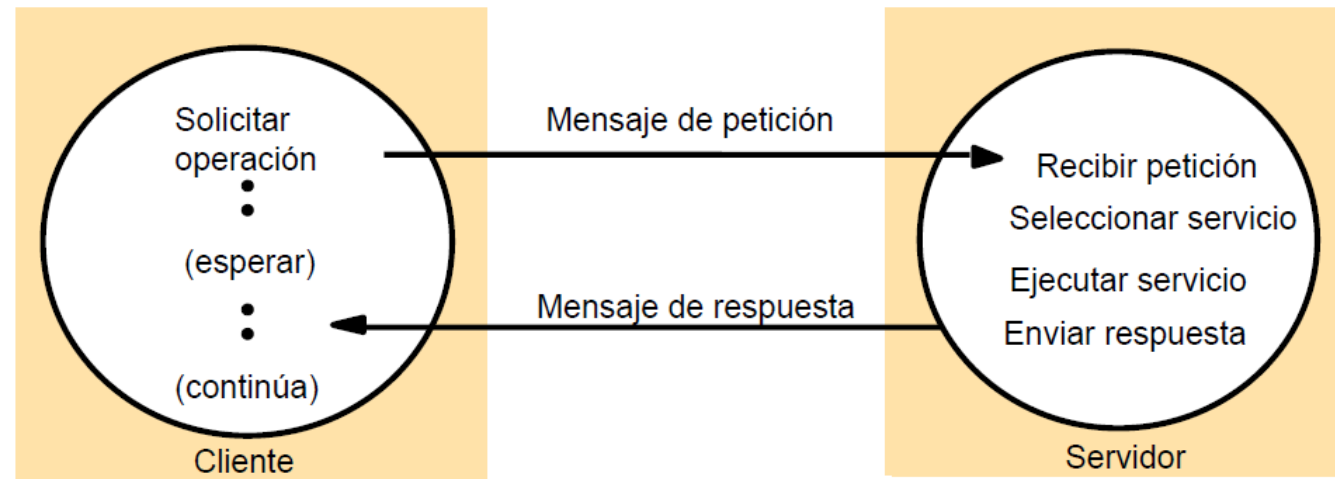


Imagen de F. García Carballera

1. El proceso cliente transmite una petición al proceso servidor.
2. El proceso servidor ejecuta la petición solicitada.
3. El proceso servidor transmite la respuesta al cliente.

Modelo general de RPC

Es posible utilizar una capa de software que abstraee al programador de:

Utilizar operaciones primitivas de enviar – espera – recepción.

Cómo debe ser el formato de los parámetros enviados y la respuesta

Cómo localizar el proceso servidor ¿En que máquina se encuentra? ¿En que puerto esta escuchando?

El cliente no necesita conocer el tipo de máquina o SO del servidor para codificar los parámetros en el mensaje enviado.

Mecanismo de comunicación
de los procesos

Esta capa de software se denomina middleware



RPC
(Remote Procedure Call)
Llamada a Procedimiento Remoto

Modelo general de RPC

Objetivo: hacer que el software distribuido se programe igual que una aplicación no distribuida

Mediante el modelo **RPC** la comunicación se realiza conceptualmente igual que la invocación de un procedimiento local

RPC

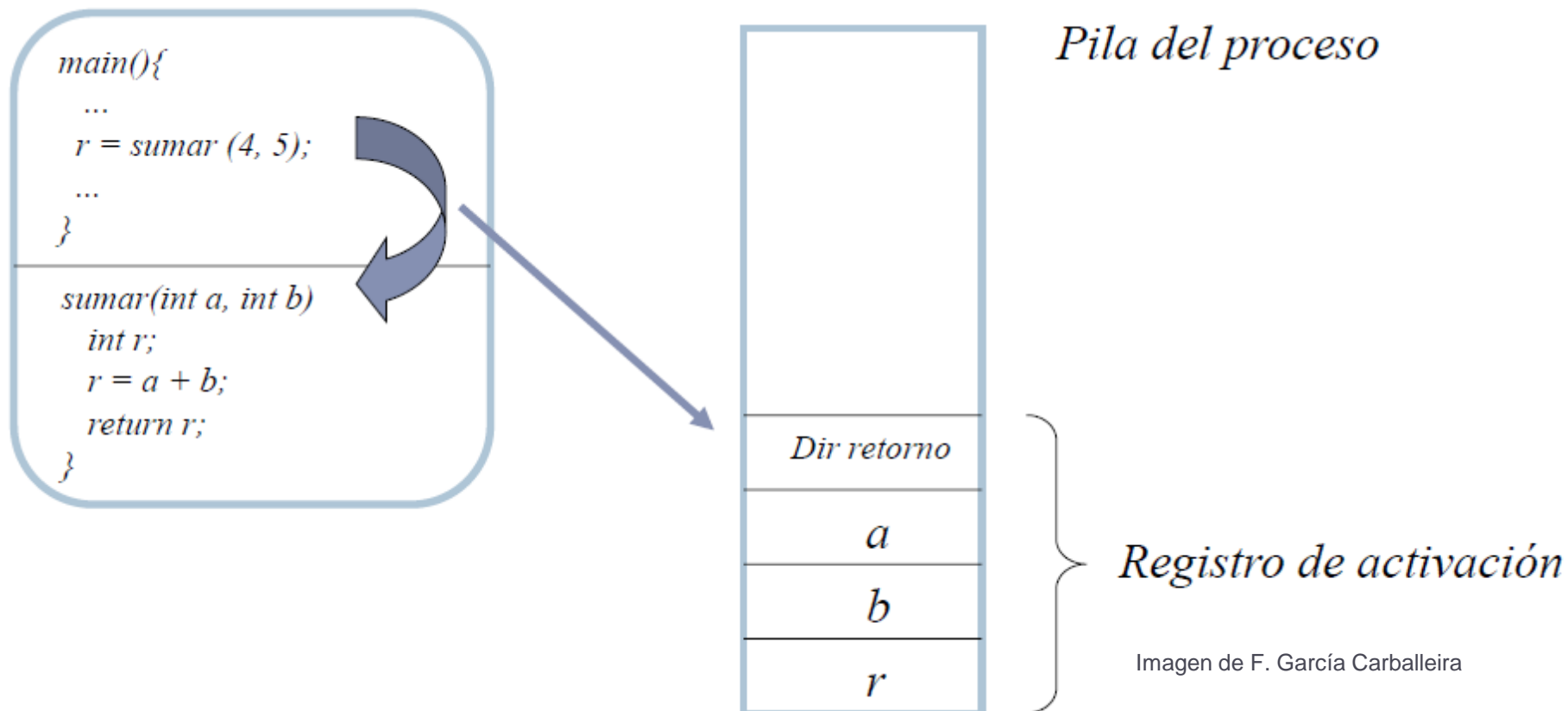
Los detalles de comunicación se esconden detrás de una interface.

La semántica de llamados en RPC es la misma para proc. locales como remotos.

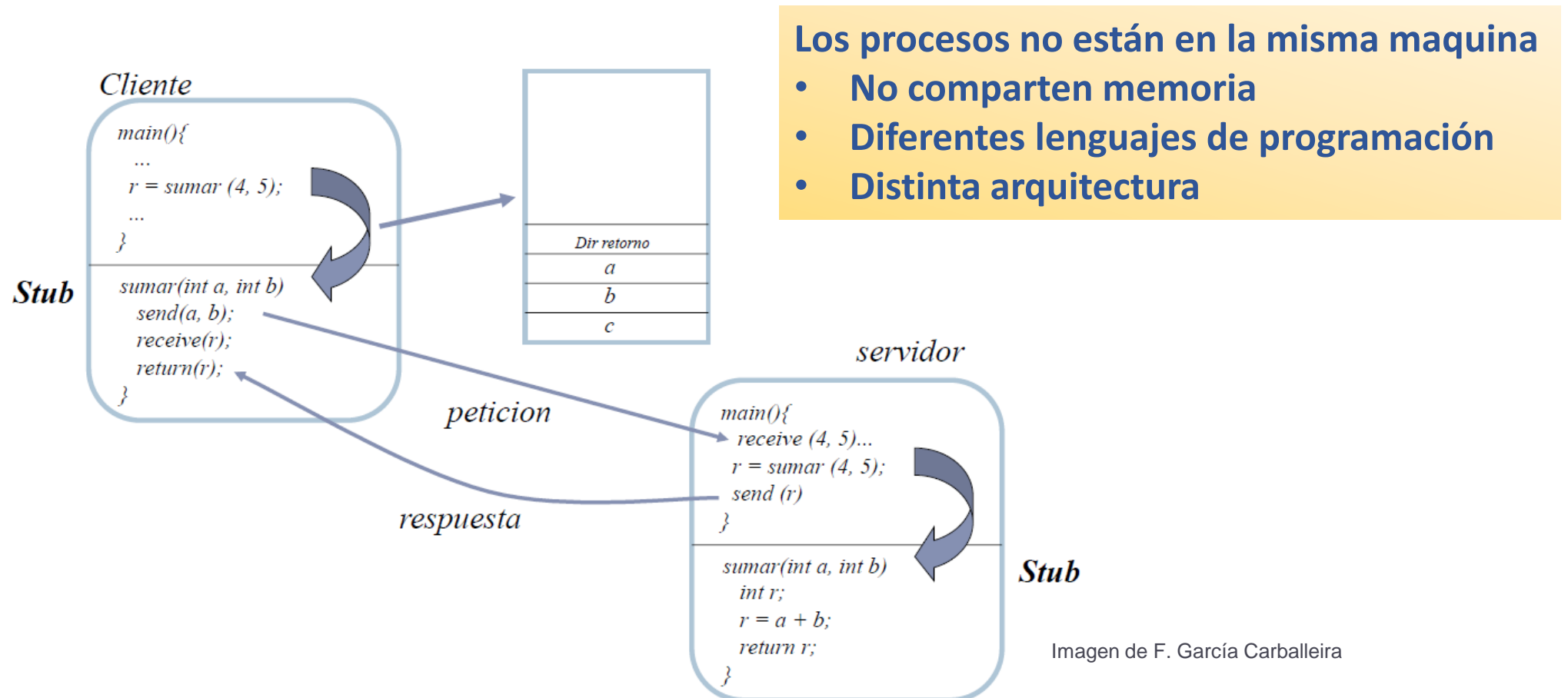
RPC

El concepto de RPC fue presentado por primera vez por Birrell y Nelson en 1984 para el paso de datos entre máquinas bajo entornos heterogéneos.

Llamada local



Llamada remota



Conceptos básicos

Una RPC tiene dos **participantes**:

- ❖ Un **cliente activo**, que envía una RPC al servidor
- ❖ Un **servidor pasivo**, que ejecuta un procedimiento, calcula un resultado y lo devuelve al cliente
- ❖ Un **servicio de red** es una colección de uno o más programas remotos
- ❖ Un **programa remoto** implementa uno o más procedimientos remotos
- ❖ Un servidor puede soportar más de una **versión** de un programa remoto. Permite al servidor ser compatible con las **actualizaciones de protocolos**

Un **procedimiento remoto**, sus parámetros y sus resultados se especifican en un fichero de especificación del protocolo escrito en el lenguaje de especificación de RPC y XDR.

STUB

Se **generan automáticamente** por el software de RPC

- ❖ Stub del cliente
- ❖ Stub del servidor

Un **stub** es una pieza de código usada en el cliente y el servidor

Responsable de **convertir los parámetros (marshalling)** de la aplicación cliente/servidor durante una llamada a procedimiento remoto

- ❖ Espacio de direcciones independiente del cliente y servidor
- ❖ Representaciones de datos diferentes (big-endian, little-endian)

Se apoya de los servicios de comunicación del sistema operativo

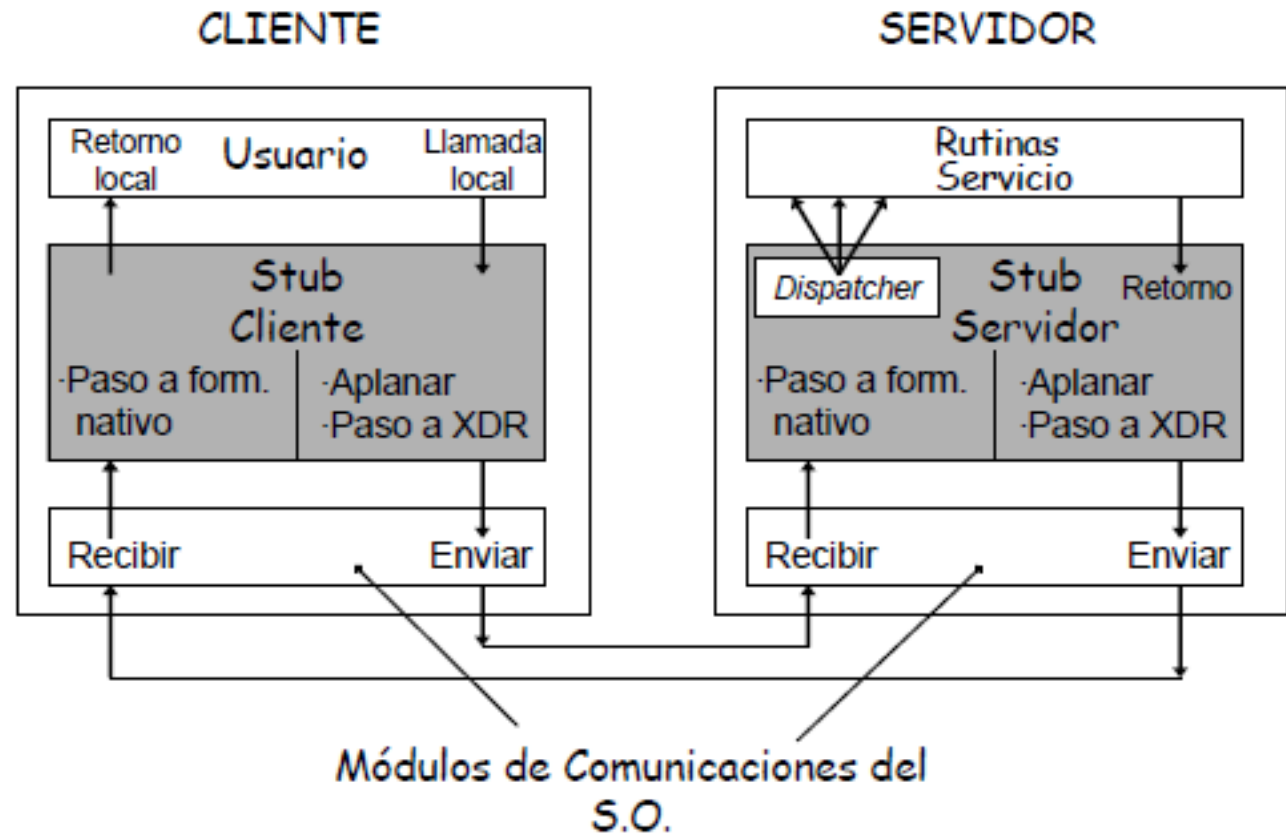
Las RPC de un servicio deben construirse a partir de la definición de su interfaz

Interfaz definida en un lenguaje de definición de interfaces (IDL)

Definen las rutinas remotas

Procesador de interfaces

RPC: Elementos básicos



STUB DEL CLIENTE

RPC: Elementos básicos

Tareas realizadas:

- Suplantar al procedimiento a ejecutar
- Localiza al proceso servidor (IP servidor, Nombre del servicio, Versión, protocolo) .
- Empaquetar (marshalling) id. función y argumentos según un formato estándar.
- Envía el mensaje al servidor .
- Espera la recepción del mensaje .
- Desempaqueta los resultados (unmarshalling) y los devuelve al cliente.

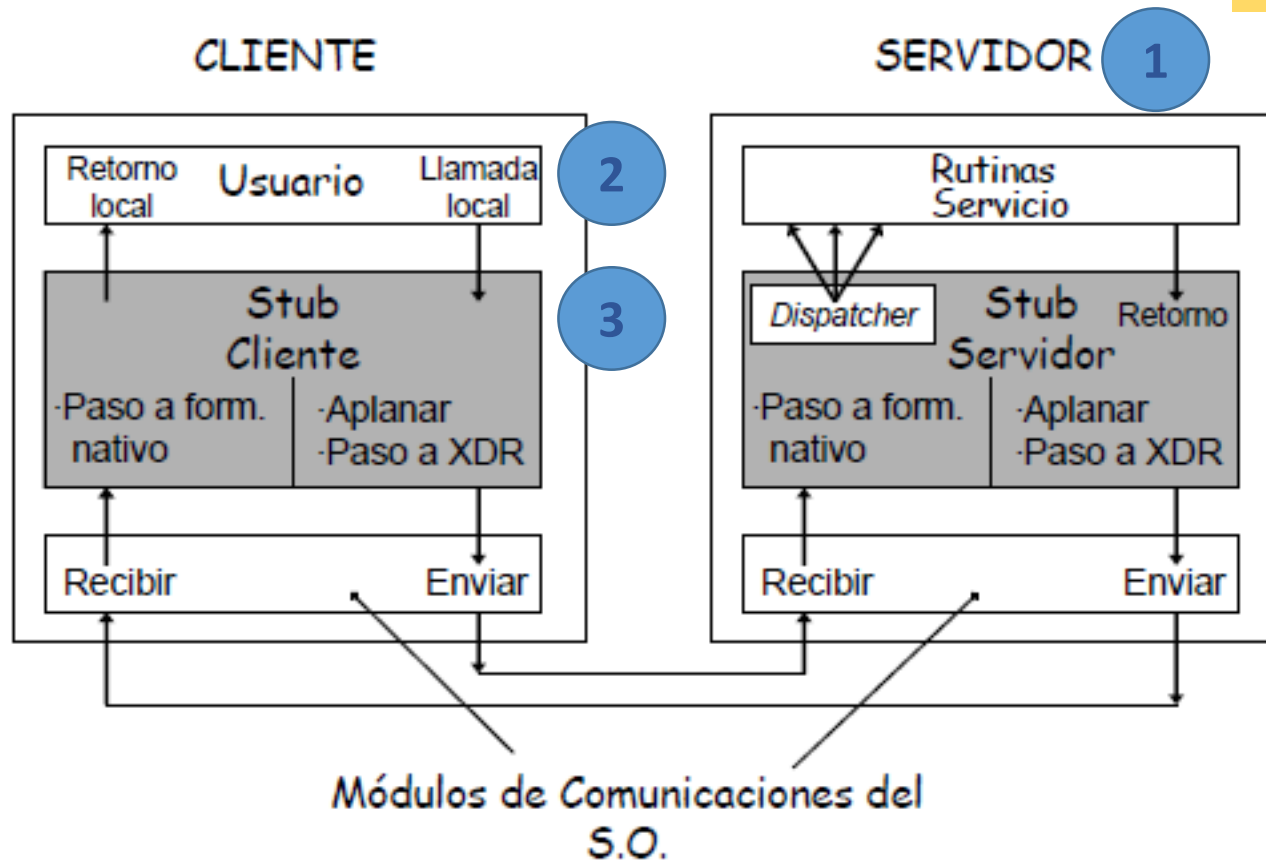
STUB DEL SERVIDOR

RPC: Elementos básicos

Tareas realizadas:

- Registra el servicio (Nombre del servicio, versión)
- Si el servidor es concurrente, gestiona procesos/threads de servicio
- Ejecuta bucle de espera de mensajes
 - Recibe petición
- Desempaqueta el mensaje (unmarshalling)
- Determina qué función invocar. Se encarga el Dispatcher
- Invoca el procedimiento con los argumentos
- Empaqueta el resultado (marshalling)
- Envía el mensaje al stub del cliente

RPC: Elementos básicos



Proceso cliente (llamador)

2. Conectar al servidor

2.1 El cliente invoca una llamada a procedimiento remoto

3. Stub del cliente:

3.1 Localizar al servidor

3.2 Marshalling de los argumentos

3.3 Enviar el mensaje al servidor

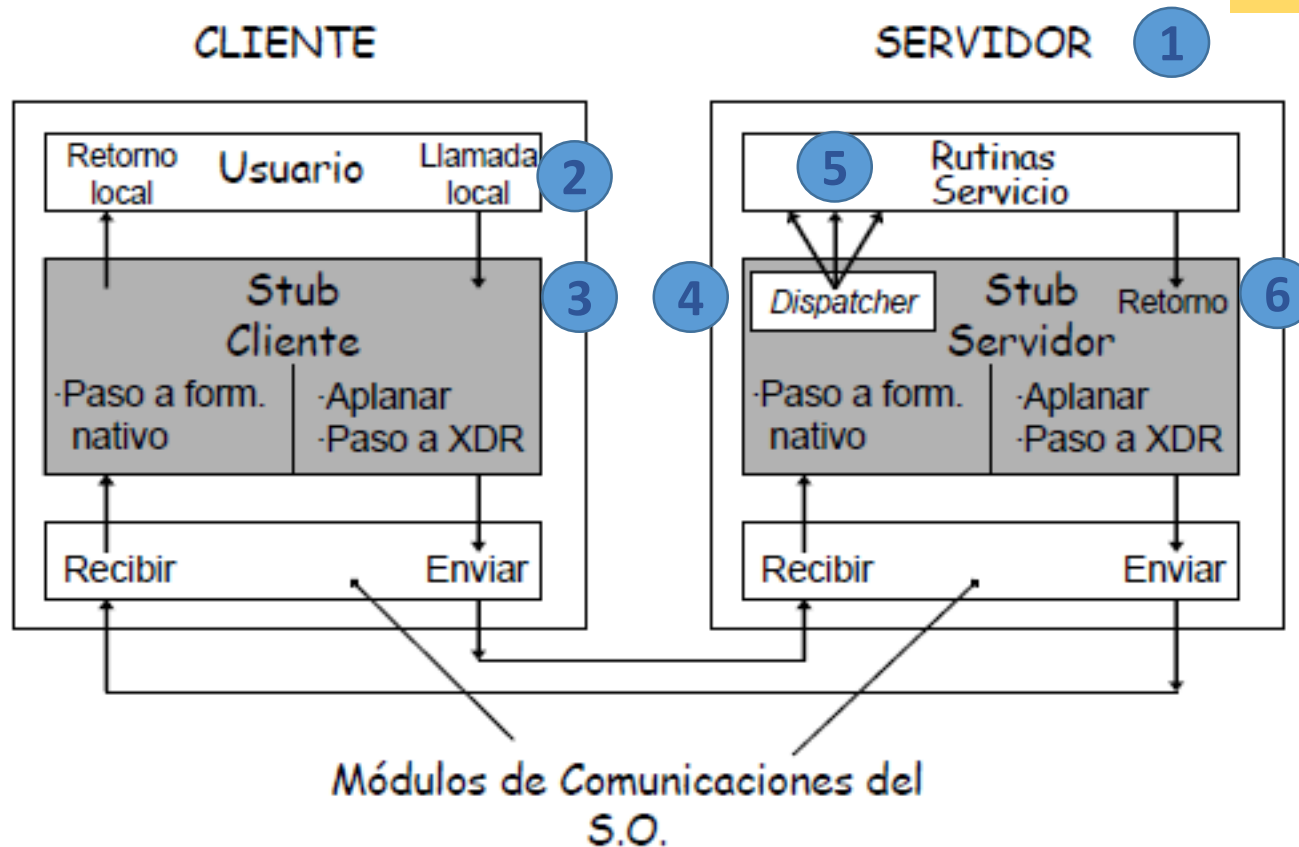
3.4 Requiere una llamada al sistema (Envía el mensaje)

3.5 Bloquearse hasta esperar la respuesta

3.6 El mensaje es transferido al sistema remoto empleado un conjunto de protocolos

7. Obtener la respuesta

RPC: Elementos básicos



Proceso servidor (recibe la llamada)

- 1. Registrar los programas que tienen procedimientos remotos

4. Stub del servidor:

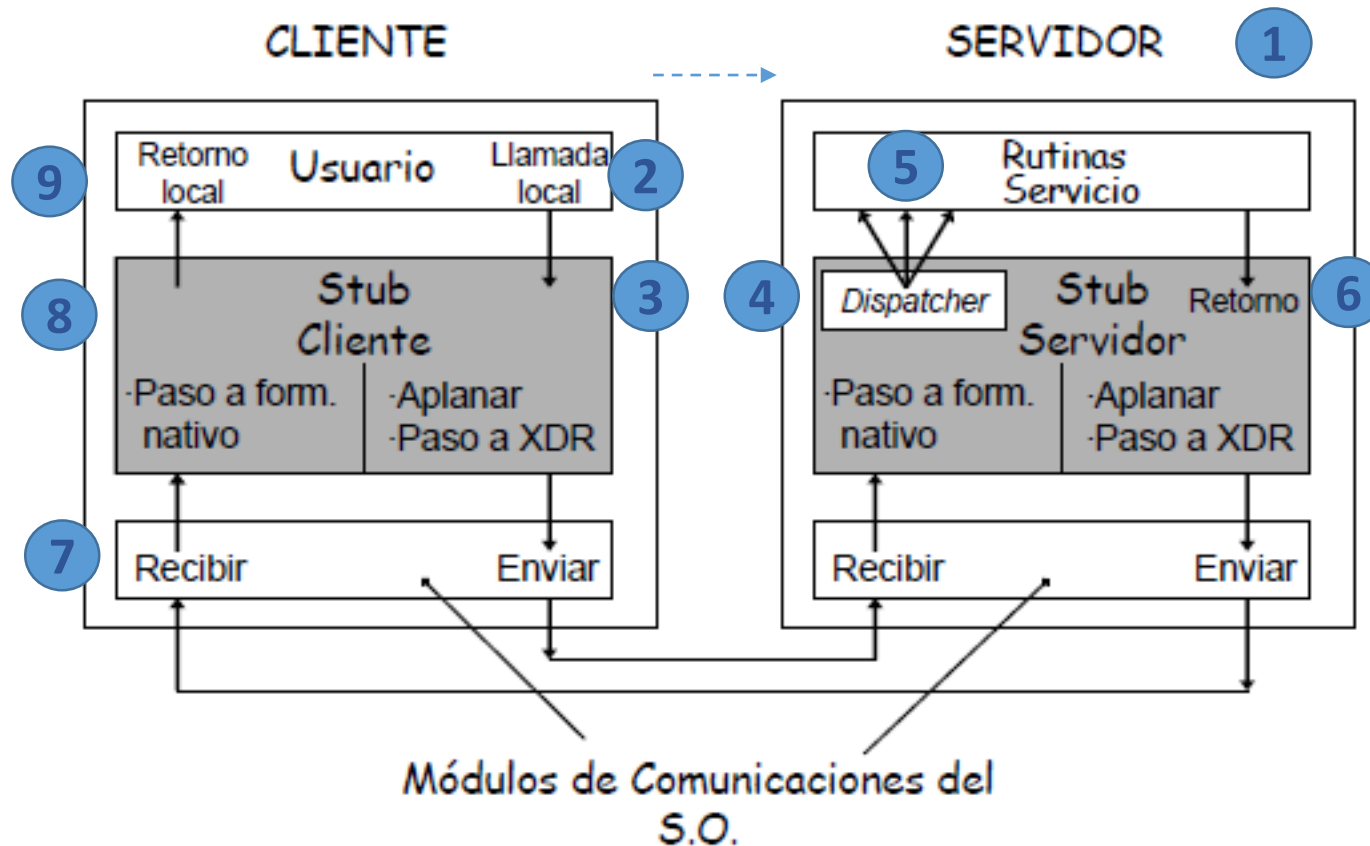
- 4.1 Recibir la petición del cliente
- 4.2 Unmarshalling de los argumentos
- 4.3 Invocar el procedimiento de manera Local y pasa los argumentos

5. Se ejecuta el procedimiento

6. Stub del servidor:

- 6.1 Obtener la respuesta
- 6.2 Marshalling del resultado y enviarla al cliente
- 6.3 Requiere una llamada al sistema (Envía el mensaje)

RPC: Elementos básicos



Proceso cliente (recibe la respuesta)

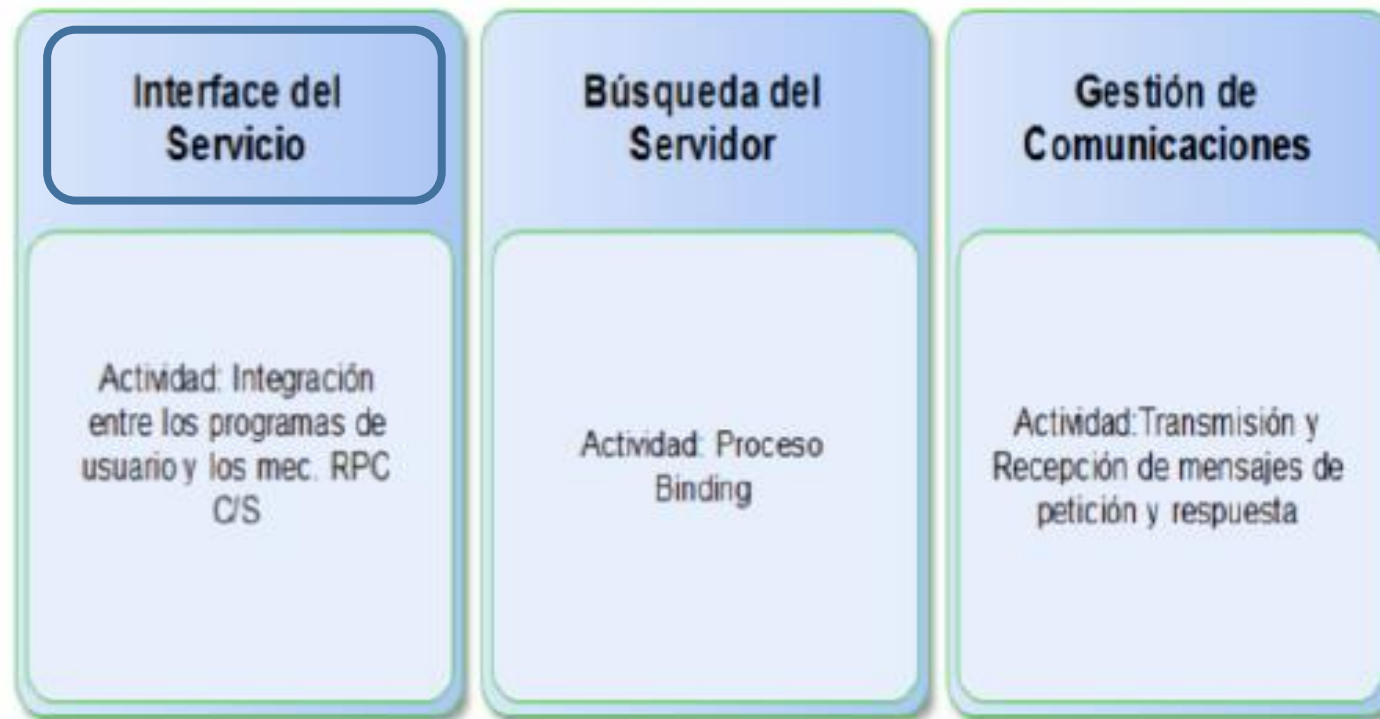
7. **Obtener** la respuesta

8. Stub del cliente:

- 8.1 Recibir la respuesta del servidor
- 8.2 Unmarshalling de la respuesta
- 8.3 Envía la respuesta al cliente que invoco el procedimiento

9. Para el cliente es un retorno a una función normal

Tareas de RPC

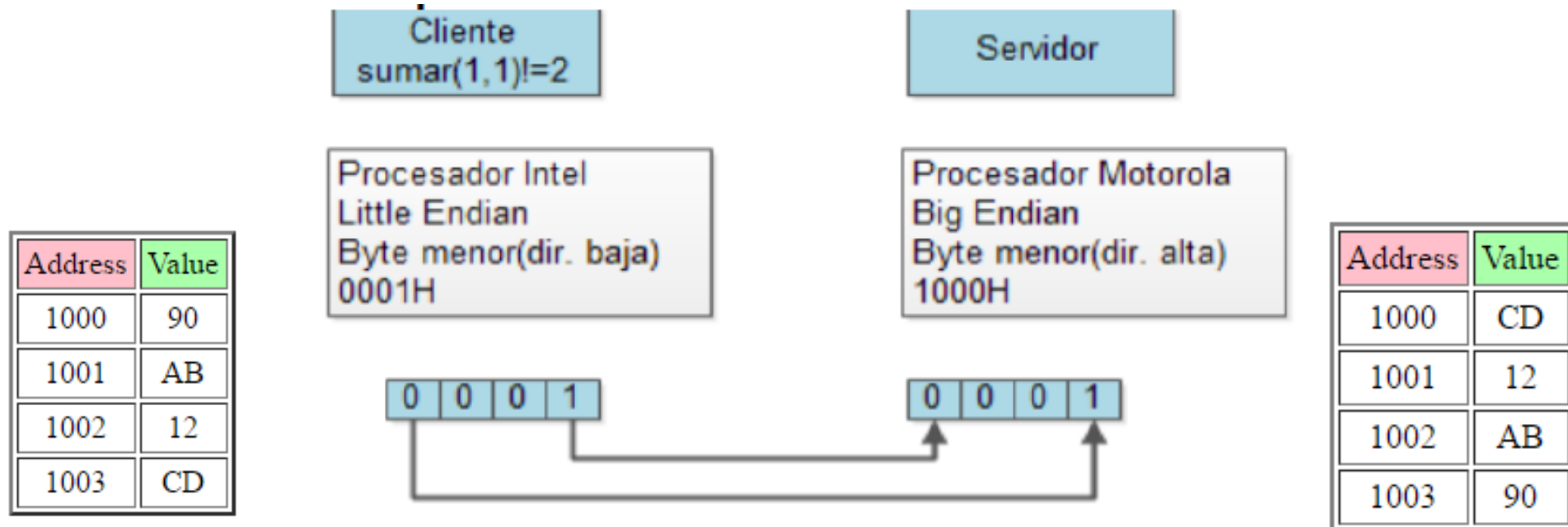


Transferencia de parámetros

La representación de los datos puede ser diferente

Problemas en la representación de los datos:

- ❖ Servidor y cliente pueden ejecutar en máquinas con **arquitecturas distintas** (ordenamiento de bytes)
- ❖ Problemas con los punteros Una dirección sólo tiene sentido en un **espacio de direcciones**



Transferencia de parámetros

La representación de los datos puede ser diferente

Problemas en la representación de los datos:

- ❖ Servidor y cliente pueden ejecutar en máquinas con arquitecturas distintas (ordenamiento de bytes)
- ❖ Problemas con los punteros Una dirección sólo tiene sentido en un espacio de direcciones

Ejemplos de esquemas de representación de datos:

- ❖ **XDR** (*eXternal Data Representation*) es un estándar que define la representación de tipos de datos (**RFC 1832**)
- ❖ **Representación** común de datos de CORBA (**CDR**)
- ❖ **Serialización** de objetos de Java
- ❖ **XML** (*eXtensible Markup Language*) es un metalenguaje basado en etiquetas definida por W3C

Transferencia de parámetros

Definición de la codificación externa de datos

Alternativa 1

* Antes de transmitir usar un formato general conocido (Rep. Ext de Datos).

Ej: XDR Sun, Courier de Xerox
ASN1.

Alternativa 2

* Máquina con arquitectura común:

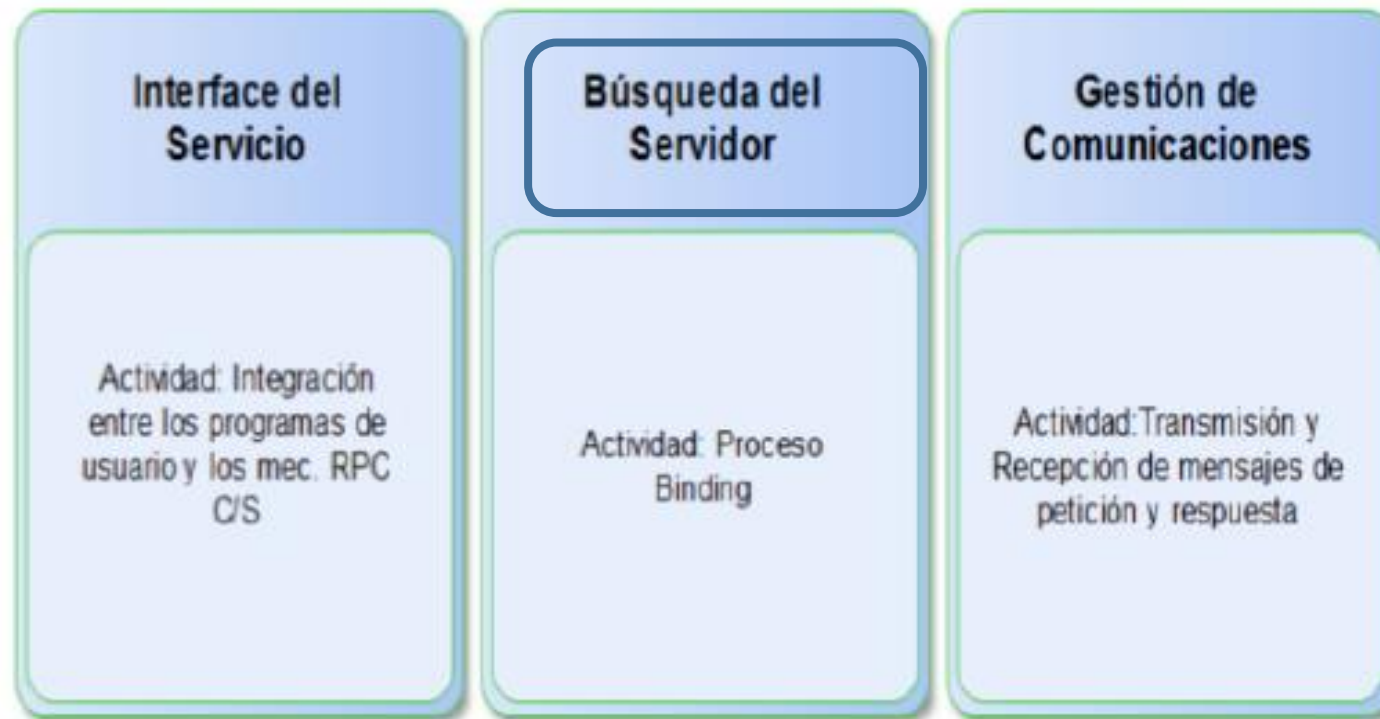
Negociación en el uso de Repres. Exter. de Datos.

Alternativa 3

* Transmitir los datos en un formato nativo junto con un identificador de la arquitectura.

* El Receptor consulta el identificador y define la conversión.

Tareas de RPC



Servicio de enlace (Binding)

- ❖ Servicio que permite establecer la asociación entre el cliente y el servidor
- ❖ Mantiene una tabla de traducciones entre nombres de servicio y puertos.
- ❖ Implica localizar al proceso servidor que ofrece un determinado servicio

Incluye funciones para:

- Registrar un nombre de servicio
- Eliminar un nombre de servicio
- Buscar el puerto correspondiente a un nombre de servicio

- ❖ Diseñado para el sistema de ficheros **NFS**
- ❖ Descrito en **RFC 1831**
- ❖ También se denomina **ONC-RPC** (Open network computing)
- ❖ Se puede elegir **UDP** o **TCP**

Cliente y servidor deben estar de acuerdo en el protocolo de transporte a utilizar

- ❖ Utiliza un lenguaje de definición de interfaces denominado XDR
- ❖ Soporte:

Para C a través del compilador **rpcgen** en UNIX/Linux

Para java

Para Microsoft

Para definir la interface se utiliza un **lenguaje de Definición de Interfaz** (IDL) permite especificar el **formato** de los **procedimientos remotos** y otras **opciones de comunicación**

La interfaz es compartida por

- ❖ Cliente
- ❖ Servidor

Una **interfaz** especifica:

- ❖ Nombre de servicio que utilizan los clientes y servidores
- ❖ Nombres de procedimientos
- ❖ Parámetros de los procedimientos
 - Nombre parámetro.
 - Entrada o salida
- ❖ Tipos de datos de los argumentos

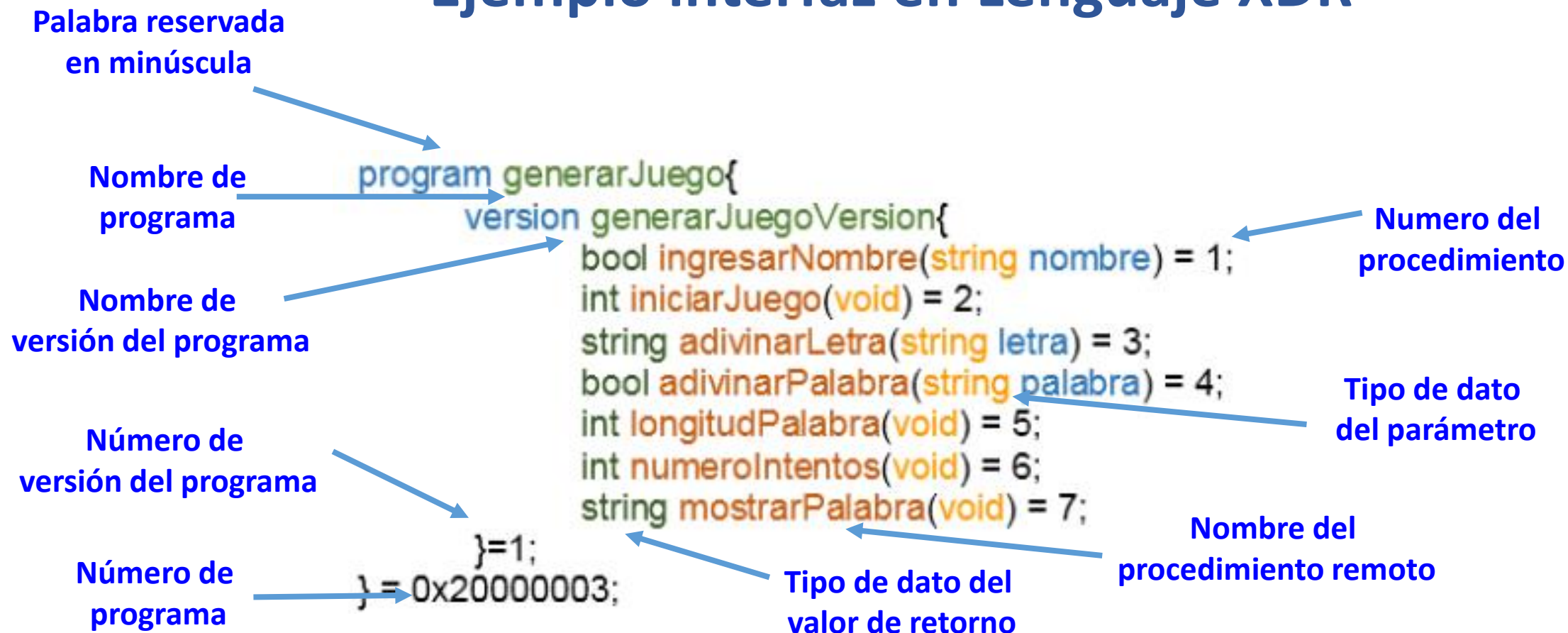
```
program generarJuego{  
    version generarJuegoVersion{  
        bool ingresarNombre(string nombre) = 1;  
        int iniciarJuego(void) = 2;  
        string adivinarLetra(string letra) = 3;  
        bool adivinarPalabra(string palabra) = 4;  
        int longitudPalabra(void) = 5;  
        int numeroIntentos(void) = 6;  
        string mostrarPalabra(void) = 7;  
    }=1;  
} = 0x20000003;
```

El lenguaje para la definición de interfaces es XDR

XDR (*eXternal Data Representation*) es un estándar para la transferencia de datos (**RFC 1832**)

- ❖ Utilizado inicialmente para representaciones externas de datos
- ❖ Se extendió a lenguajes de definición de interfaces
- ❖ Una **interfaz** contiene:
 - Un número de programa
 - Un número de versión del programa
 - Un conjunto de procedimientos remotos:
 - Un nombre y un número de procedimiento
 - Los procedimientos sólo aceptan un parámetro de entrada (se encapsulan en una estructura)
 - Los parámetros de salida se devuelven mediante un único resultado
- ❖ El lenguaje ofrece una notación para definir:
 - constantes
 - definición de tipos
 - estructuras, uniones
 - programas

Ejemplo interfaz en Lenguaje XDR



XDR fue diseñado para trabajar a través de diferentes lenguajes, sistemas operativos y arquitecturas de máquina.

Localización de los servicios en sun RPC

Restricciones a los valores numéricos:

- Versión y Procedimiento: Ninguna.
- Programa:

Nº Programa Rango	Descripción
0x00000000 – 0x1FFFFFFF	Definido por Sun Microsystems
0x20000000 – 0x3FFFFFFF	Definido por el usuario
0x40000000 – 0x5FFFFFFF	Transitorio. Generados dinámicamente por las aplicaciones.
0x60000000 – 0xFFFFFFFF	Reservado

ESTANDAR XDR

Ejemplo de una interfaz en la cual se definen los procedimientos remotos:

En la interface se ha definido una estructura para almacenar la información de un usuario

```
/*Descripcion de los datos a transferir entre sistemas utilizando la notacion XDR*/

/*Declaracion de constantes*/
const MAXNOM=30;
const MAXAPEL=30;
/*definicion del tipo de dato *proxNodo*/
typedef struct nodo_usuario *proxNodo;

/*Declaracion de la estructura que contendra los datos del usuario*/
struct nodo_usuario{
    char nombre[MAXNOM];
    char apellido[MAXAPEL]; /*atributo del usuario, como un vector estatico de tamaño maximo*/
    int edad;
    char identificacion[12];
    proxNodo nodoSiguiente; /* referencia al siguiente elemento de la lista*/
};

/*Definicion de la interface, que permite realizar 3 operaciones*/
program gestion_usuarios{
    version gestion_usuario_version{
        void registrarUsuario(proxNodo usuario)=1;
        proxNodo listarUsuarios(void)=2;
        proxNodo obtenerUsuario(string identificacion)=3;
    }=1;
}=0x20000001;
```

ESTÁNDAR XDR-A

Constantes

- Const

Enteros

- int (32 bits)
- unsigned int
- hyper int (64 bits)

Reales

- float(32 bits IEEE)
- double(64 bits IEEE)

Cadenas

- string <m>
char nombre_tipo X[n]

Arreglos

- Fijos nombre_tipo X[n]
- Variables nombre_tipo X<n>

Estructuras

- struct{
 declaración comp_A
 declaración comp_B
} identificador;



Marshalling de los datos

Se dispone de rutinas estándar que aplanan automáticamente datos

- Enteros
- Reales
- Caracteres

- Cadenas
- Arreglos
- Estructuras
- Registros con discriminantes
- Tipos enumerados

- Tipos opacos (tipos abstractos)

Tener en cuenta que:

El sistema RPC genera un archivo donde se implementan los procedimientos de serialización para datos definidos por el desarrollador

Recordar...

Marshalling

Aplanado de los items de un tipo de dato en una secuencia de items básicos de información y la traducción de estos items en una representación externa de información.

Unmarshalling

Traducción de la representación externa de la información a la representación local (nativa) de los items de un tipo de dato.

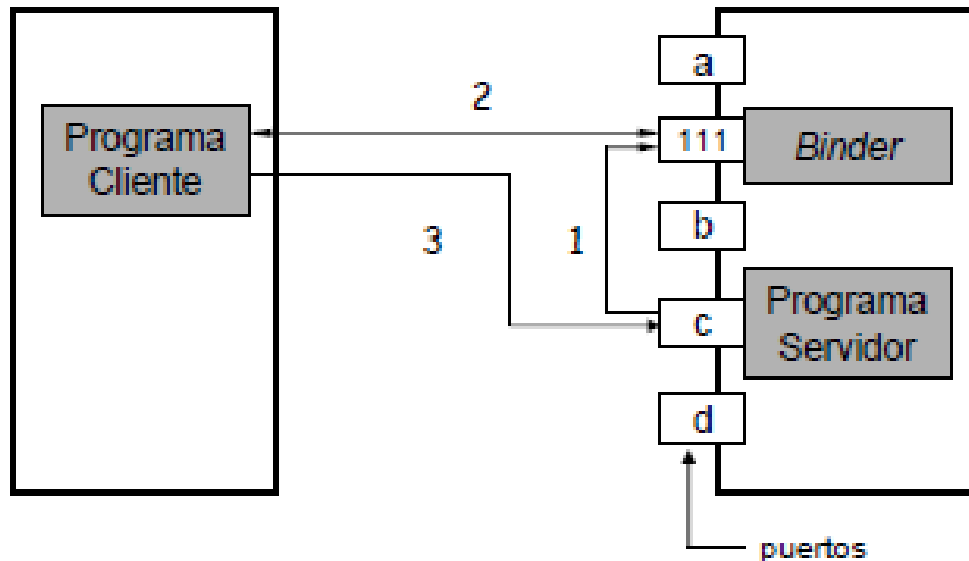
RPC-Binding

- ❖ El **enlace** en las RPC de sun se realiza mediante un proceso denominado **Portmapper**
- ❖ En cada **servidor** ejecuta un proceso **portmapper** en un **puerto bien conocido (111)**. No ofrece un servicio de binding a nivel de red si no un servicio ligado a la maquina.
- ❖ El **portmapper** almacena por **cada servicio local**:
 - El número de programa
 - El número de versión
 - El número de puerto
- ❖ **Enlace dinámico**:
 - El número de puertos disponibles es limitado y el número de programas remotos potenciales puede ser muy grande.
 - Sólo el portmapper ejecutará en un puerto determinado (111) y los números de puertos donde escuchan los servidores se averiguan preguntando al portmapper

RPC-Binding

La RPC de Sun no ofrece un servicio de binding global a nivel de red, sino local para cada máquina.

Servicio → Puerto



1. Cuando un **servidor** arranca **registra** en el *portmapper* la siguiente información:

- El número de programa
- El número de versión
- El número de puerto

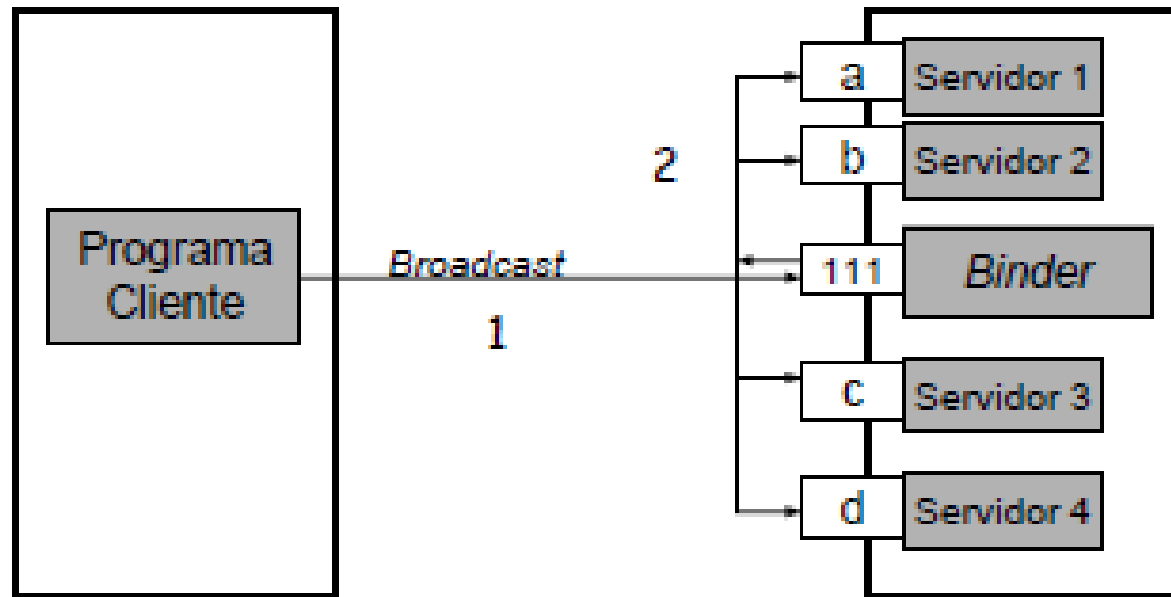
2. Cuando un **cliente** necesita invocar un procedimiento remoto envía al **portmapper** del host remoto (necesita conocer la dirección IP del servidor):

El número de programa y el número de versión

3. El portmapper devuelve el puerto donde escucha el programa.

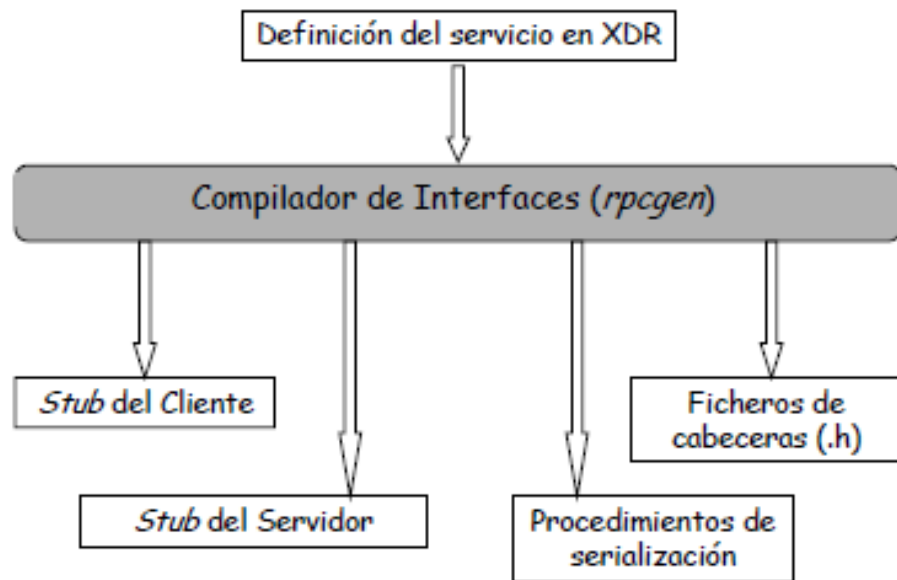
4. El cliente realiza la petición al puerto retornado

RPC-Binding



Una forma de localizar un servicio que no se conoce en que maquina se encuentra es haciendo un broadcast a todos los binders para que hagan una llamada al procedimiento 0 de todos los servicios.

Generación de los componentes para utilizar RPC de sun

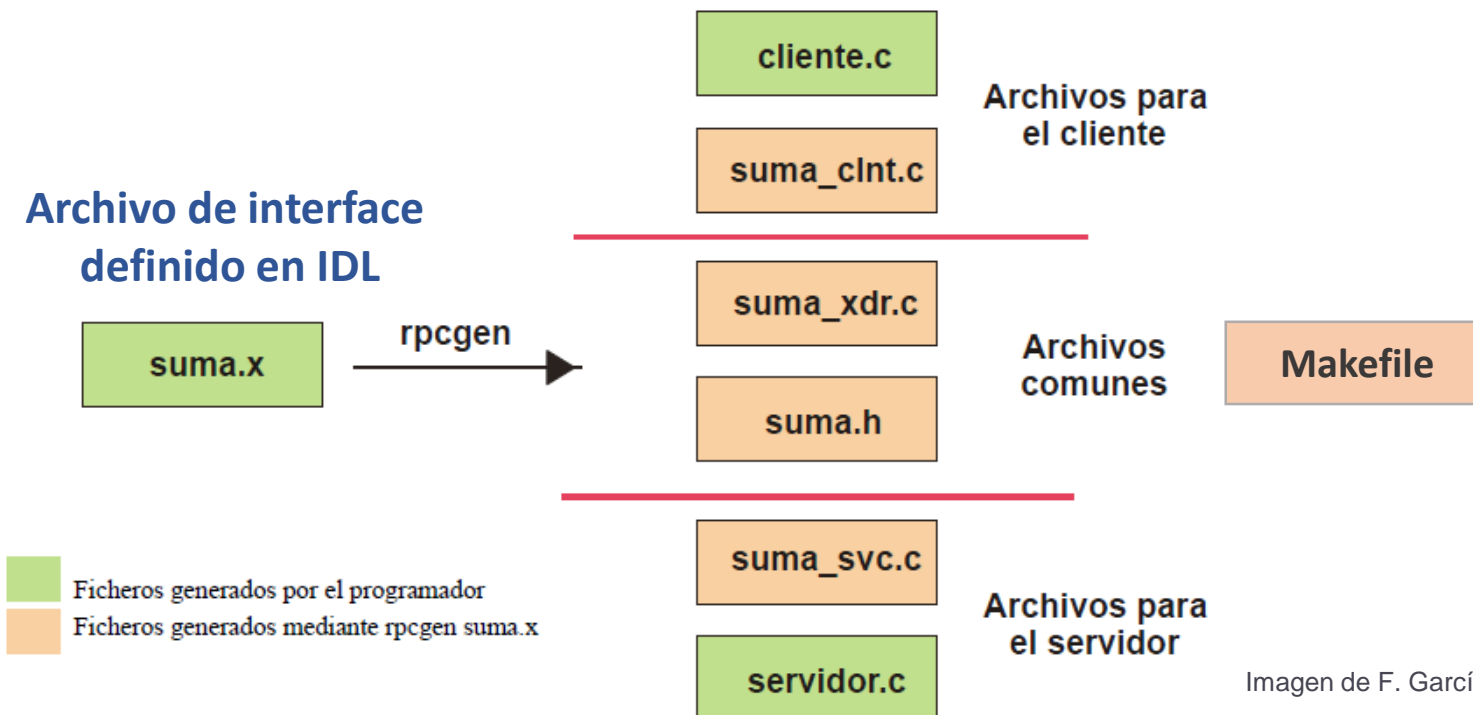


rpcgen es el compilador de interfaces que genera código **C** para:

- ❖ Stub del cliente.
- ❖ Stub del servidor y procedimiento principal del servidor.
- ❖ Procedimientos para el empaquetado y desempaquetado XDR
- ❖ Fichero de cabecera (.h) con los tipos y declaración de prototipos de procedimientos

Compilador de interfaces (rpcgen)

- ❖ RPC de Sun posee un compilador de interfaces denominado *rpcgen*.
- ❖ A partir de una interfaz definida en XDR y con ayuda del *rpcgen*, se obtiene gran parte de los componentes del mecanismo completo de una RPC



FUNCIONES DEL LADO DEL CLIENTE Y SERVIDOR

Para la explicación de cada función se utilizan dos ejemplos que se encuentran en la pagina del curso.

```
program generarJuego{  
  version generarJuegoVersion{  
    bool ingresarNombre(string nombre) = 1;  
    int iniciarJuego(void) = 2;  
    string adivinarLetra(string letra) = 3;  
    bool adivinarPalabra(string palabra) = 4;  
    int longitudPalabra(void) = 5;  
    int numeroIntentos(void) = 6;  
    string mostrarPalabra(void) = 7;  
  }=1;  
} = 0x20000003;
```

juegoAHOR.x



rpccgen

Cliente.c
juegoAHOR.x
makeC

juegoAHOR.h
juegoAHOR_clnt.c

juegoAHOR.h
juegoAHOR_svc.c
Servidor.c

juegoAHOR.x
makeS

Ejemplo para desarrollar un programa de RPC

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS

Antes de iniciar...

Instalar el paquete **build-essential**

El paquete build essential contiene las librerías necesarias para desarrollar aplicaciones usando Sun RPC, tales como cc, gcc, make etc

Comando:

sudo apt-get install build-essential



Ejemplo para desarrollar un programa de RPC

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS

Antes de iniciar...

Verificar que el N_S (**proceso portmapper**) esta activo. En caso de error instalar el paquete libc6-dev

Comando:

rpcinfo -p localhost

```
root@debianDanielServer:/home/daniel# rpcinfo -p localhost
programa vers proto  puerto
100000    2    tcp    111    portmapper
100000    2    udp    111    portmapper
100024    1    udp    39124  status
100024    1    tcp    42571  status _
```

Antes de iniciar...

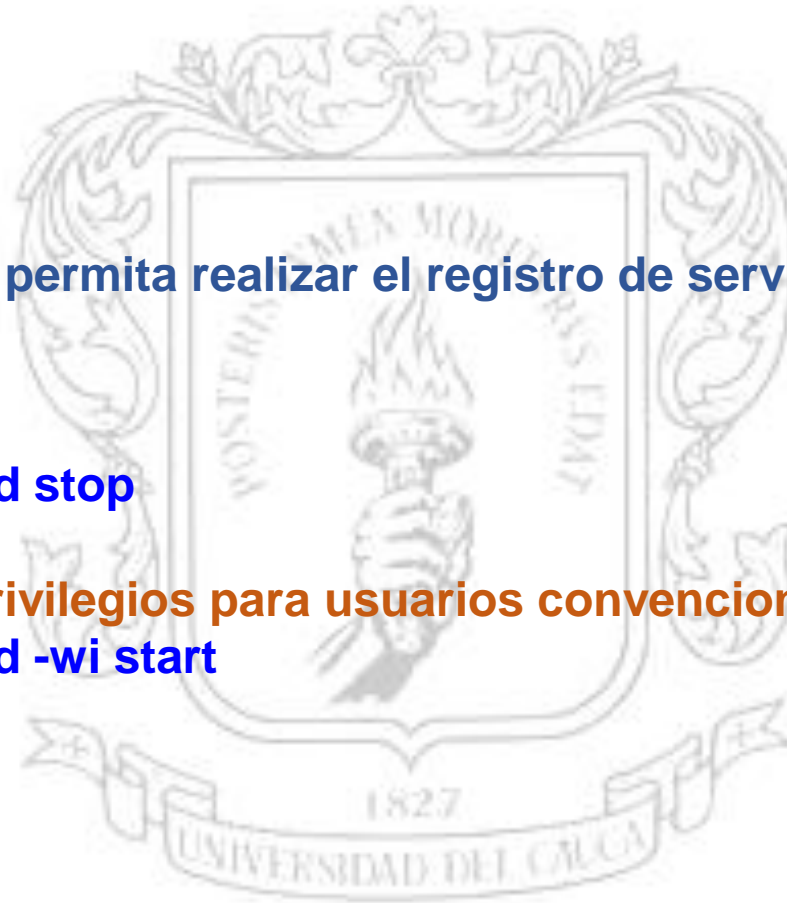
En caso que el N_S no permita realizar el registro de servicios en

1.- Detener el N_S

sudo /etc/init.d/rpcbind stop

2.- Subir el N_S con privilegios para usuarios convencionales


sudo /etc/init.d/rpcbind -wi start



1. Crear la siguiente estructura de directorios



Definición de la interfaz remota



```
program generarJuego{  
    version generarJuegoVersion{  
        bool ingresarNombre(string nombre) = 1;  
        int iniciarJuego(void) = 2;  
        string adivinarLetra(string letra) = 3;  
        bool adivinarPalabra(string palabra) = 4;  
        int longitudPalabra(void) = 5;  
        int numeroIntentos(void) = 6;  
        string mostrarPalabra(void) = 7;  
    } = 1;  
} = 0x20000003;
```


1. Procesamiento de la Interface

Ubicados en el directorio de trabajo, utilice el compilador de interfaces 'rpcgen' para procesar la definición de interfaces creada en el punto anterior. Desde una consola debe introducir el siguiente comando:

```
$ rpcgen juegoAHOR.x
```

Mediante el compilador de interfaces se generan siguientes archivos:

juegoAHOR_clnt.c: Contiene la funcionalidad del client stub.

juegoAHOR_.h: En este archivo de cabecera se definen estructuras declaradas en la interface remota.

juegoAHOR_svc.c: Contiene la funcionalidad del server stub.

Distribuir adecuadamente los archivos generados en las carpetas cliente/ y servidor/

Cliente:

juegoAHOR.x
juegoAHOR_.h
juegoAHOR_clnt.c

Servidor:

juegoAHOR.x
juegoAHOR_.h
juegoAHOR_svc.c



Crear el código del Cliente.

Pasos a seguir:

- a. Ubicar el directorio cliente, mediante el comando:
cd cliente
- b. Generar la plantilla del cliente, mediante el comando:
rpcgen -Sc juegoAHOR.x > Cliente.c

Ubicados en el subdirectorio 'cliente', cambiar los permisos para escribir en Cliente.c, luego modificar el código generado (archivo Cliente.c), introduciendo la lógica de control del programa, y los mensajes guía para orientar al usuario.

Crear el código del Servidor.

De acuerdo al requerimiento inicial es responsabilidad del programador del servicio implementar el código del servidor, para lo cual se utilizarán las plantillas de las Rutinas de Servicio.

Pasos a seguir:

- a. Ubicar el directorio servidor, mediante el comando:
cd servidor
- b. Generar la plantilla del cliente, mediante el comando:
rpcgen -Ss juegoAHOR.x > Servidor.c

Ubicados en el subdirectorio servidor, cambiar los permisos para escribir en Servidor.c, luego modificar el código generado (archivo Servidor.c), introduciendo la lógica de cada una de los procedimientos

Compilar códigos

Pasos a seguir:

Estando en la carpeta **Servidor**, desde la consola generar el archivo **makefile** a través de la opción que posee la aplicación rpcgen para facilitar la compilación de los archivos fuente. Esto se logra mediante el siguiente comando:

```
rpcgen -Sm juegoAHOR.x > makeS
```

Estando en la carpeta **Cliente**, desde la consola generar el archivo **makefile** a través de la opción que posee la aplicación rpcgen para facilitar la compilación de los archivos fuente. Esto se logra mediante el siguiente comando:

```
rpcgen -Sm juegoAHOR.x > makeC
```

Luego de haber generado los anteriores archivos, se edita el **makefile** generado para el servidor y el cliente:

Editar y modificar los siguientes parámetros en el archivo **makeC**

CLIENT: cliente

SERVER: Borrar valor (Debe quedar en blanco)

SOURCES_CLNT.c = Cliente.c

SOURCES_CLNT.h = juegoAHOR.h

Editar y modificar los siguientes parámetros en el archivo **makeS**

CLIENT: Borrar valor (Debe quedar en blanco)

SERVER: Servidor

SOURCES_SVC.c = Servidor.c

SOURCES_SVC.h = juegoAHOR.h



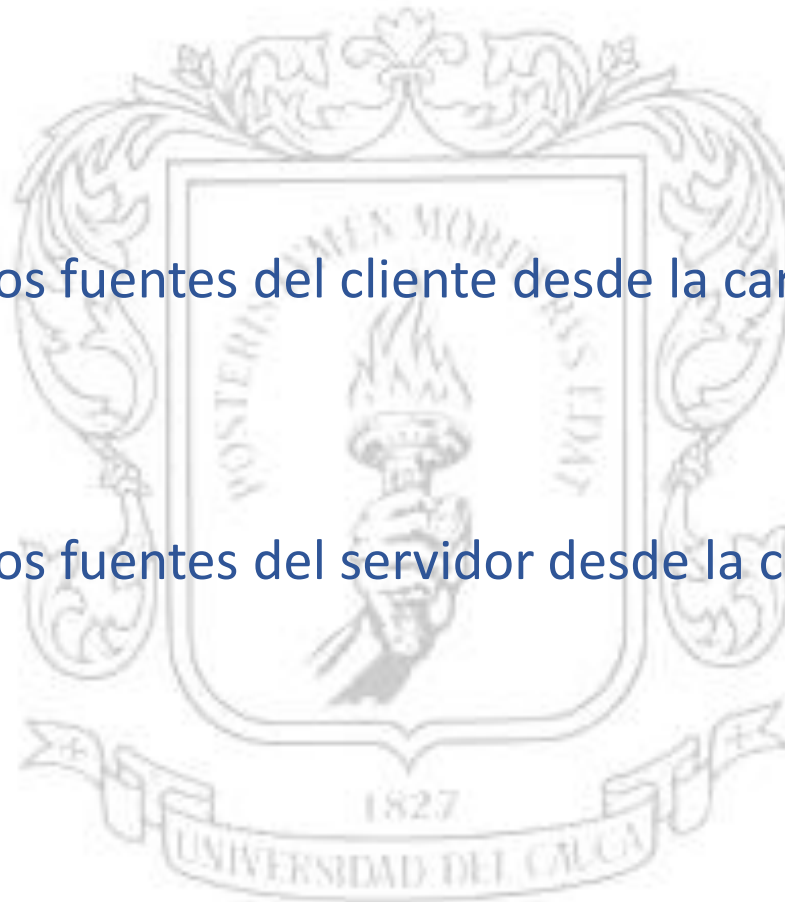
Compilar los fuentes:

Para compilar los códigos fuentes del cliente desde la carpeta **Cliente:**

make -f makeC

Para compilar los códigos fuentes del servidor desde la carpeta **Servidor:**

make -f makeS



Ejecutar el cliente y el servidor

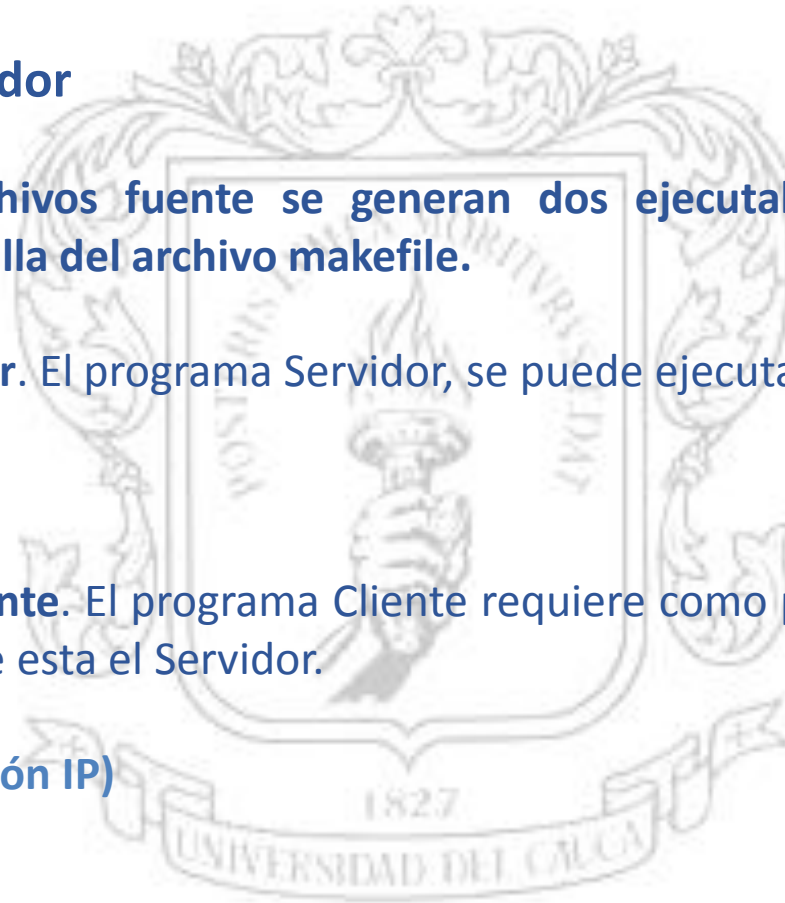
Cuando se compilan los archivos fuente se generan dos ejecutables, cuyo nombre depende de los parámetros fijados en la plantilla del archivo makefile.

Ubicados en la carpeta **Servidor**. El programa Servidor, se puede ejecutar localmente o en otra máquina.

./servidor

Ubicados desde la carpeta **Cliente**. El programa Cliente requiere como parámetro de entrada el nombre de la máquina (o dirección IP) donde esta el Servidor.

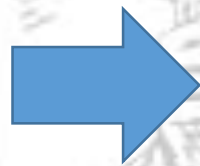
./cliente localhost (o dirección IP)



Para la explicación de cada función se utilizan los ejemplos que se encuentran en la pagina del curso.

```
program generarJuego{  
  version generarJuegoVersion{  
    bool ingresarNombre(string nombre) = 1;  
    int iniciarJuego(void) = 2;  
    string adivinarLetra(string letra) = 3;  
    bool adivinarPalabra(string palabra) = 4;  
    int longitudPalabra(void) = 5;  
    int numeroIntentos(void) = 6;  
    string mostrarPalabra(void) = 7;  
  }=1;  
} = 0x20000003;
```

juegoAHOR.x



rpcegen

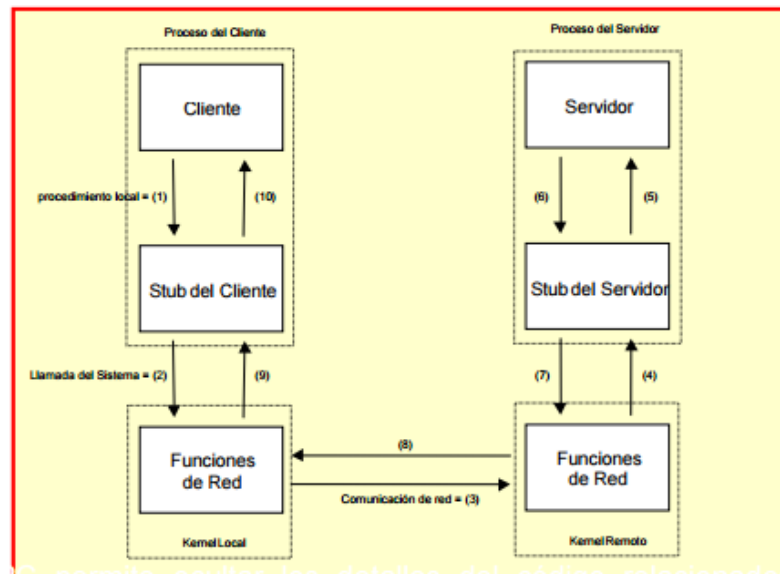
Cliente.c
juegoAHOR.x
makeC

juegoAHOR.h
juegoAHOR_clnt.c

juegoAHOR.h
juegoAHOR_svc.c
Servidor.c

juegoAHOR.x
makeS

La lógica del cliente, el stub del cliente, la lógica del servidor, el stub del servidor y el dispatcher ejecutan un conjunto de funciones predefinidas que permiten realizar las siguientes tareas:



- Registro del servicio en el portmap.
- Ejecución del proceso servidor.
- Localización del servicio en el portmap.
- Marshalling y un marshalling de los argumentos enviados.
- Invocación a los procedimientos remotos.
- Retorno de un valor al procedimiento remoto.

Funciones del programa usuario

Principales funciones:

Sintaxis:

```
#include <rpc/rpc.h>

CLIENT *clnt_create(const char *host,
                    const u_long prognum,
                    const u_long versnum,
                    const char *nettype);

CLIENT *clnt;

clnt = clnt_create (host, gestion_usuarios, gestion_usuario_version, "udp");
if (clnt == NULL) {
    clnt_pcreateerror (host);
    exit (1);
}
```

La función `clnt_create ()` :

- ❖ Se contacta con el portmapper de host.
- ❖ Crea y devuelve un **identificador** para un programa y versión específica en un host remoto donde se encuentra el servidor. Esto se hace especificando un **protocolo de transporte**.
- ❖ El portmapper indica la localización (puerto).



Funciones del programa usuario

Principales funciones:

Sintaxis:

```
#include <rpc/rpc.h>

void clnt_destroy(CLIENT *clnt);

CLIENT *clnt;

clnt = clnt_create (host, gestion_usuarios, gestion_usuario_version, "udp");
if (clnt == NULL) {
    clnt_pcreateerror (host);
    exit (1);
}

clnt_destroy (clnt);
```

La función `clnt_destroy()`:

Destruye el identificador del cliente RPC.
Cancela el descriptor almacenado en `clnt`.



Funciones del stub del cliente

Principales funciones:

Syntax

```
#include <rpc/rpc.h>

enum clnt_stat clnt_call(CLIENT *clnt,
                        const u_long procnum,
                        const xdrproc_t inproc,
                        const caddr_t in,
                        const xdrproc_t outproc,
                        caddr_t out,
                        const struct timeval tout);

bool_t *
ingresarnombre_1(char **argp, CLIENT *clnt)
{
    static bool_t clnt_res;

    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, ingresarNombre,
                  (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
                  (xdrproc_t) xdr_bool, (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}
```

La función `clnt_call()`:

Invoca a un procedimiento remoto.

Parámetros:

- Descriptor del cliente.
- Identificación de la operación
- Rutina de serialización del argumento
- Argumento de entrada de la operación
- Rutina para serializar el resultado
- Dirección de la variable que almacena el resultado
- Tiempo máximo de espera



Funciones del servidor

El programa servidor se compone de



STUB DEL SERVIDOR

+

DISPATCHER



RUTINAS DEL SERVIDOR



Funciones del stub del servidor

Principales funciones:

```
SVCXPRT *svcudp_create(int sock);
```

```
SVCXPRT *svctcp_create(int sock,  
    unsigned int send_buf_size,  
    unsigned int recv_buf_size);
```

Estas rutinas crean un servicio de transporte UDP/IP o TPC/IP según sea el caso, a la que se devuelve un puntero. El transporte se asocia con el socket sock, (numero que identifica al socket). Si el argumento es RPC_ANYSOCK, se crea un nuevo socket, y se le asocia un puerto de escucha



Funciones del stub del servidor

Principales funciones:

```
bool_t svc_register(SVCXPRT *xpirt, unsigned long prognum,  
                   unsigned long versnum,  
                   void (*dispatch)(svc_req *, SVCXPRT *),  
                   unsigned long protocol);
```

Registra el servicio en el portmap mediante:

el número de servicio, la version del servicio, el procedimiento dispatcher, el protocolo utilizado para el intercambio de mensajes y el Puerto en el cual escucha el servicio (contenido en el parametro xpirt)



Funciones del stub del servidor

Principales funciones:

`bool_t svc_sendreply(SVCXPRT *xprt, xdrproc_t outproc, char *out);`

Es invocada por la rutina dispatcher para enviar el resultado de la llamada a un procedimiento remoto.

- *xprt* es el descriptor de transporte asociado a la solicitud.
- *outproc* es la rutina XDR asociada para codificar el resultado.
- *out* es la dirección del resultado



Funciones del stub del servidor

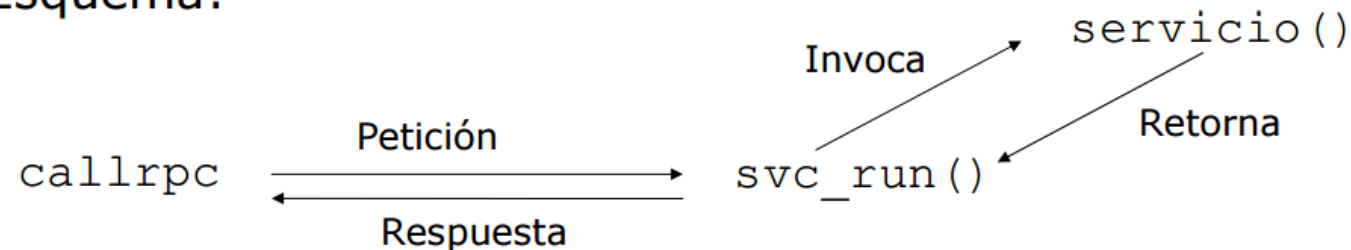
`svc_run()`

Permite pasar al proceso servidor al estado de recepción de peticiones

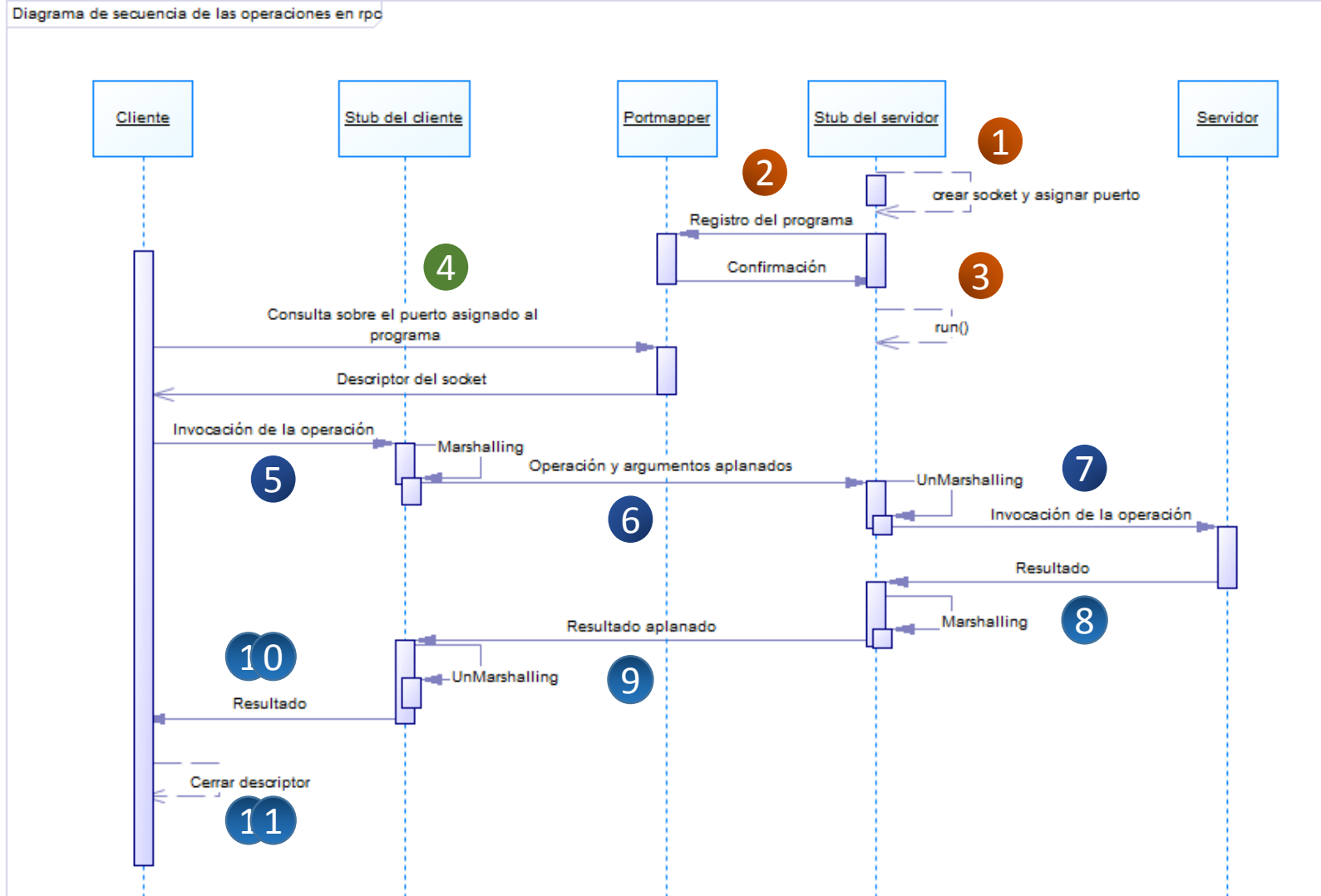
Sin parámetros. No retorna nunca. Es la última función a invocar en el servidor.

Esta función deja residente el servidor y le transfiere el control al runtime de las RPC para que reciba las peticiones de servicio de los clientes y las gestione.

Esquema:



Funciones principales de sun rpc



- 1 `svcudp_create` o `svctcp_create`
- 2 `svc_register`
- 3 `svc_run()`
- 4 `clnt_create()`
- 5 **Invocación del procedimiento**
- 6 `clnt_call()`
- 7 **Invocación del**
- 8 **procedimiento local**
- 9 `svc_sendreply`
- 10 **Respuesta del procedimiento**
- 11 `clnt_destroy()`



Funciones de RPC

Niveles de acceso a RPC

La biblioteca RPC se divide en subconjuntos de funciones, conocidos como Niveles de Acceso a RPC. Según el grado de sofisticación de la aplicación a desarrollar, se puede acceder a subconjuntos de funciones de la biblioteca de RPC.

Existen al menos 3 Niveles:

- Nivel superior: Programas y funciones que internamente usan RPC.
- Nivel intermedio
- Nivel inferior



Las operaciones en rpc normalmente son síncronas y en serie

- El cliente se bloquea después de una llamada a rpc hasta recibir una respuesta.
- Un cliente solo puede emitir una solicitud a la vez.
- En el lado del servidor, solo procesa una solicitud a la vez.



- RPC es sincrónico y en serie, el cliente se bloquea después de realizar una llamada RPC.
- Para implementar el paralelismo se recurre al uso de hilos o procesos hijos.
- Cada cliente puede realizar varias llamadas remotas simultáneas si maneja varios descriptores usando `clnt_create()`.
- Es necesario proteger las variables compartidas y utilizar mecanismos apropiados para sincronizar la interacción de los servicios.

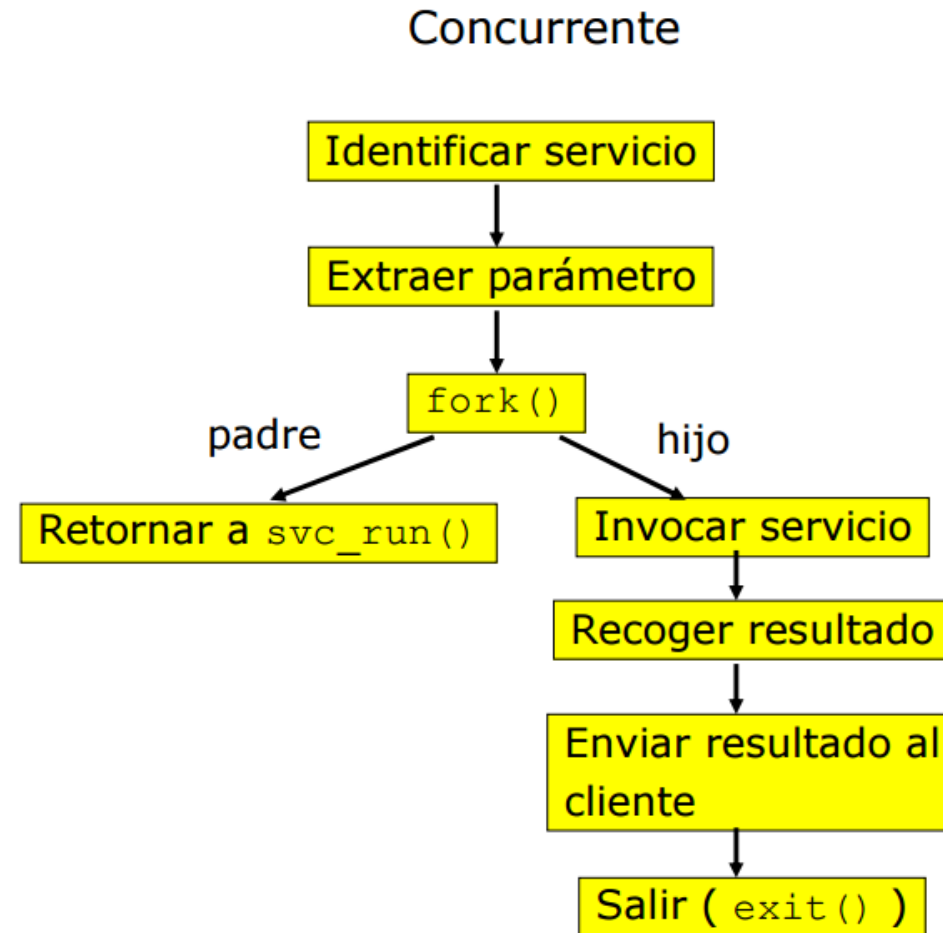
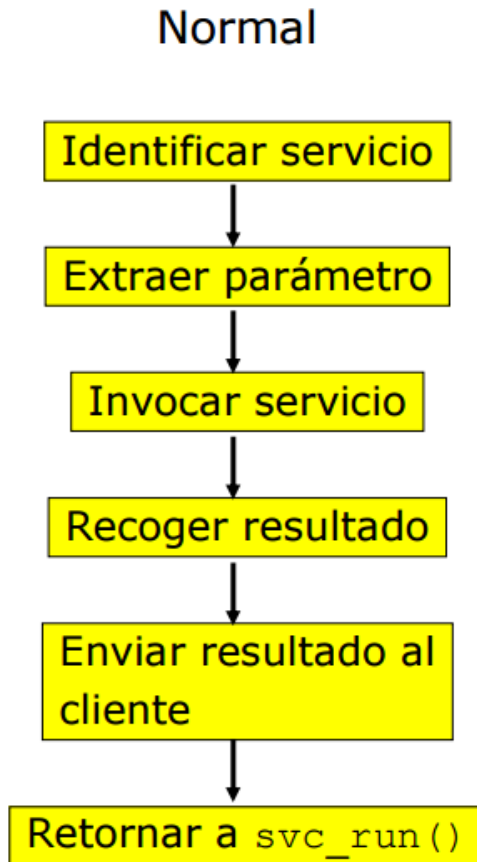


En el proceso servidor tenemos la rutina repartidora (dispatcher) que realiza la siguientes tareas:

1. Identifica y selecciona el procedimiento que invoca el cliente inicializando los filtros XDR adecuados.
2. Devuelve un mensaje de error si el procedimiento no existe.
3. Extrae del paquete que recibe por la red el argumento del procedimiento.
4. Invoca el procedimiento con el parámetro recibido.
5. Recoge el resultado devuelto por el servicio y se lo envía al cliente.
6. Retorna a `svc_run()`.

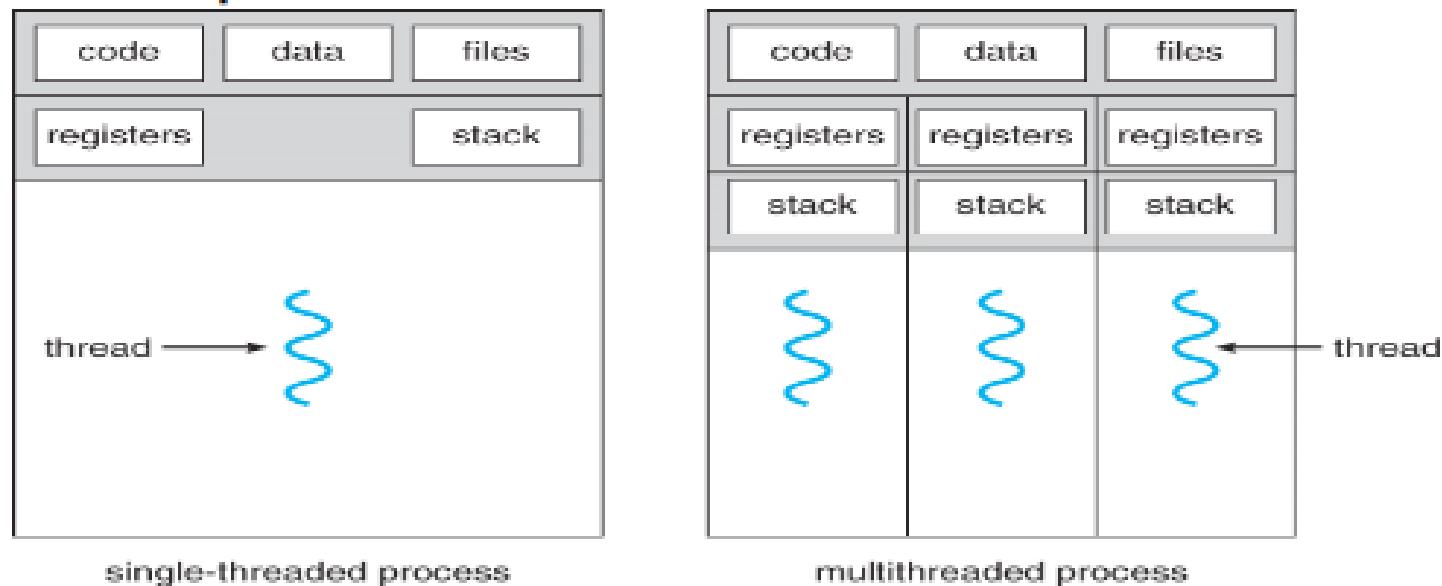


En cualquier momento antes del punto 3 el repartidor puede crear un proceso o hilo nuevo que haga el trabajo y retornar a `svc_run()`.



Cada proceso tiene asignado un espacio de direcciones y un hilo de ejecución

Hilo: unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. Los hilos comparten el mismo espacio de direcciones.



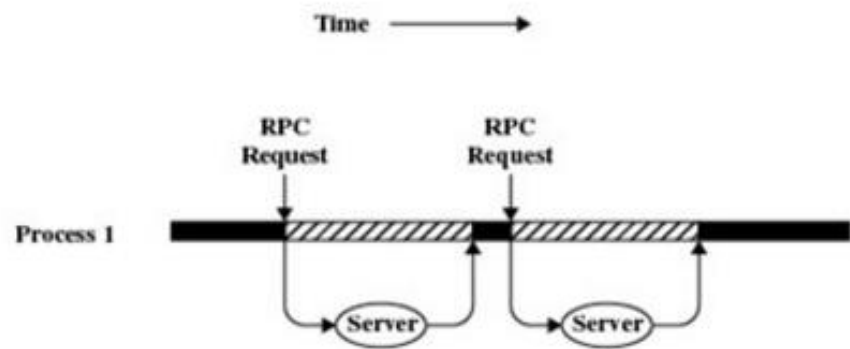
Los Hilos permiten a una aplicación realizar varias tareas al tiempo(concurrencia).



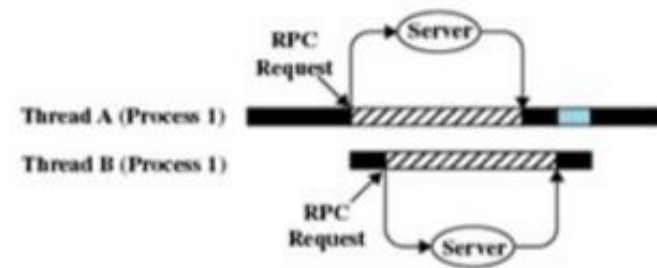
- Los stub en RPC utilizan 1 solo hilo con variables estáticas para pasar información a los servicios.
- Un servidor con hilos debe separar espacio de memoria para el paso de parámetros y valores de retorno antes de crear los hilos.
- En rpcgen se tiene la opción `-M`, que genera stubs multi-thread para paso de argumentos.



Concurrencia en el servidor



Ejemplo de RPC con un hilo



(b) RPC Using One Thread per Server (on a uniprocessor)

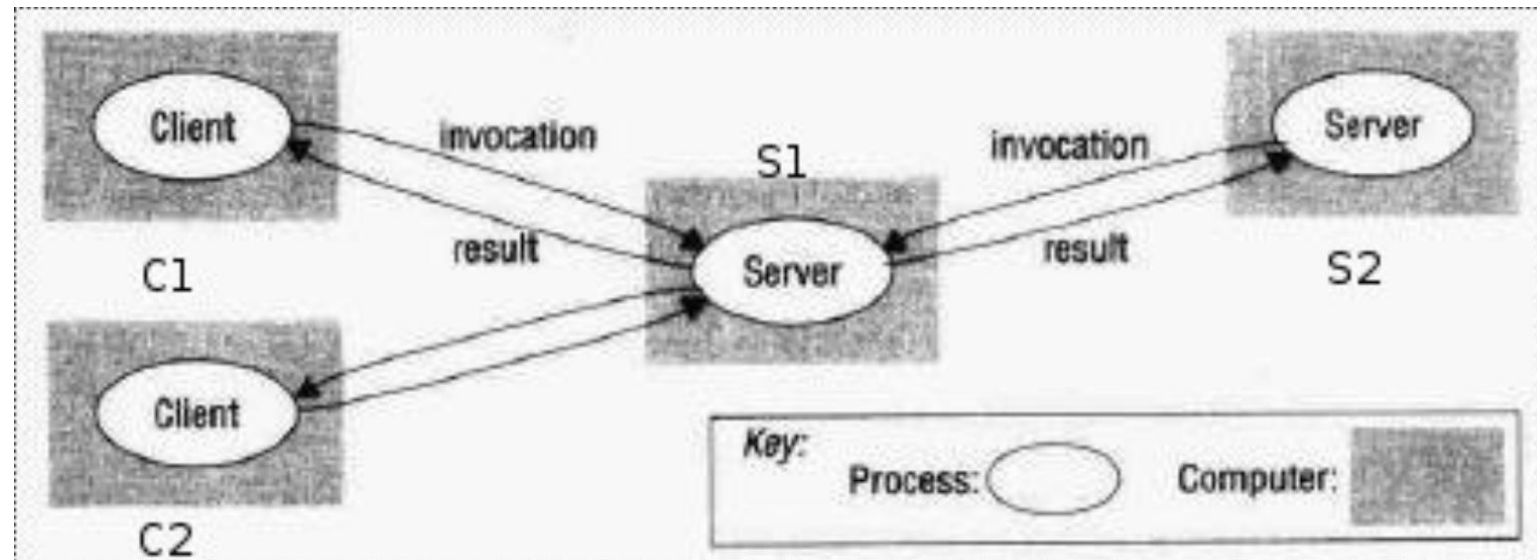
- Blocked, waiting for response to RPC
- Blocked, waiting for processor, which is in use by Thread B
- Running



Ejemplo

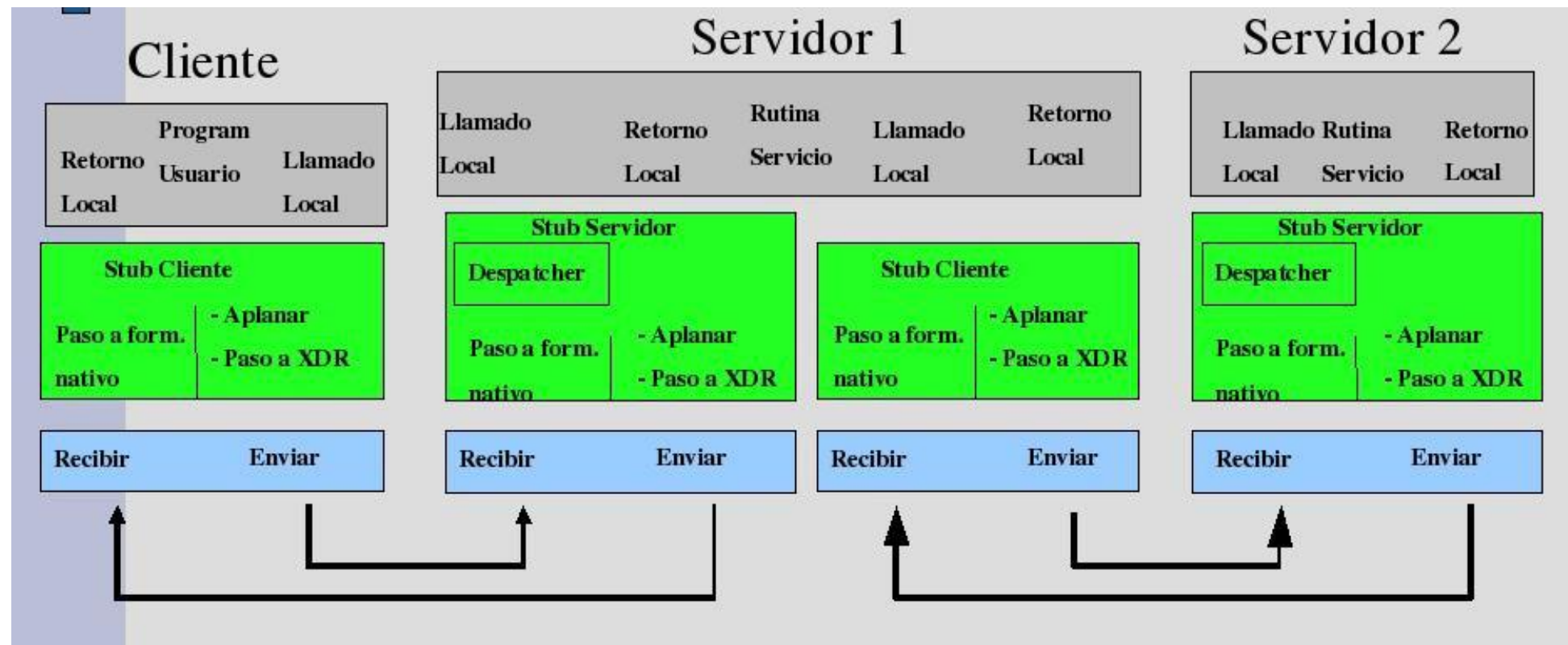
Ejemplo:

Una aplicación posee un servicio compuesto prestado por los procesos S1 y S2 (Ver Figura). El sistema esta basado en RPC. Dibujar la arquitectura de los nodos C1, S1 y S2 durante la petición y respuesta de un servicio. Asumir que el registro del servicio ya fue realizado.



Ejemplo

Solución



Ejemplo arquitectura de nodos

Para las votaciones presidenciales se instalan 5 servidores (S1 a S5) que controlan las votaciones de las 5 regiones geográficas de Colombia. Los votantes se conectan al servidor respectivo dependiendo de la región donde se encuentran.

Los periodistas tanto de radio como TV para obtener los resultados de la jornada electoral deben consultar a un servidor S6 en la ciudad de Bogotá. Dibujar la arquitectura de los nodos cliente, los servidores S1-S5 y S6 de este sistema de votación.



Ejemplo arquitectura de nodos

En una red de bibliotecas universitarias los usuarios que prestan libros se conectan a un servidor para consultar si existen libros disponibles para prestamos en la red de bibliotecas. El sistema cuenta con un servidor central, el cual recibe la petición del usuario, el servidor primero consulta a la biblioteca principal de la red de universidades, si no encuentra libros disponibles, realiza una petición los servidores de las otras bibliotecas de la red. El sistema cuenta con una biblioteca principal y dos bibliotecas alternas.

Dibujar la arquitectura de los nodos cliente y servidores de este sistema de bibliotecas.



Referencias

- Alonso, Jose Miguel, TCP/IP en UNIX: Programación de aplicaciones distribuidas, Edit. Rama, ISBN 970-15-0368-6
- Paco Aylagas, Isabel Muñoz, (2003), Notas y Transparencias de Sistemas Distribuidos, Departamento de Informática Aplicada, Universidad Politécnica de Madrid. Disponible:
http://www.dia.eui.upm.es/cgi-bin/asigfram.pl?cual=sis_dis&nombre=Sistemas-Distribu%EDdos
- NetworkIT TCPaccess, RPC/XDR Programmer's Reference, Version 5.3, Disponible:
<http://www.workers.com.br/manuais/53/html/tcp53/rp/rp.htm>

