75.74 Sistemas Distribuidos I
RPC

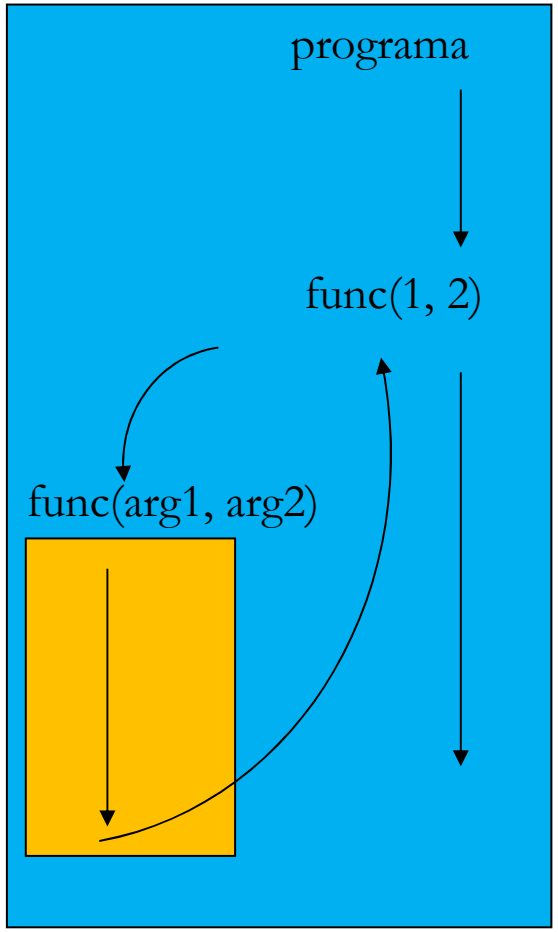
Mariano Méndez

Modelo Cliente/Seridor

- El modelo Cliente/ Servidor tiene algunas desventajas:
  - El paradigma esencialmente se construye entorno a la comunicación la entrada / salida.
  - Esta interface nos fuerza a construir nuestras aplicaciones distribuidas utilizando las funciones read/write, que no es la forma habitual en la que se construyen las aplicaciones no distribuidas.
  - En las aplicaciones centralizadas el modelo de construcción habitual se basa en las llamada a procedimientos /funciones.

RPC

- Llamada a una función o procedimiento local.



RPC: Birrel y Nelson

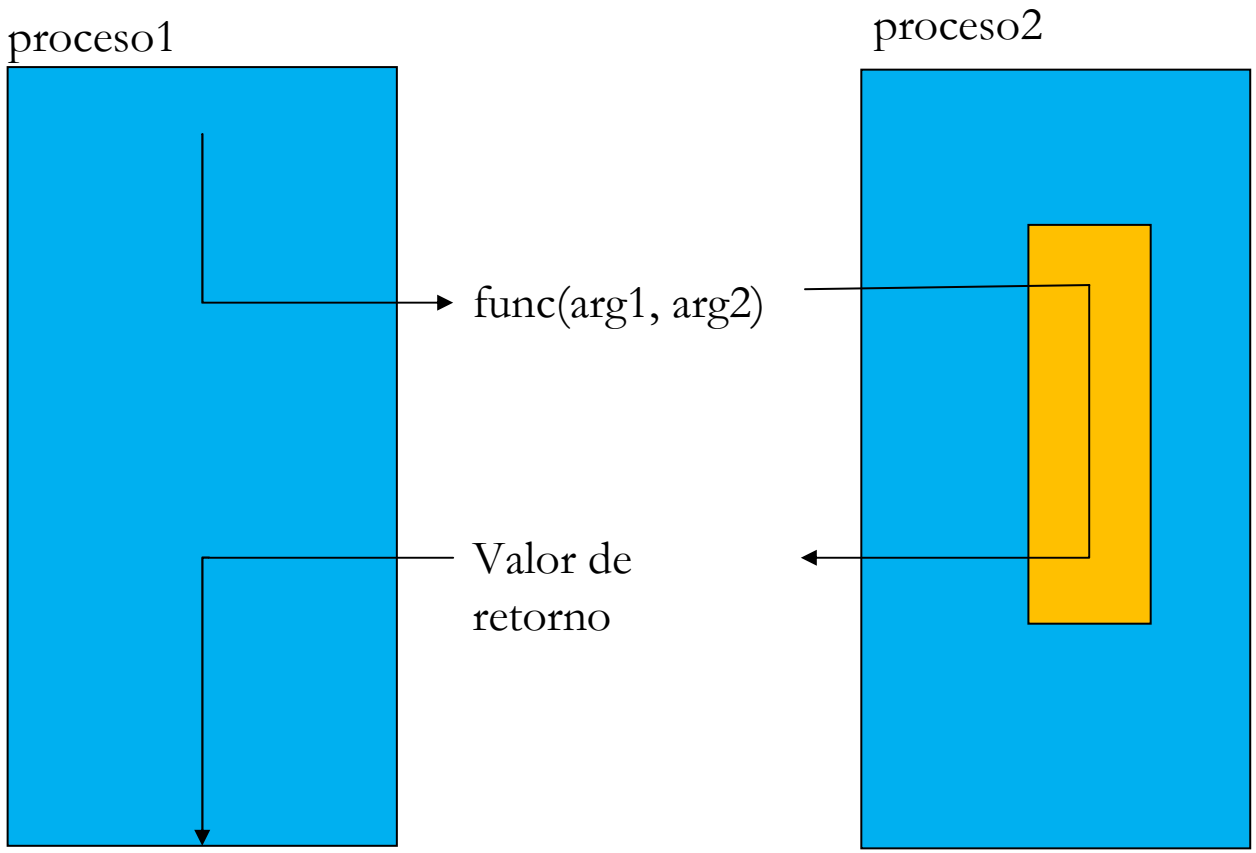
- La idea de **Birrel** y **Nelson** fue:
  - Permitir a los programas que llamasen a procedimientos localizados en otras máquinas.
  - Cuando un proceso en la máquina “A” llama a un procedimiento en la máquina “B”:
    - El proceso que realiza la llamada se suspende.
    - La ejecución del procedimiento se realiza en “B”.

RPC: Birrel y Nelson

- La información se puede transportar de un lado al otro mediante los parámetros y puede regresar en el resultado del procedimiento.

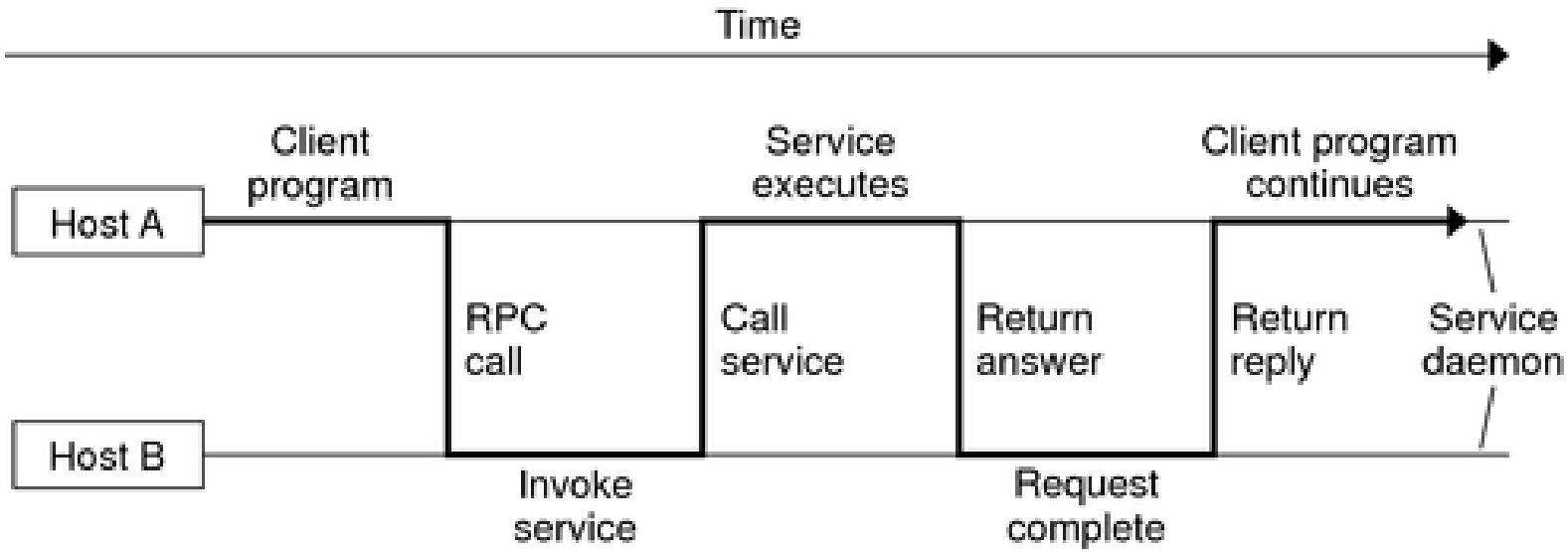
- El programador no se preocupa de una transferencia de mensajes o de la e / s.

RPC



Llamada a una función o procedimiento remoto.

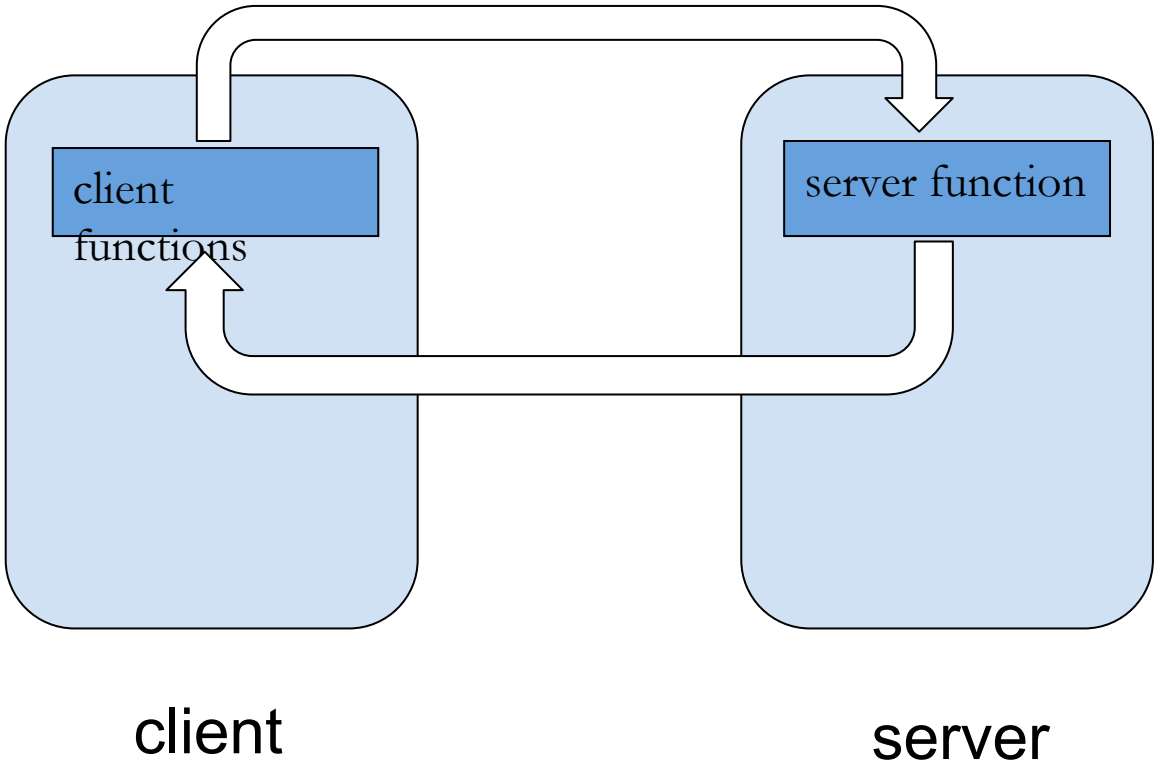
RPC:


Time
Client program
Service executes
Client program continues
Host A
RPC call
Call service
Return answer
Return reply
Service daemon
Host B
Invoke service
Request complete

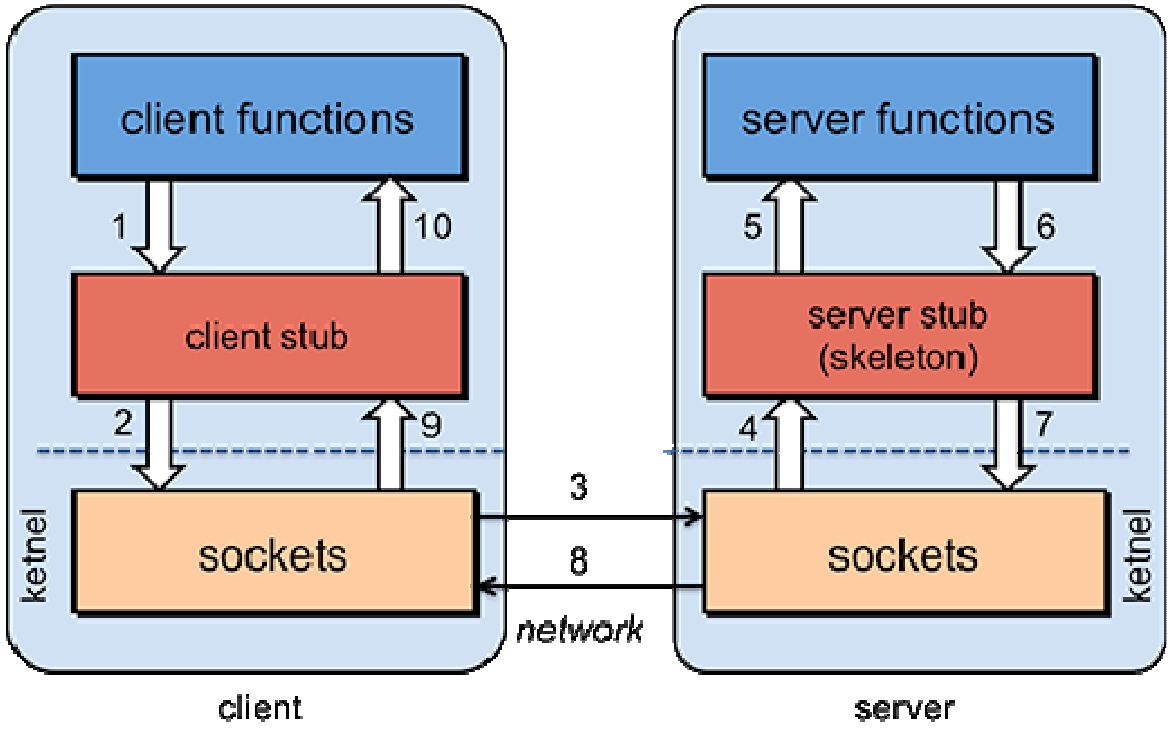
RPC: Definicion de **Birrel** y **Nelson**

The idea of RPC is quite simple. **It is based on the observation that procedure calls are a well-known and well-understood mechanism for transfer of control and data within a program running on a single computer.** Therefore, it is proposed that this same mechanism be extended to provide for transfer of control and data across a communication network. (...) the calling environment is suspended, the parameters are passed across the network to the environment where the procedure is to execute, and the desired procedure is executed there. When the procedure finishes and produces its results, the results are passed backed to the calling environment, where execution resumes **as if returning from a simple single-machine call**.

RPC: Esquema de comunicación

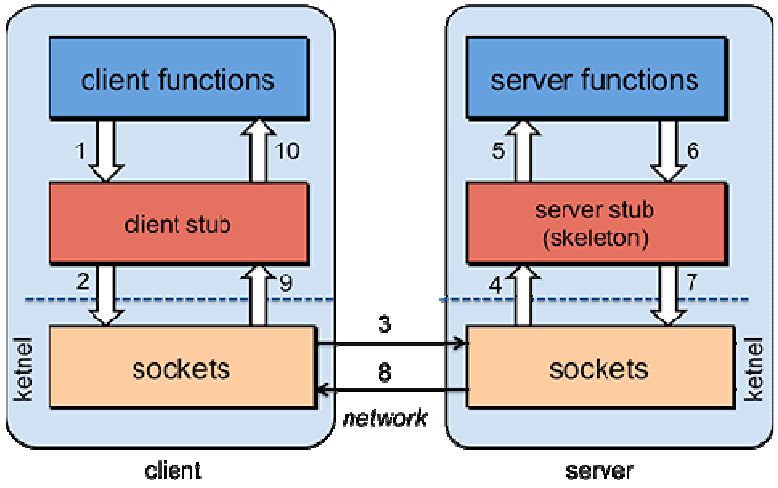


RPC: Esquema de comunicación



RPC: Esquema de comunicación

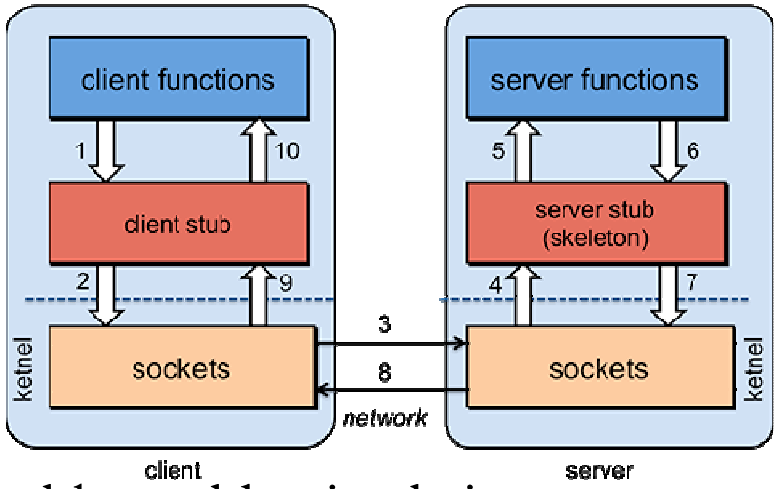
1- El cliente llama a un procedimiento local, llamado el "stub del cliente". Para el proceso cliente, esto parece ser un procedimiento más, porque es un procedimiento local.



Simplemente hace algo diferente ya que el procedimiento real está en el servidor. El stub del cliente empaqueta los parámetros para el procedimiento remoto (esto puede implicar la conversión a un formato estándar) y construye uno o más mensajes de red. El empaquetado de los argumentos en un mensaje de red se llama marshaling y requiere la serialización de todos los elementos de datos en un formato que es un array de bytes plano.

RPC: Esquema de comunicación

2 – Los mensajes son enviados por el stub del cliente al sistema remoto (a través de una System Call hecho al kernel local utilizando el API de Sockets).

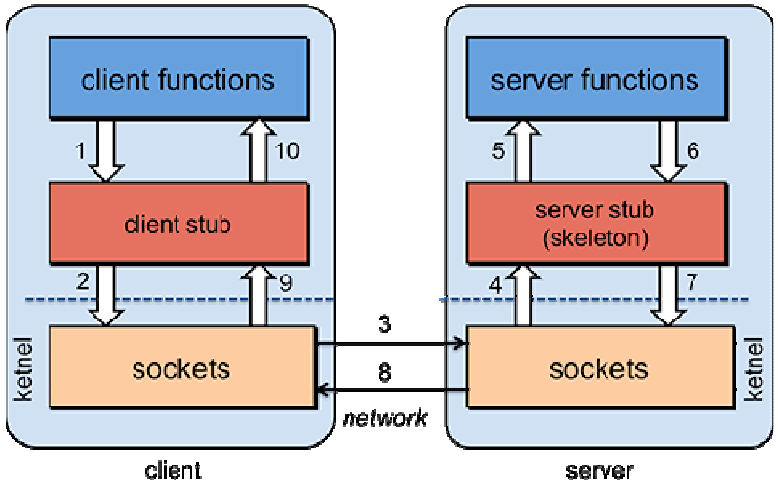


3- Los mensajes son transferidos por el kernel hacia el sistema remoto a través de algún protocolo (ya sea sin conexión u orientado conexión).

4- El stub de servidor, a veces llamado ”skeleton”, recibe los mensajes en el servidor. Se desempaquetan los argumentos que vienen dentro de los mensajes y, si es necesario, se conviertes de un formato de red estándar en una forma específico de la arquitectura de la maquina.

RPC: Esquema de comunicación

5 – El stub del servidor llama a la función en cuestión en el servidor (que, para el cliente, es el procedimiento remoto), pasándole los argumentos que recibió por parte del cliente.



6-Cuando la función del servidor termina su ejecución, el control vuelve al stub de servidor con su valor de retorno.

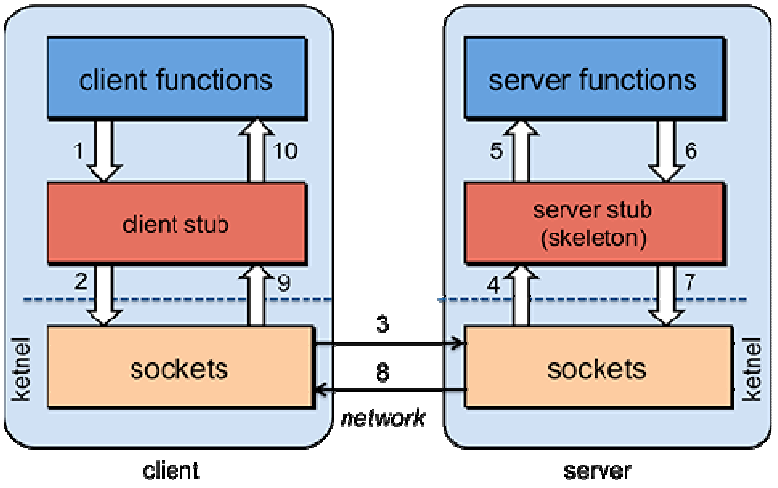
7-El stub del servidor convierte el valor de retorno, si es necesario, y los "marshaliza" en uno o más mensajes para enviar al stub del cliente.

8-Los mensajes son enviados de vuelta a través de la red hacia el stub del cliente.

RPC: Esquema de comunicación

9- El stub del cliente lee los mensajes del kernel local.

10- El stub del cliente devuelve los resultados de la función al cliente, convirtiéndolos de la representación de red a una local, si fuera necesario.



RPC

- 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"***.

RPC: Pasaje de Parámetros

- Por Valor:
  - Solo se copian los datos en los mensajes y se pasan a través de la red
- Pasaje por Referencia:
  - Es complejo
  - El problema en el pasaje de parámetros es si se pasan punteros, ya que un puntero solo tiene sentido en el espacio de direccionamiento (address space) en donde se encuentra el proceso.
  - Una solución puede ser pasar los elementos del vector en su totalidad dentro del mensaje, así el stub del servidor podría llamar a la rutina apuntando a un buffer propio, donde guardó el vector que le pasaron como parámetro. Si el server escribe sobre esta parte de la memoria, el stub del servidor tiene que encargarse de devolverlo al cliente, así puede copiarlo modificado. Luego los parámetros por referencia son reemplazados por un mecanismo de copy/restore

RPC: Semántica

- La semántica de llamar a un procedimiento local es simple: un procedimiento se ejecute exactamente una vez, cuando lo llamamos.

- Con un procedimiento remoto, el aspecto exactamente una vez es bastante difícil de lograr.

RPC: Semántica

- Un procedimiento remoto se puede ejecutar:
  - 0 veces si el servidor se colgó o murió antes de ejecutar el código del servidor.
  - una vez si todo funciona bien.
  - una o más veces si el servidor dejó de funcionar después de regresar al stub del servidor pero antes de enviar la respuesta. El cliente no recibirá la respuesta de retorno y puede decidir volver a intentarlo, ejecutando así la función más de una vez. Si no intenta de nuevo, la función se ejecuta una vez.
  - más de una vez si el cliente entra en timeOut y retransmite. Es posible que la solicitud original puede haberse retrasado. Ambos pueden ser ejecutados (o no).

RPC: Semántica

- Sistemas RPC generalmente ofrecen ya sea la semántica de al menos una o la semántica de como máximo una vez o una elección entre ellos.
- Uno tiene que entender la naturaleza de la aplicación y la función de los procedimientos remotos para determinar si es seguro para llamar a una función posiblemente más de una vez.
- Si una función se puede ejecutar cualquier número de veces sin daño, es idempotente (por ejemplo, la hora del día, funciones matemáticas, leer datos estáticos). De lo contrario, es una función noidempotent (por ejemplo, añadir o modificar un archivo).

RPC: Implementaciones

- ***ONC RPC*** llamada a procedimiento remoto de Sun.
- ***DCE/RPC*** llamada a procedimiento remoto de Open Software Foundation.
- ***Distributed Component Object Model*** (DCOM), Modelo de Objetos de Componentes Distribuidos de Microsoft.
- ***Java Remote Method Invocation*** (RMI), Invocación de Métodos Remotos para Java.
- **ORB -** CORBA.
- **XML-RPC**.

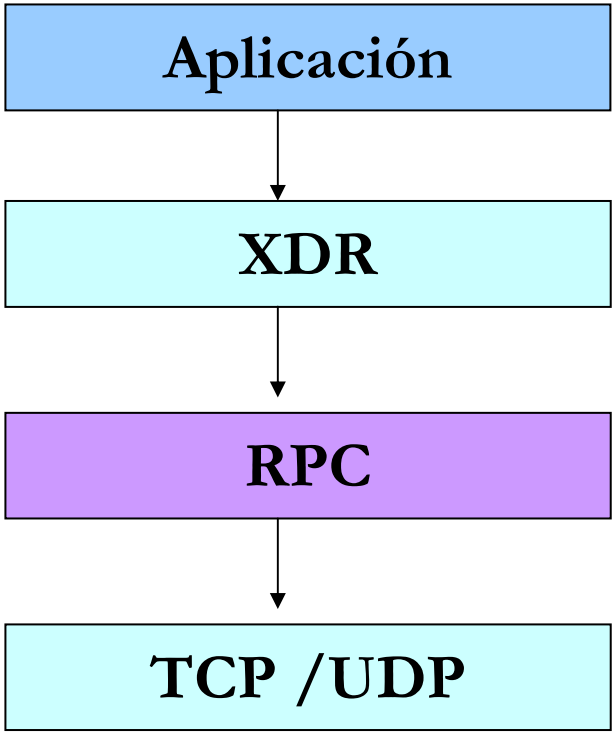
RPC: implementacion SUN

ONC RPC, abreviación del inglés *Open Network Computing Remote Procedure Call*, es un protocolo de llamada a procedimiento remoto (RPC) desarrollado por el grupo ONC de Sun Microsystems.

RPC: ONC

- XDR (eXternal Data Representation):
- Protocolo de presentaciónón de datos, según el modelo OSI.
- Se utiliza para la representaciónón externa de datos.
- Una forma estándar de codificar datos de modo que sean transportables entre distintos sistemas

RPC: ONC



RPC: Implementaciones

Para ofrecer un mecanismo de llamada a procedimientos remotos sintácticamente equivalente al de llamada local el entorno RPC debe proporcionar y dar soporte a una infraestructura que ofrezca transparencia en la invocación remota.

Objetivo: es deseable que el programador del sistema distribuido no perciba la diferencia entre llamada local y llamada remota (transparencia).

RPC: Implementaciones

Para las tareas adicionales (empaquetado de datos, comunicación) un entorno RPC debe ofrecer

- Código adicional para soportar la llamada remota a un procedimiento concreto normalmente generado de forma automatica
- Librerias y herramientas complementarias (runtime) para soportar la ejecucion del entorno RPC.

RPC: Componentes

- Idea clave:
  - uso de proxies o representantes, tanto del cliente como del servidor
- Representante del servidor en la maquina cliente (stub)
  - realiza el papel de servidor en la maquina cliente
- Representante del cliente en la maquina servidor (skeleton)
  - realiza el papel de cliente en la m maquina servidor
- Proporcionan transparencia en la llamada remota

RPC: Componentes

- Generados automáticamente en base a la interfaz definida para el Procedimiento Remoto
  - Programador solo debe programar el código del procedimiento remoto (servidor) y el código que hace la llamada remota. (cliente)

RPC: IDL

- XDR especifica cómo aplanar/desaplanar los datos intercambiados entre stub y skeleton en un formato independiente de la arquitectura
- Usado para definir el tipo y la estructura de los argumentos y valores de retorno de los procedimientos remotos
- Emplea como lenguaje IDL (interface definición language) una ampliación de XDR que permite la definición de procedimientos
  - Permite identificar procedimientos y versiones con un núm.
  - Especifica argumentos de entrada + valor de retorno (no la Implementacion)

RPC: IDL

- Proceso Remoto tiene una identificación única:
  - Programa
  - Versión
  - Procedimiento

RPC: IDL

```
program display_prg {
   version display_ver  {
        int print_hola (void) = 1;
    } = 1;
} = 0x20000001;
```

RPC: IDL

```
struct entrada {
   int arg1;
   int arg2; };
program CALCULADORA {
     version CALCULADORA_VER {
            int sumar(entrada) = 1;
            int restar(entrada) = 2;
            int multiplicar(entrada) = 3;
            int dividir(entrada) = 4;
     } = 1;
} = 0x30000001;
```

IDL

Número definidos para utilizar:

0x00000000 - 0x1FFFFFFF: definidos por Sun (portmapper, nfs, nis, ...)

0x20000000 - 0x3FFFFFFF: definidos por el usuario

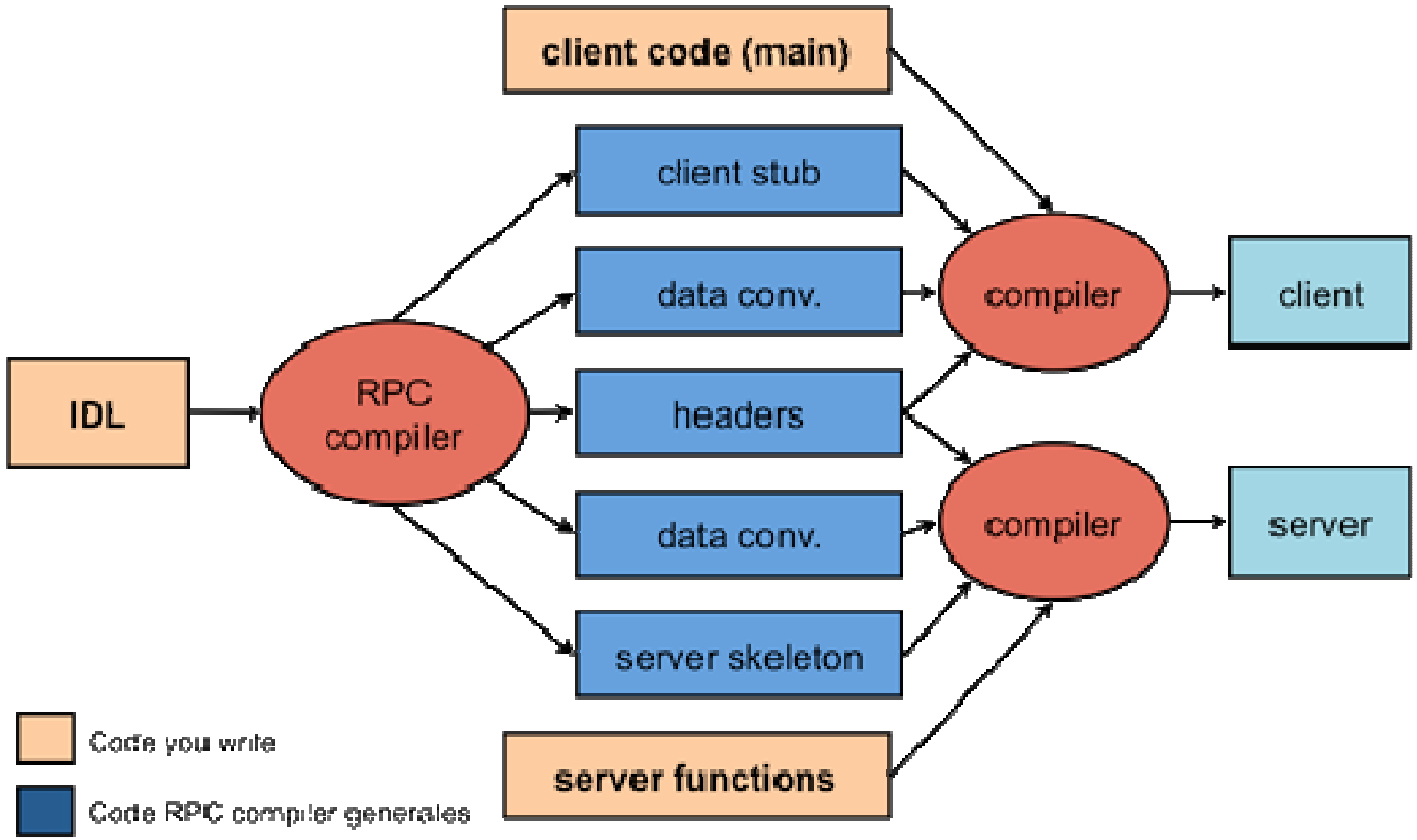
0x40000000 - 0x5FFFFFFF: reservado

0x60000000 - 0xFFFFFFFF: reservado

RPC: Componentes

- El compilador de interfaces RPCGEN
- Elementos generados con RPCGEN a partir de la interfaz (con -a):
  - Proxies
    - Stub
    - Skeleton
  - Librerias(.h)
  - Makefile
  - Cliente.c
  - Servidor.c

RPC: Componentes



RPC: Componentes

- El compilador de interfaces RPCGEN
- Elementos generados con RPCGEN a partir de la interfaz (con -a):
  - Proxies
    - Stub
    - Skeleton
  - Librerias(.h)
  - Makefile
  - Cliente.c
  - Servidor.c

RPC: Paso a Paso

1) Se edita el archivo .x en nuestro caso hola.x:

```
program display_prg {
    version display_ver  {
       int print_hola (void) = 1;
    } = 1;
} = 0x20000001;
```

RPC: Paso a Paso

2) Se abre una consola y se ejecuta el comando "***rpcgen –a hola.x***". Como producto de la ejecucion se han de haber generado los sigientes archivos:
   - hola_clnt.c
   - hola_svc.c
   - hola.h
   - Makefile.hola
   - hola_client.c
   - hola_server.c
   - hola_xdr.c

RPC: Paso a Paso

hola_clnt.c: Stub del cliente, se encarga del marshalling y de la comunicación con el kernel del cliente.

hola_svc.c: Skeleton o Stub del servidor, registra al servidor en el sistema unix para indicarle que atienda a las llamadas, desempaqueta y realiza la llamada concreta en el servidor.

hola.h: Aquí están los prototipos de nuestras funciones. Cualquier cliente que quiera usarlas, deberá hacer un include de este fichero. El prototipo no es exactamente como esperaríamos. A cada función le añade en el nombre unas sufijos para indicar el número de versión. Define también otras constantes como nombre de programa, número de versión, etc, que son útiles a la hora de hacer la conexión con el servidor.

Makefile.hola: Es el fichero Makefile necesario para compilar todos los demás ficheros de código generados por rpcgen. El comando de unix "make -f Makefile.hola" nos generará los ejecutables de cliente y servidor.

RPC: Paso a Paso

hola_xdr.c: Como RPC permite llamadas de clientes a servidores que estén en máquinas distintas y, por tanto, puedan tener una arquitectura distinta, es necesario traducir los parámetros y resultados a un "código" universal, independiente de las máquinas. Si los parámetros son tipos básicos (int, float, char, etc), el sistema unix ya tiene unas funciones de conversión (xdr_int(), xdr_float(), etc). Si los parámetros, son estructuras definidas por nosotros, las funciones de conversión hay que hacerlas. rpcgen genera automáticamente dichas funciones, las pone en el fichero hola_xdr.c

RPC: hola_server.c

```
/*
*This is sample code generated by rpcgen.
*These are only templates and you can use them
* as a guideline for developing your own functions. */
#include "hola.h”
int *
print_hola_1(argp, rqstp)
        void *argp;
        struct svc_req *rqstp;
{
        static int  result;
   /*
    * insert server code here
    */
   return(&result);
}
```

RPC: hola_client.c

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

void display_prg_1(host)
char *host;
{
                CLIENT *clnt;
                int  *result_1;
                char*  print_hola_1_arg;
                clnt = clnt_create(host, display_prg, display_ver, "udp");
                if (clnt == NULL) {
                                clnt_pcreateerror(host);
                                exit(1);
                }

   result_1 = print_hola_1((void*)&print_hola_1_arg, clnt);
                if (result_1 == NULL) {
                                clnt_perror(clnt, "call failed:");
                }
   clnt_destroy( clnt );
}
main(argc, argv)
int argc;char *argv[];
{
                char *host;
                if(argc < 2) {
                                printf("usage: %s server_host\n", argv[0]);
                                exit(1);
                }
                host = argv[1];
                display_prg_1( host );
}
```

RPC: make

```
gcc -g -DRPC_SVC_FG -c -o hola_clnt.o hola_clnt.c
gcc -g -DRPC_SVC_FG -c -o hola_client.o hola_client.c gcc -g -DRPC_SVC_FG -c -o
hola_xdr.o hola_xdr.c
gcc -g -DRPC_SVC_FG -o hola_client hola_clnt.o hola_client.o hola_xdr.o
gcc -g -DRPC_SVC_FG -c –o hola_svc.o hola_svc.c
gcc -g -DRPC_SVC_FG -c –o hola_server.o hola_server.c
gcc -g -DRPC_SVC_FG –o hola_server hola_svc.o hola_server.o hola_xdr.o
```

RPC: Paso a Paso

En una terminal ejecutamos “make –f Makefile.hola” y se obtienen los binarios del cliente y del servidor.

RPC: cosas varias

- rpcgen depende de la plataforma
- Makefile no siempre se genera
- Hay que levantar el servicio de rpc :

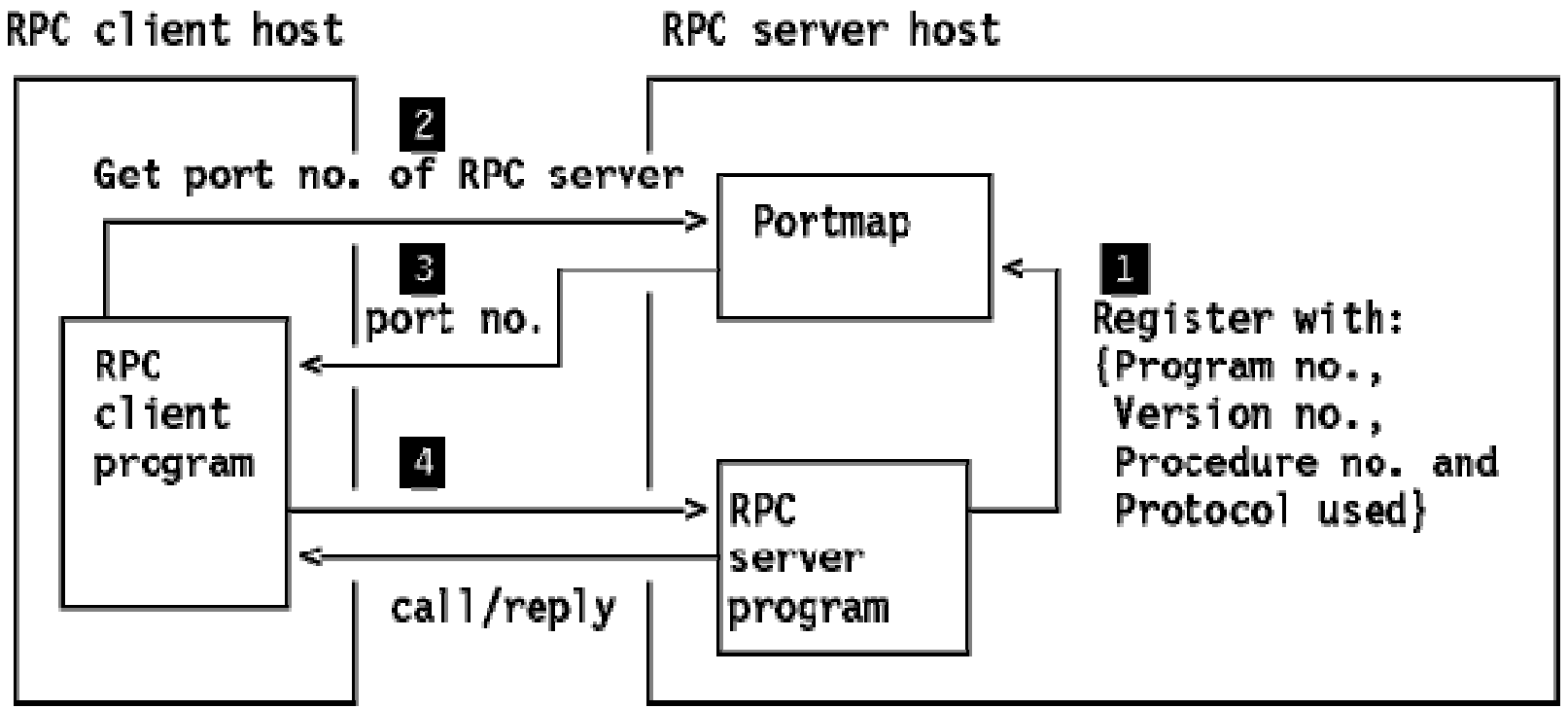
OS	command
Mac OS X	launchctl start com.apple.portmap
Linux	/sbin/portmap
*BSD	/usr/sbin/rpcbind

RPC: Portmap

Existe una aplicación en el Servidor, *portmap*, encargada de mapear los puertos con los programas remotos.

El funcionamiento es sencillo, *Portmap* se encuentra escuchando en el puerto 111 el cliente le pregunta por el puerto utilizado por un programa específico y *portmap* devuelve el puerto y así ya sabemos a que puerto conectarnos.

RPC: Portmap


RPC client host
RPC server host
2
Get port no. of RPC server
Portmap
3
port no.
1
Register with:
{Program no.,
Version no.,
Procedure no. and
Protocol used}
RPC
client
program
4
RPC
server
program
call/reply

Preguntas ?