UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INFORMÁTICA Y ANÁLISIS NUMÉRICO

ASIGNATURA ***SISTEMAS OPERATIVOS***

2º DE GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 3

Llamadas a Procedimientos Remotos (RPC)

Profesores:

Enrique García Salcines

Javier Pérez Rodríguez

Indice de la práctica

1 Objetivo de la práctica

La presente práctica persigue familiarizar al alumnado con la creación y gestión de llamadas a procedimientos remotos en sistemas distribuidos. En concreto estudiaremos una de las implementaciones en lenguaje C, denominada Sun-RPC muy extendida en entornos tipo UNIX.

En una primera parte se dará una introducción teórica sobre RPC, siendo en la segunda parte de la misma cuando se practicarán los conceptos aprendidos mediante programación en C, utilizando las herramientas que proporciona el estándar y la librería libgc.

2 Recomendaciones

El lector debe completar las nociones dadas en las siguientes secciones con consultas bibliográficas, tanto en la Web como en la biblioteca de la Universidad, ya que unos de los objetivos de las prácticas es potenciar su capacidad autodidacta y su capacidad de análisis de un problema. Es recomendable que, aparte de los ejercicios prácticos que se proponen, pruebe y modifique otros que se encuentren en la Web (se dispone de una gran cantidad de problemas resueltos en C sobre esta temática), ya que al final de curso deberá acometer un examen práctico en ordenador como parte de la evaluación de la asignatura.

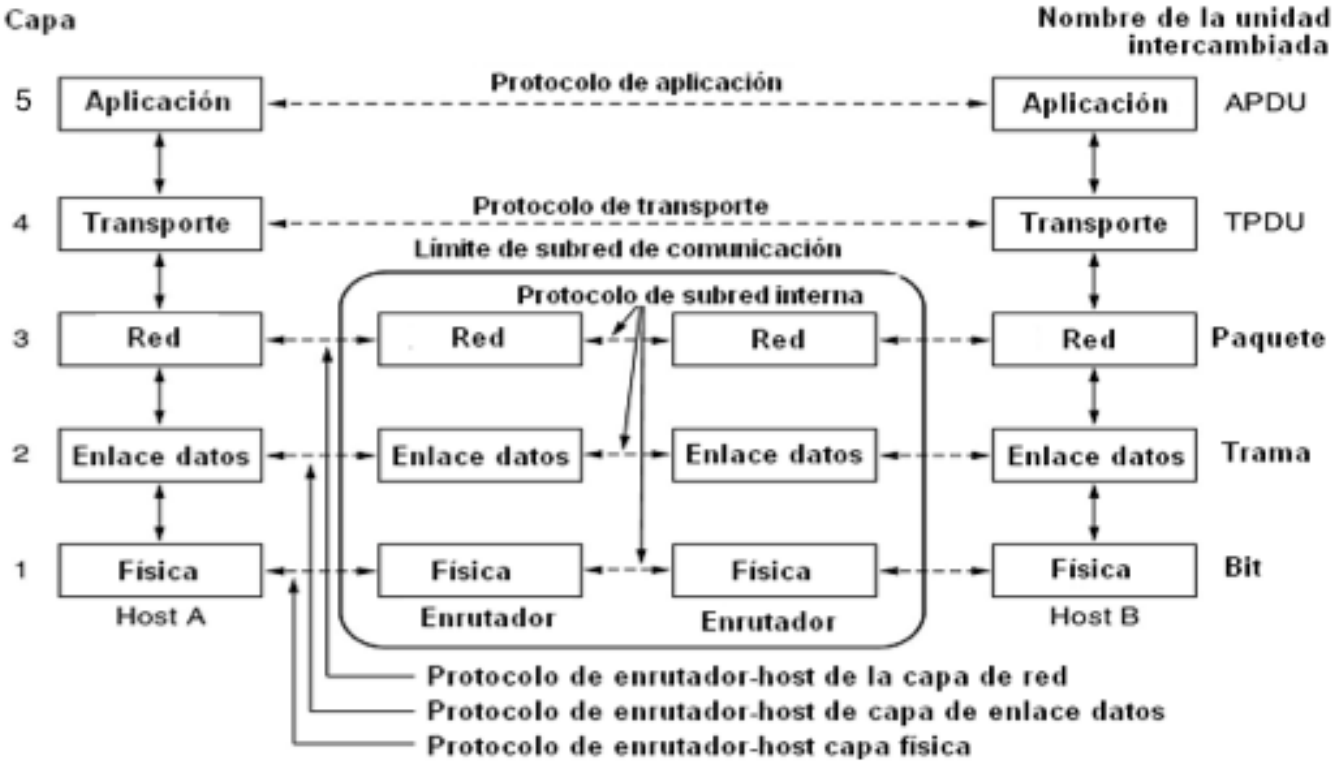
Al igual que se le instruyó en las asignaturas de Metodología de la Programación, es recomendable que siga unas normas y estilo de programación claro y consistente. No olvide tampoco comentar los aspectos más importantes de sus programas, así como añadir información de cabecera a sus funciones (nombre, parámetros de entrada, parámetros de salida, objetivo, etc). Estos son algunos de los aspectos que se también se valorarán y se tendrán en cuenta en el examen práctico de la asignatura.

3 Conceptos teóricos

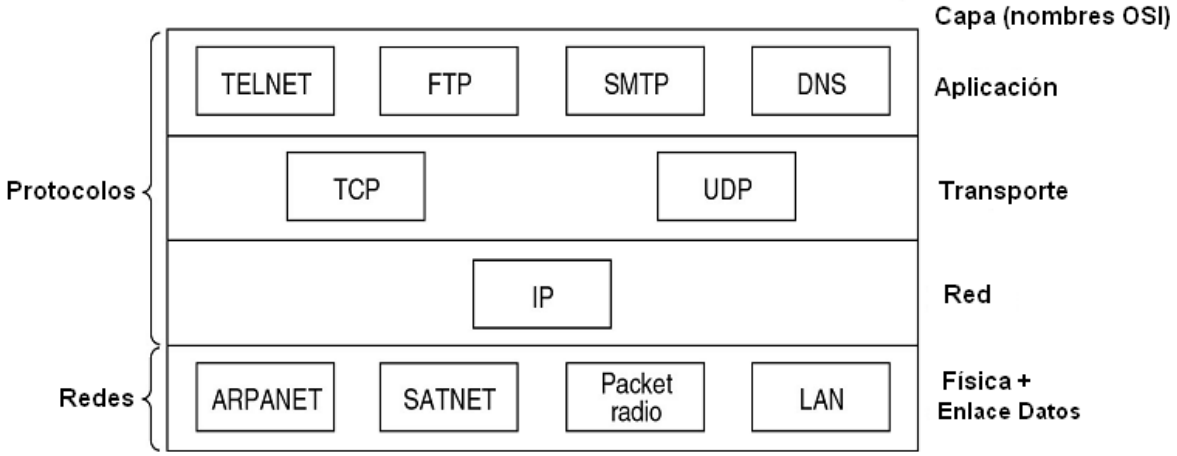
En las primeras redes de computadoras el hardware era lo fundamental. Esta estrategia ya no funciona y actualmente el software de Redes, está altamente estructurado y consiste en un modelo de capas perfectamente diferenciadas, cada una de las cuales realizan una determinada función. Podemos definir entonces los siguientes conceptos asociados al modelo.

- **Servicio:** indica qué hace la capa, o sea qué servicios brinda a la capa superior, no la forma en que la capa superior tiene acceso al servicio. Define el aspecto semántico (significado) de la capa.
- **Interfaz:** indica a los procesos que están sobre una capa cómo acceder a la misma. Tiene que ver con la sintaxis (cuáles son los parámetros y resultados que se esperan)
- **Protocolo:** implementa los servicios, cada capa debe decidir qué protocolos utilizar,

siempre y cuando se proporcione el servicio ofrecido



Por tanto, las entidades o procesos en máquinas distintas pueden comunicarse entre sí, a través de los distintos protocolos que tienen implementados cada capa.



La evolución de los paradigmas de comunicación entre procesos ha sido la siguiente:

- En el **nivel menos abstracto**, la comunicación entre procesos implica la transmisión de series binarias sobre una conexión, utilizando una transferencia de datos de bajo nivel, serie o paralelo. Este paradigma de comunicación puede ser válido para el desarrollo de software de un driver de red por ejemplo.

- El siguiente nivel es un paradigma bien conocido, denominado interfaz de programación de aplicaciones de **sockets (API de SOCKETS).** Por medio del paradigma de sockets dos procesos intercambian datos por medio de una estructura lógica denominada sockets, habiendo uno de cada tipo en cada extremo. Los datos a transmitir se escriben sobre el sockets. En el otro extremo, un receptor lee o extrae datos del sockets.
- Los paradigmas de **llamadas a procedimientos remotos (RPC)** o de invocación de métodos remotos proporcionan una mayor abstracción, permitiendo al proceso realizar llamadas a procedimientos o la invocación de métodos de un proceso remoto, con la transmisión de datos como argumentos o valores de resultados.

En computación distribuida [1], la llamada a procedimiento remoto (en inglés, remote procedure call, RPC) es un programa que utiliza una computadora para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas. El protocolo que se utiliza para esta llamada es un gran avance sobre los sockets de Internet[2] usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando estas encapsuladas dentro de las RPC.

Las RPC son muy utilizadas dentro de la comunicación cliente-servidor. Siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando este de vuelta el resultado de dicha operación al cliente.

RPC ofrece un conjunto de ventajas como la compartición de recursos, la concurrencia, y la alta escalabilidad, sin embargo conlleva una mayor complejidad que otros paradigmas de programación, además de los posibles aspectos de seguridad a tener en cuenta en la comunicación cliente-servidor.

A continuación se detallan distintas implementaciones de RPC:

- **Sun-RPC (ONC-RPC: Open Network Computing-RPC):** RPC definido por Sun MicroSystems dentro del desarrollo del sistema de ficheros distribuidos NFS (*Network File System*) muy extendido en entornos Unix y basados en éste.
- **DCE/RPC (Distributed Computing Environmen RPC):** RPC definido por la Open Software Foundation
- **Java-RMI:** invocación de métodos remotos en Java
- **CORBA (Common Object Requesting Broker Architecture):** soporta la invocación de métodos remotos bajo un paradigma orientado a objetos en diversos lenguajes.
- **SOAP (Simple Object Access Protocol):** protocolo RPC basado en el intercambio de datos (parámetos+resultados) en formato XML

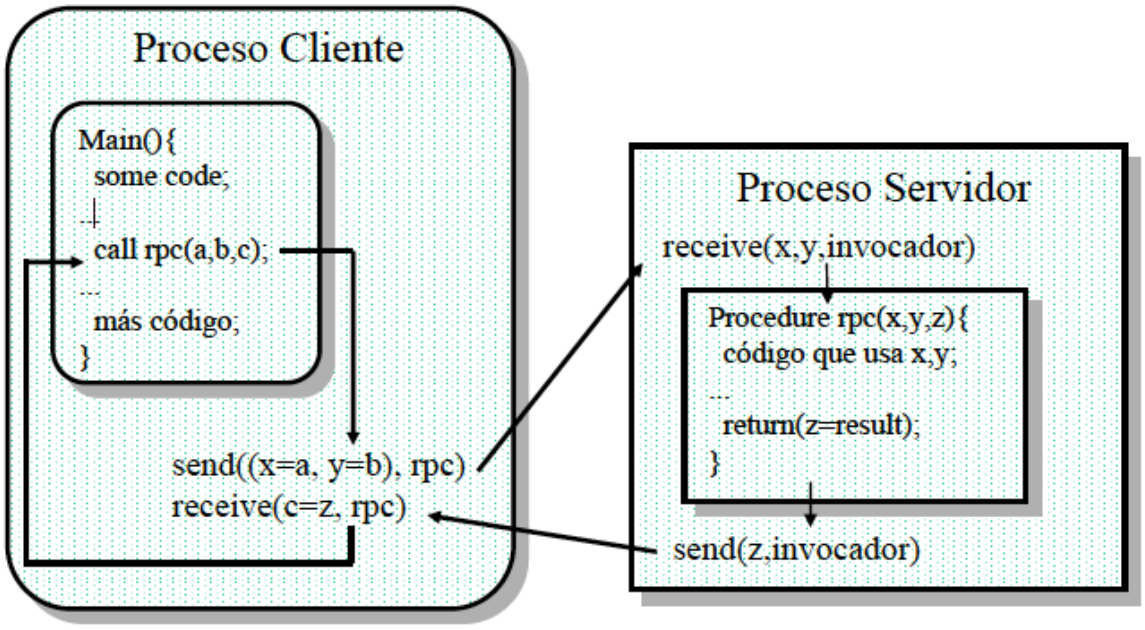
[1] https://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida
[2] https://es.wikipedia.org/wiki/Socket_de_Internet

- **DCOM (Distributed Component Object Model):** Modelo de Objetos de Componentes Distribuidos de Microsoft, con elementos de DCE/RPC
- .**NET Remoting:** Infraestructura de invocación remota de .NET

El mecanismo de llamada a procedimiento remoto del Open Network Computing de Sun (ONC RPC[3]) es uno de los más extendidos, debido al indiscutible éxito de algunos de los productos que se sustentan sobre el mismo, como NFS. Eso hace que resulte una opción interesante a la hora de familiarizarse, desde un punto de vista docente, con la programación de servicios utilizando RPCs, ya que cualquier sistema UNIX dispone en su instalación básica de todo el software requerido para su utilización. Es por ello que usaremos esta implementación en la presente práctica.

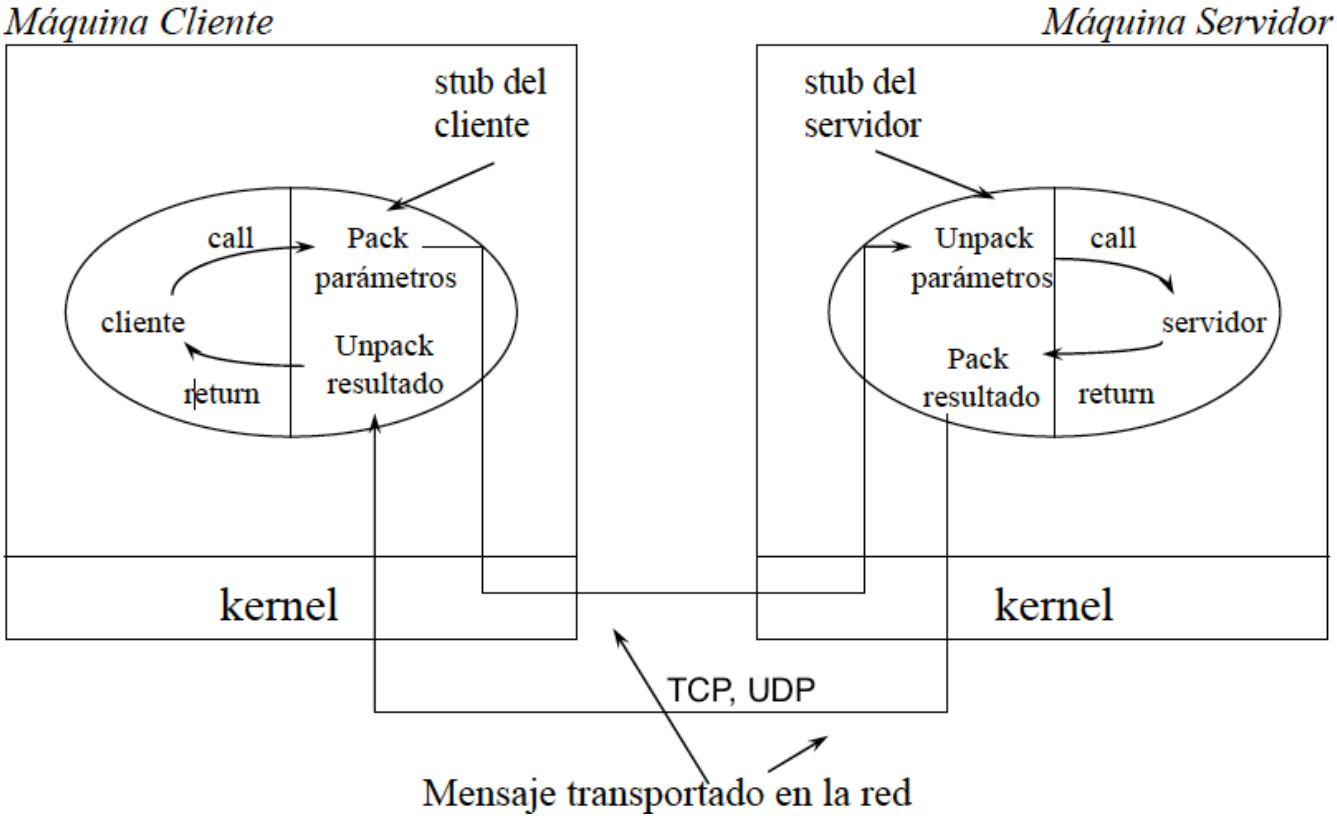
3.1 Modo de operación

A continuación se muestra la operativa de una llamada a procedimiento remoto para llamar a desde un proceso Cliente a un procedimiento que se ejecuta en un proceso Servidor.



Veamos en realidad lo que ocurre a más bajo nivel en la máquina del cliente y la máquina del servidor introduciendo los conceptos de StuB del Cliente y Stub del Servidor.

[3] https://es.wikipedia.org/wiki/ONC_RPC



Creación servidor ⇒ añadir stub de cliente y stub servidor

Clientes usan stub para llamar a los servicios ofrecidos por servidor

- No envían mensajes a través de las primitivas de comunicación.
- Realizan llamada a procedimiento stub con parámetros adecuados.
- Stub empaqueta parámetros, envía *send* al núcleo y llama a *receive*.
- Núcleo envía mensaje a núcleo remoto.

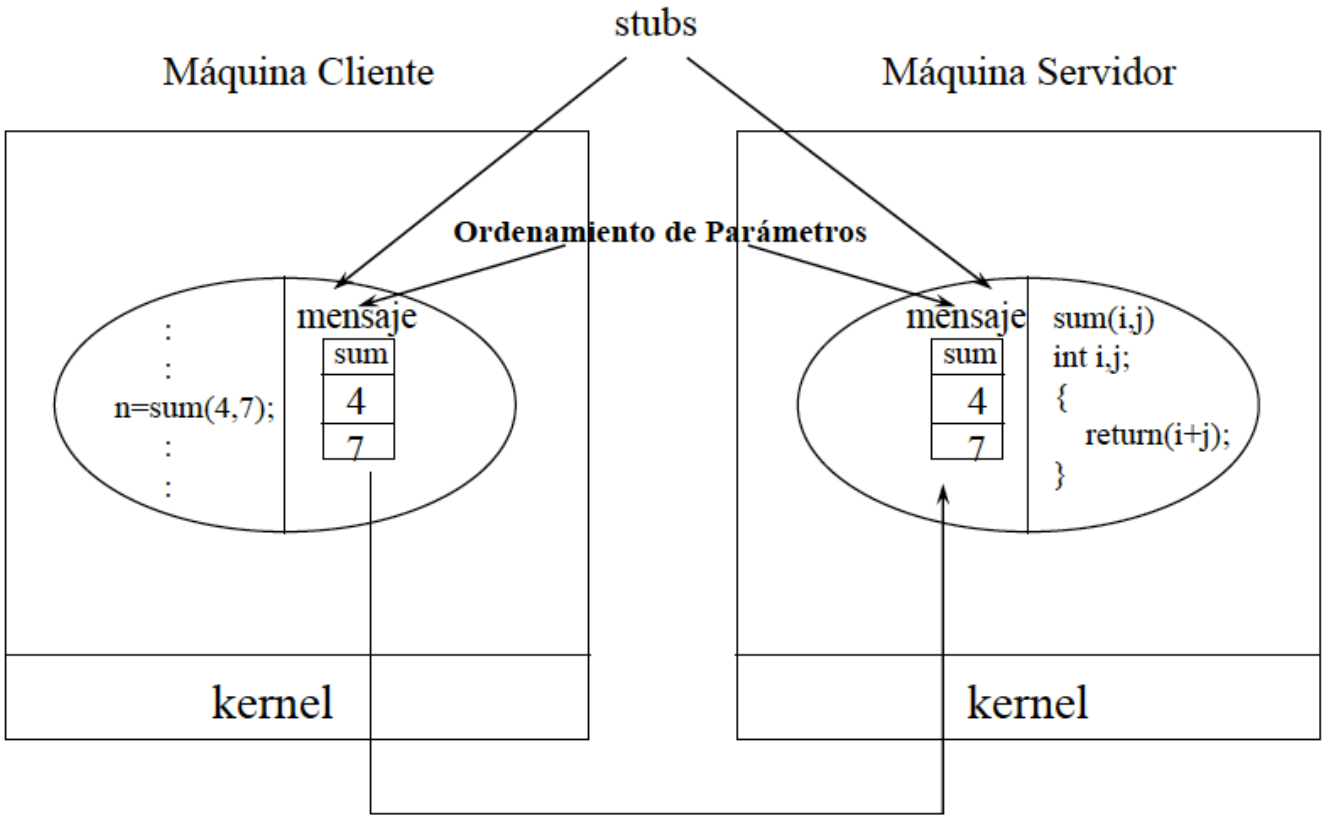
Máquina servidor→ Stub servidor llama a *receive* y espera mensaje

- Mensaje llega a núcleo remoto ⇒ lo deja en el buzón del stub que desempaqueta parámetros y realiza llamada al servidor.
- Stub servidor recibe resultados ⇒ los empaqueta para formato *send* y llama a *send* para enviárselos al cliente.
- Nuevo ciclo con *receive* a la espera de una nueva petición.

Mensaje resultados llega al núcleo cliente → buzón correspondiente

- Cliente llamó a *receive* ⇒ recoge los resultados del buffer.
- Stub desempaqueta los parámetros y devuelve el resultado.

Transferencia de Parámetros en RPC



Se busca ofrecer un entorno de programación lo más similar posible a un entorno no distribuido. El sistema **RPC oculta los detalles de implementación** de esas llamadas remotas:

- Implementa la llamada remota mediante un diálogo petición respuesta
- Se encarga de enviar/recibir mensajes para comunicar ambas partes
- Se encarga de gestionar los contenidos de esos mensajes (empaquetado y formateado de datos)

Usualmente el desempeño empleando RPC disminuye entre 10 y 100 veces en comparación con una llamada a un procedimiento local.

Normalmente el servidor está siempre corriendo y a la espera de que algún cliente llame a alguna de sus funciones.. Cuando el cliente llama a una función del servidor, la función se ejecuta en el servidor y el cliente detiene su ejecución hasta que el servidor termina.

En el código del programa servidor básicamente hay que seguir los siguientes pasos:

- Codificar las funciones que van a ser llamadas siguiendo un determinado mecanismo.

- Informar al sistema operativo (unix) de un nombre, una versión y funciones que publica.
- Bucle para la espera que alguien llame a alguna de sus funciones.

Mientras que el programa cliente debe:

- Establecer una conexión con el servidor.
- Llamar a las funciones.
- Cerrar la conexión con el servidor.

3.2 Componentes de programación ONC-RPC

Para implementar una arquitectura cliente-servidor con RPC es necesario entender y dominar los componentes y pasos necesarios para generar los ejecutables del servidor y del cliente.

3.2.1 IDL (interface definition language

RPC emplea **XDR**[4] **(eXternal Data Representation)** que es un protocolo estándar para la descripción y codificación de datos entre los stubs y que garantiza portabilidad entre sistemas de arquitecturas diferentes. Se utiliza para definir el tipo y estructura de los argumentos y valores de retorno de los procedimientos remotos

IDL (Interface definition language) es una ampliacion de XDR que permite:

- Identificar procedimientos y versiones con un número
- Especifica argumentos de entrada + valor de retorno (**NO LA IMPLEMENTACIÓN**)

Tipos de datos y gramática del fichero IDL:

- Entero con signo:
  int identifier;
- Entero sin signo:
  unsigned int identifier;
- Enumeración:
  Enum {name-identifer = constant, ...} identifier
  Ejemplo: enum {RED=2, YELLOW=3, BLUE=5} colors;
- Punto flotante de Precisión simple (32 bits)
  float identifier;
- Punto flotante de Precisión doble (64 bits)
  double identifier;
- Cadena de caracteres:

[4] https://es.wikipedia.org/wiki/External_Data_Representation

string identifier <TAM>;

- Arrays:

  tipo_dato identifier <TAM>;

- Estructura:

  struct {
  tipo_datoA varA;
  tipo_datoB varB;
  .....
  }

Se extiende el lenguaje XDR para incluir procedimiento y versionado

Procedimiento:

```
id_tipo nombre_proc (id_tipo) = valor;
```

Lista de procedimientos:

```
Procedimiento; lista de procedimientos;
```

Versión:

```
version id_version { lista de procedimientos }= valor;
```

Lista de versiones:

```
Versión; lista de versiones;
```

Programa:

```
program nombre_programa { lista de versiones } = valor;
```

Ejemplos: los archivos IDL tienen extensión .x

Interface.x

```
program SERVIDOR_CALCULO {
    version SC_TERCERA {
            int duplica (int) = 1;
            float ENTRE3 (int) = 2;
    } = 3;
} = 0x201ABCBF;
```

sumador.x

```
struct datos {
    int sum1;
    int sum2;
}
program SUMADOR {
    version SUM_TERCERA {
            int suma (datos) = 1;
            double div3 (float) = 2;
    } = 3;
} = 0x201ABCBF;
```

Al final de cada figura podemos observar un número en hexadecimal (0x201ABCBF) que el el valor del número de programa. Este número no debería repetirse para dos servicios distintos.

Existe un número razonable de servicios creados con los RPCs de SUN Microsystems, entre los cuales destacan NFS, portmap, mount, yellowpage y otros. En vista de que el número de programas es un valor de 32 bits, SUN ha realizado una distribución de ese rango de números y es la que se muestra a continuación:

Desde	Hasta	Valores asignados por
0x00000000	0x1FFFFFFF	Sun Microsystems, Inc.
0x00000000	0x3FFFFFFF	El Administrador del Sitio
0x00000000	0x5FFFFFFF	Transitorios
0x00000000	0xFFFFFFFF	Reservados

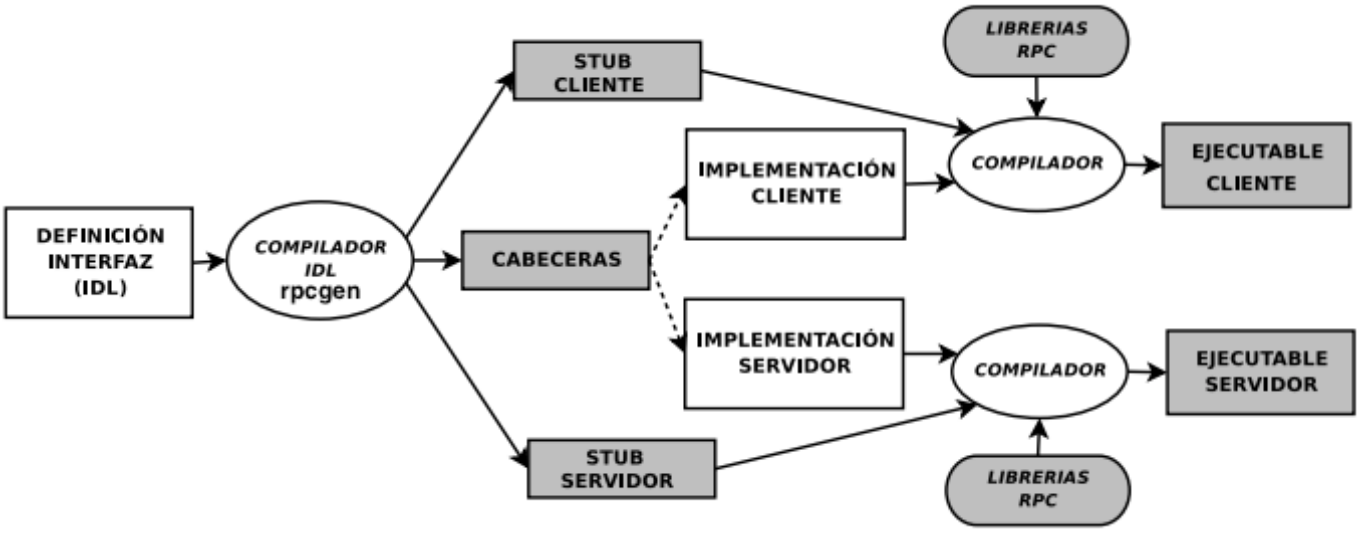
Nombre	Num. Asignado	Descripción
portmap	100000	Port mapper
rstatd	100001	rstat, rup, y perfmeter
rusersd	100002	Remote users
nfs	100003	Network file systme
ypserv	100004	yp (ahora se llama NIS)
mountd	100005	mount, showmount
dbxd	100006	DBXprog(debugger)
ypbind	100007	NIS binder
walld	100008	rwall, shutdown
yppasswdd	100009	yppasswd
etherstatd	100010	Ethernet statistics

Nombre	Num. Asignado	Descripción
rquotad	100011	rquotaprog,quota,rquota
sprayd	100012	spray
selection_svc	100015	selection sevice
dbsessionmgr	100016	unify, netdbms, dbms
rexd	100017	rex, remote_exec
office_auto	100018	alice
lockd	100020	klmprog
lockd	100021	nlmprog
statd	100024	status monitor
bootparamd	100026	bootstrap
pcnfsd	150001	NFS para PC

Números de Prpgramas Asignados en RPC's

3.2.2 RPCGEN

Es un compilador para IDL, que toma la definición de la interfaz de un procedimiento remoto y genera el "stub" del cliente y el "stub" del servidor.



Opciones del programa rpcgen:

Comunes a la mayoría de las máquinas:

-c	Genera únicamente los filtros XDR.
-h	Genera únicamente el fichero de cabecera .h
-l	Genera únicamente el extremo del cliente.
-m	Genera únicamente el extremo del servidor.
-s [udp|tcp]	Sólo usa el protocolo especificado.

Para Solaris y Linux:

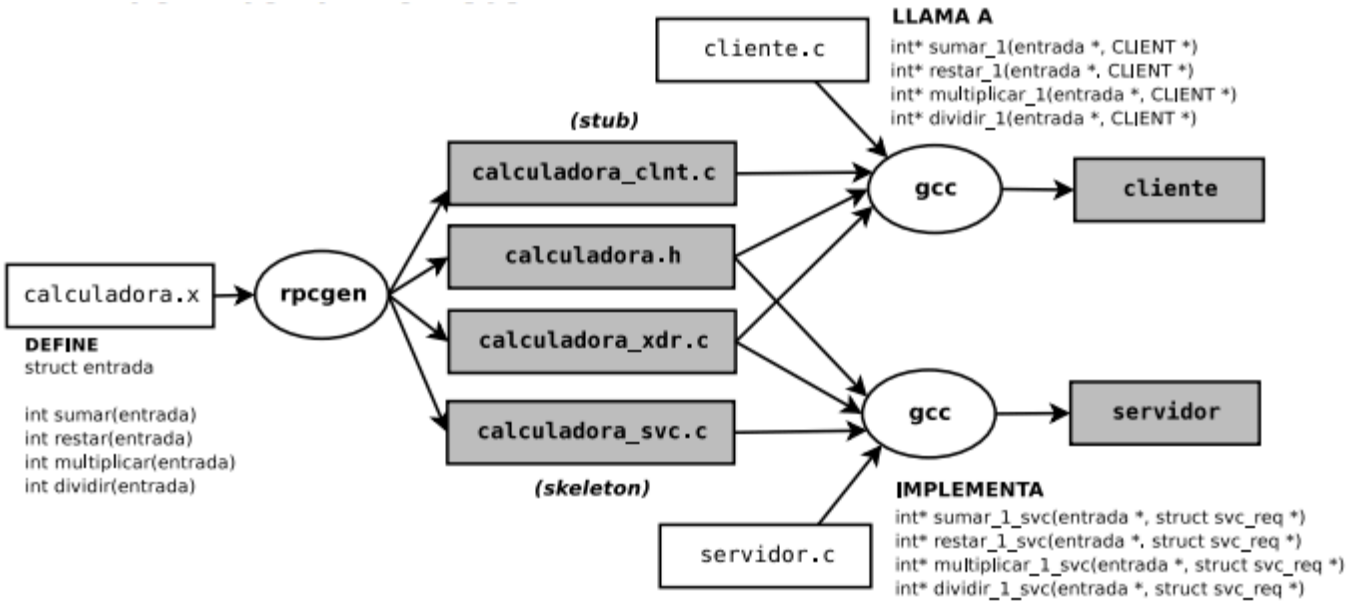
-Sc	Genera un esqueleto de cliente.
-Ss	Genera un esqueleto de servidor.
-Sm	Genera un fichero `makefile` para la aplicación.
-a	Genera todos los ficheros.

Ejemplo:

$ rpcgen –a calculadora.x

y si no hay errores, se generan de manera automática los siguientes archivos:

calculadora_clnt.c: código del stub del cliente
calculadora _svc.c: código del stub del servidor o skeleton
calculadora _xdr.c: código para empaquetar/desempaquetar datos XDR (se usan tipos complejos)
calculadora.h: fichero de cabecera
calculadora_server.c: esqueleto del programa servidor
calculadora_client.c: esqueleto del programa cliente
Makefile: fichero makefile para enlazar todos los archivos y generar los código objeto.

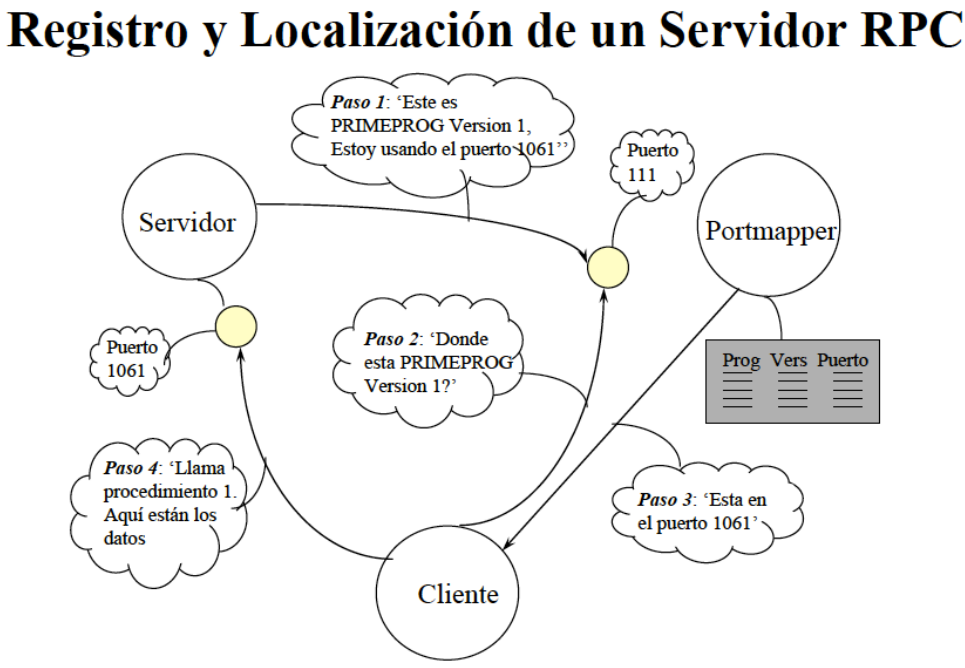


Esquema de generación de los stubs del cliente y del servidor

3.2.3 Portmapper

Es el proceso responsable de la localización de procesos remotos:

- Los servidores registran con el portmapper los procedimientos remotos que exportan
- El portmapper queda a la escucha (**puerto 111**) y redirecciona las peticiones de accesos a procedimientos hacia sus respectivos puertos locales de escucha. Actúa a modo de demonio.



3.3 Pasos para la generación del cliente y del servidor

A continuación se detallan cada uno de los pasos que hay que realizar para implementar una llamada a procedimiento remoto (RPC) utilizando el modelo ONC-RPC:

Paso 1: Diseñar el archivo IDL de definición de interfaces remotas . Este fichero como se comentó debe tener extensión .x

Ejemplo: ***suma.x***

```
struct entrada {
        int arg1;
        int arg2;
};

program SUMA {
        version SUMA_VER {
                int sumar(entrada) = 1;
        } = 1;
} = 0x30000001;
```

Paso 2: Ejecutar el compilador IDL (XDR ampliado)

```
rpcgen –a suma.x
```

Esto generará los stubs (.c), los esqueletos del programa cliente y el servidor, la cabecera del proyecto calculadora.h y además de un archivo Makefile

Nota: si estamos trabajando en GNU Linux (por ejemplo, Ubuntu, Thinstation) es necesario agregar la librería RPC al Makefile. Por tanto, tenemos que modificar la siguiente línea agregando –ltirpc

```
LDLIBS = -lnsl -ltirpc
```

Paso 3: Modificar el código esqueleto del servidor (***suma_server.c***) para implementar los servicios del servidor que se especificaron en el fichero IDL.

```
// This is sample code generated by rpcgen.

#include "suma.h"

int *sumar_1_svc(entrada *argp, struct svc_req  *rqstp)
{
        static int  result;
```

```
        // insert server code here
        result = argp->arg1 + argp->arg2;

        return(&result);
}
```

Nota: La variable sobre la que se escribe el valor a devolver debe declararse como **static** para que su dirección de memoria no sea sobrescrita al salir de la función

Paso 4: Modificar el código esqueleto del cliente (***calculadora_client.c***) para implementar los servicios del servidor que se especificaron en el fichero IDL.

```
// This is sample code generated by rpcgen.

#include "suma.h"

void suma_1(char *host)
{
        CLIENT *clnt;
        int  *result_1;
        entrada  sumar_1_arg;

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

        sumar_1_arg.arg1 = 5;
        sumar_1_arg.arg2 = 8;
        result_1 = sumar_1(&sumar_1_arg, clnt);
        if (result_1 == NULL) {
                clnt_perror(clnt, "call failed:");
        } else printf("Suma=%d\n", *result_1);

        clnt_destroy( clnt );
}



main(int argc, char *argv[])
{
        char *host;

        if(argc < 2) {
                printf("usage: %s server_host\n", argv[0]);
                exit(1);
```

```
        }
        host = argv[1];
        suma_1( host );
}
```

Paso 5: Ejecutar el fichero Makefile para generar los códigos objetos y ejecutables del cliente y el servidor.

make –f Makefile.suma

o si renombramos el archivo como Makefile bastaría con poner **make**

Paso 6: Ejecutar el servidor con priovilegios de administración

./suma_server &

Para comprobar que el servicio se está ejecutando podemos utilizar la aplicación **rpcinfo**, en concreto

Las dos últimas líneas corresponden con nuestro servidor.

```
.............................
 100000        2   udp    111  portmapper
 100000        2   tcp    111  portmapper
 0x30000001    1   udp  60659
 0x30000001    1   tcp  60674
 .............................
```

Paso 7: Ejecutar el programa cliente.

./suma_client localhost

Salida por pantalla:

Suma = 13

Resta = 4

Demos

El archivo **holamundo.x** es el archivo IDL de un programa RPC cuyo servidor devuelve "Hola TuNombre" cuando le pasas al programa cliente como argumento, tu nombre propio.

El archivo **calculadora.x** es el archivo IDL del ejemplo que se ha descrito anteriormente (7 pasos) al que se le han agregado otras operaciones aritméticas.

Por último, el archivo **arrays.x** es un archivo IDL que demuestra como pasar arrays al servidor y devolverlos al cliente.

4 Ejercicios Prácticos

1) Implementar una Calculadora "Especial" en el servidor remoto de manera que el cliente pueda llamar a las siguientes servicios de cálculo:
   a) La media aritmética de N números enteros que se pasen como argumento
   b) El máximo de los N números
   c) El factorial de un número

2) Crear una "base de datos" en el servidor utilizando un archivo de texto que contenga información de los clientes de una empresa. En concreto, se guardarán separados por coma, los siguientes campos:
   - Nombre y apellidos
   - Edad
   - Provincia

Ejemplo de registros del fichero:

Victor Sosa Sánchez, 35, Córdoba
Sonia Sanchez Perez, 27, Sevilla
Javier Ortiz Hernandez 45, Córdoba
Enrique Javier Hernandez, 23, Sevilla
……..

Implementar los siguientes servicios en el servidor:

a) Devolver todos los registros de clientes cuyo apellido coincida con una cadena de entrada que representa el apellido a buscar.
b) Devolver todos los registros de clientes que sean menores de una edad determinada y que vivan en una provincia dada.