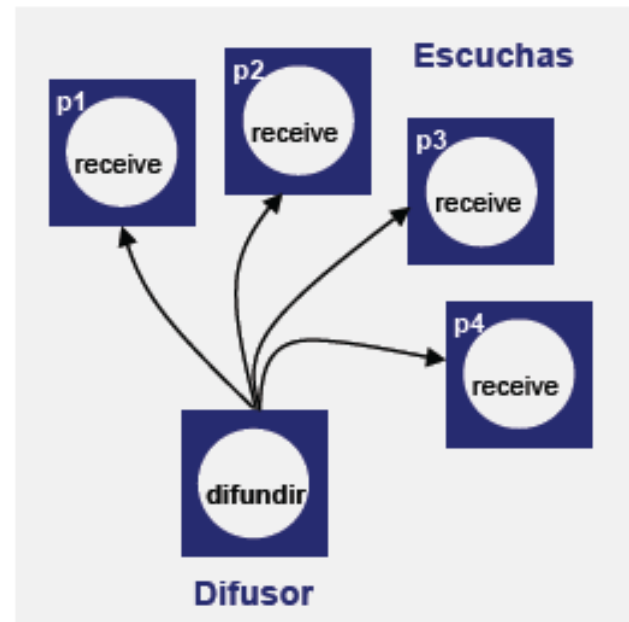
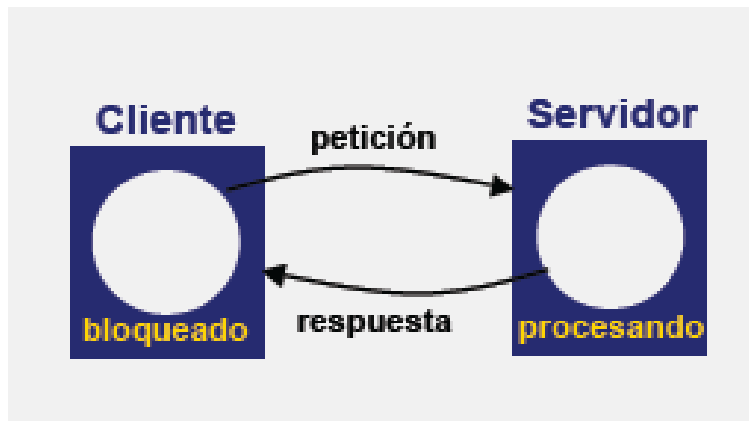


Cliente Servidor

Comunicación



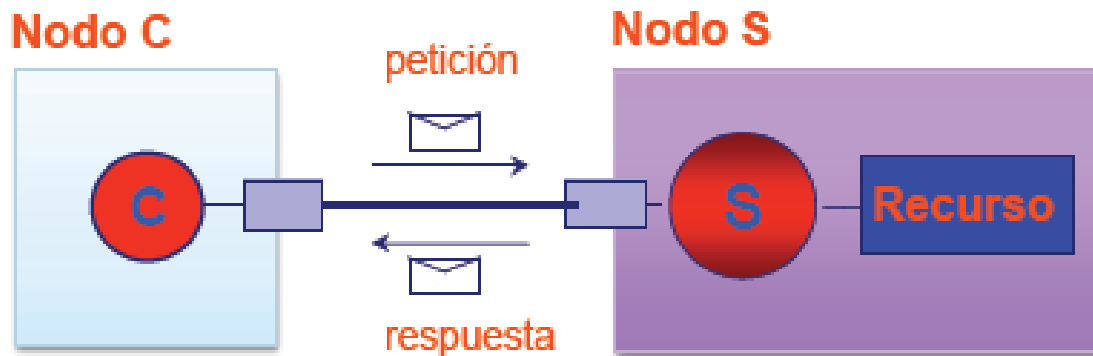
Comunicación

- Cliente -servidor: la comunicación es iniciada por un proceso, cliente que realiza una petición a otro proceso servidor, el cual responde. Puede seguir diversos modelos:
 - Basada en mensajes: sockets
 - Basada en llamadas a procedimientos remotos o RPC's (Remote Procedure Calls): RPC de Sun
 - Basada en invocaciones a objetos remotos: RMI en Java, CORBA
- Difusión a grupos: la comunicación es iniciada por un proceso difusor que envía una información a un grupo de procesos escuchas.

El modelo cliente servidor

- Basado en asociar un proceso servidor a los recursos que quieren compartirse o hacerse visibles a otros nodos.
- Existen procesos con dos patrones de comportamiento diferenciados:
 - Servidores: gestionan un recurso y ofrecen servicios relacionados con este
 - recurso a otros procesos.
 - Reciben peticiones de procesos clientes, vía mensajes
 - las ejecutan en su nombre y
 - devuelven una respuesta.
 - Clientes: formulan peticiones de servicio a los servidores

El modelo cliente servidor



CLIENTE

```
...  
send(serv, petition, ...);  
recv(serv, respuesta, ...);  
...
```

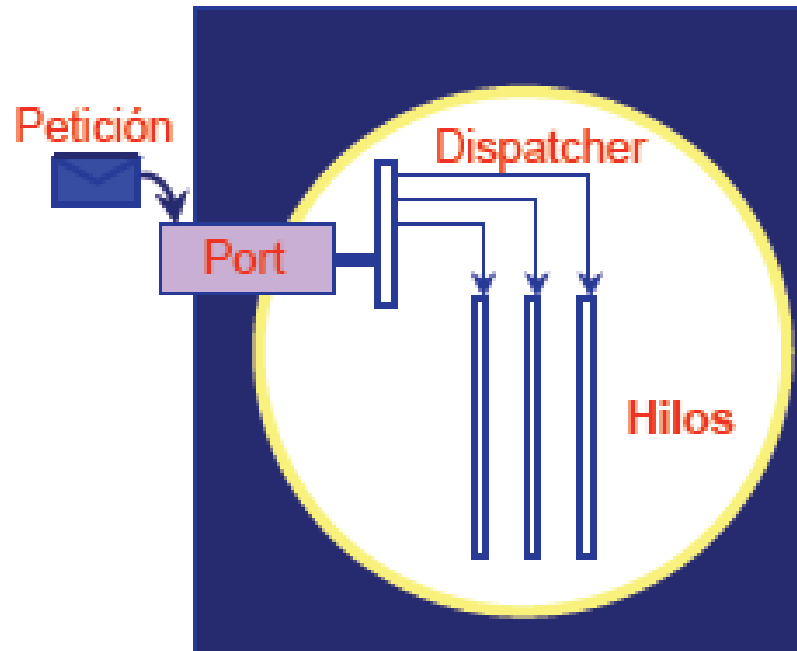
SERVIDOR

```
repeat  
  recv(any, petition, ...);  
  respuesta= trabajar(petition);  
  ret=remite(petition);  
  send(ret, respuesta, ...);  
forever;
```

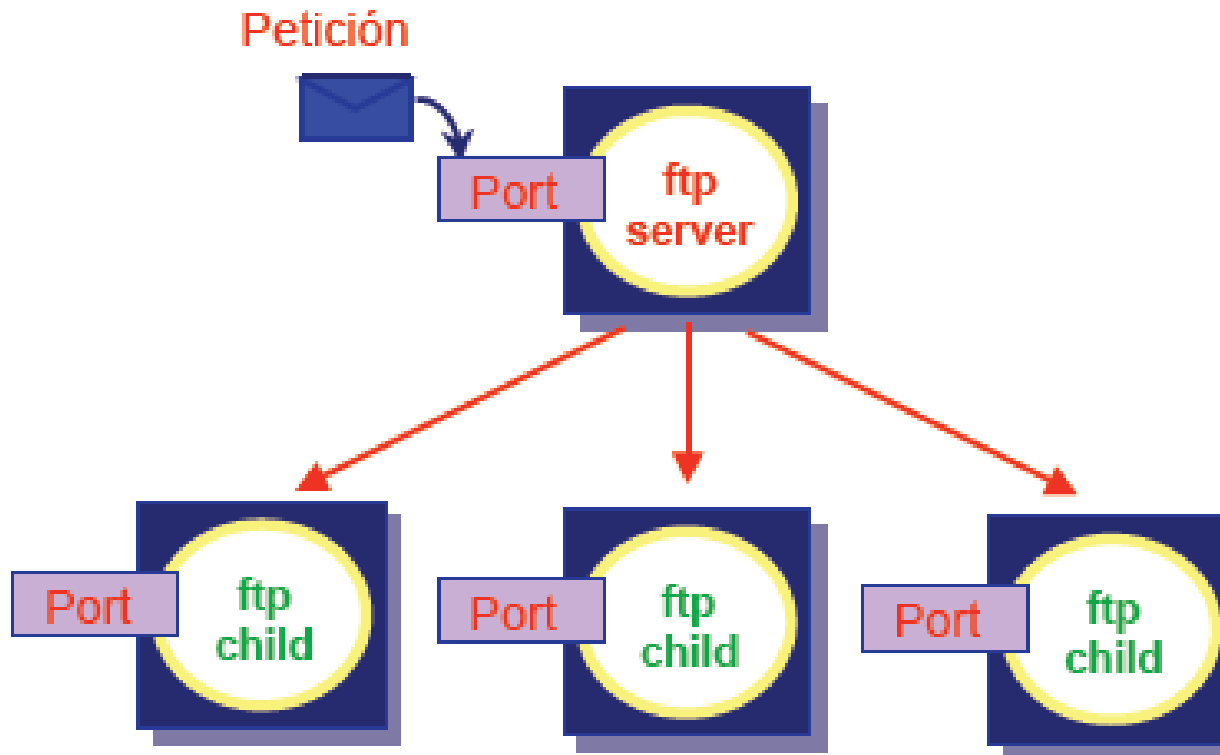
Tipos de servidores

- Un proceso servidor puede adoptar dos estrategias básicas para la gestión del recurso que custodia:
 - Servidores secuenciales: Atienden las peticiones secuencialmente.
 - Adecuados para recursos reutilizables en serie
 - Pueden introducir cuellos de botella
 - Servidores multi-hilo: pueden servir varias peticiones simultáneamente creando un nuevo hilo (thread) para atender cada nueva petición que reciben.
 - Precisan facilidades de procesos concurrentes: “threads”
 - Adecuado para recursos instanciables

Servidores multi-hilo



Servidores concurrentes



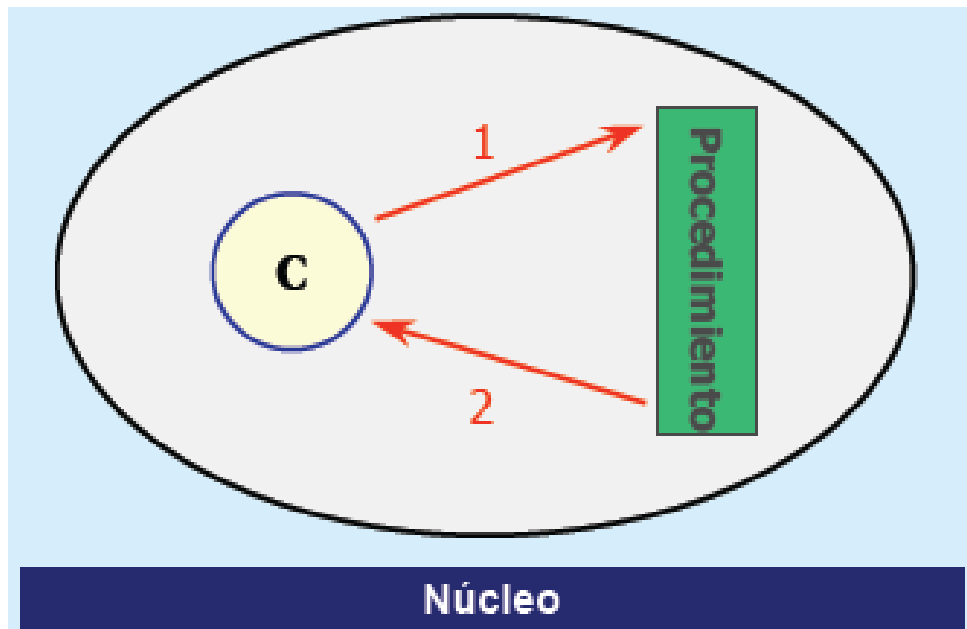
Mensajes

- Sistemas de mensajes
 - Para realizar una aplicación cliente-servidor es necesario disponer de un sistema de mensajería entre procesos, o en otras palabras, un nivel de transporte.
 - Existen dos modelos básicos de servicios de mensajería:
 - Comunicación orientada a conexión.
 - Comunicación sin conexión o datagrama.
 - Ejemplo: sockets TCP/IP, UDP/IP
- La interfaz socket
 - La interfaz socket en Java

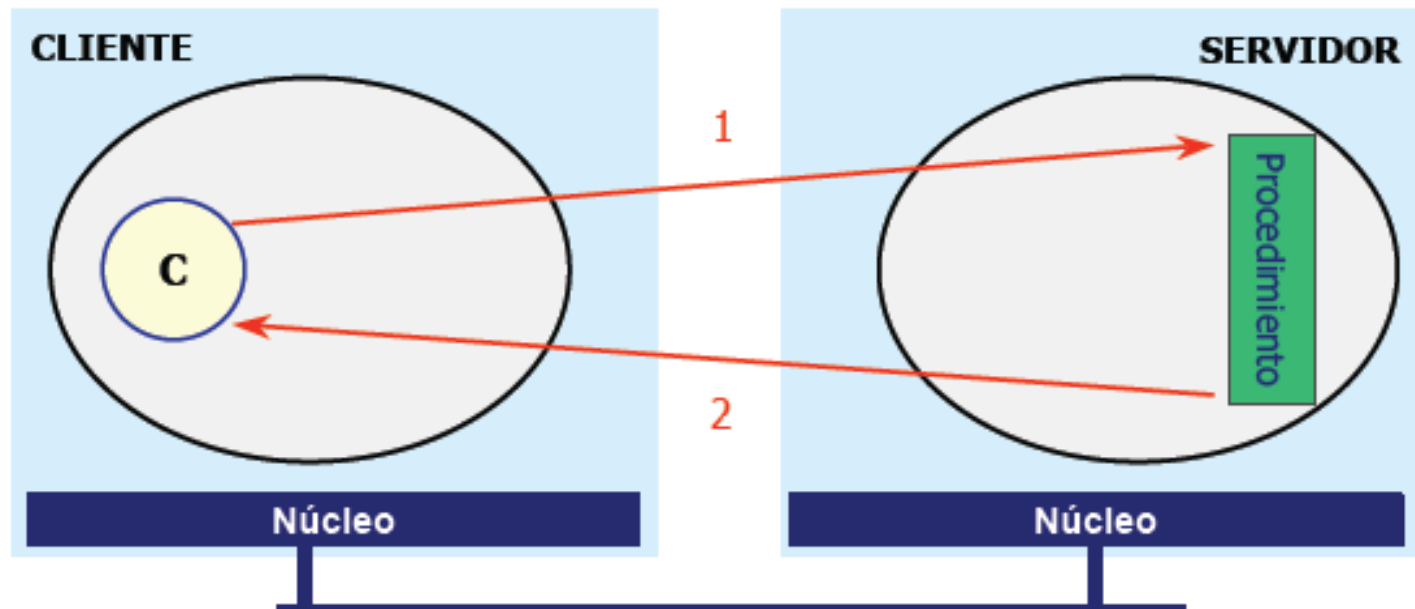
Concepto de RPC

- RPC (Remote Procedure Call): llamada a procedimiento remoto.
 - Es una forma de comunicación en sistemas distribuidos basada en extender el uso de llamadas a procedimiento para invocar servicios sobre objetos o recursos remotos en lugar de mensajes.
- Características básicas:
 - Oculta el sistema de mensajes: hace transparente su uso
 - Hace transparente la invocación de servicios locales y remotos

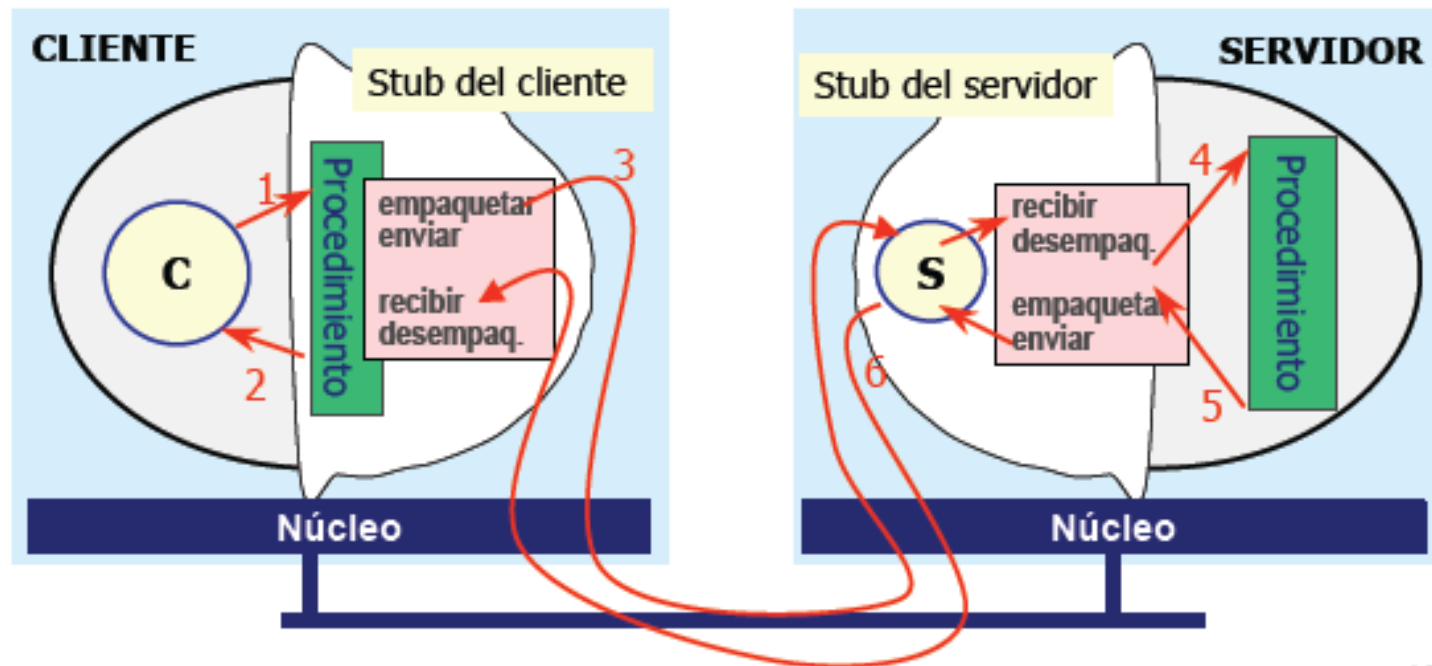
Concepto de RPC



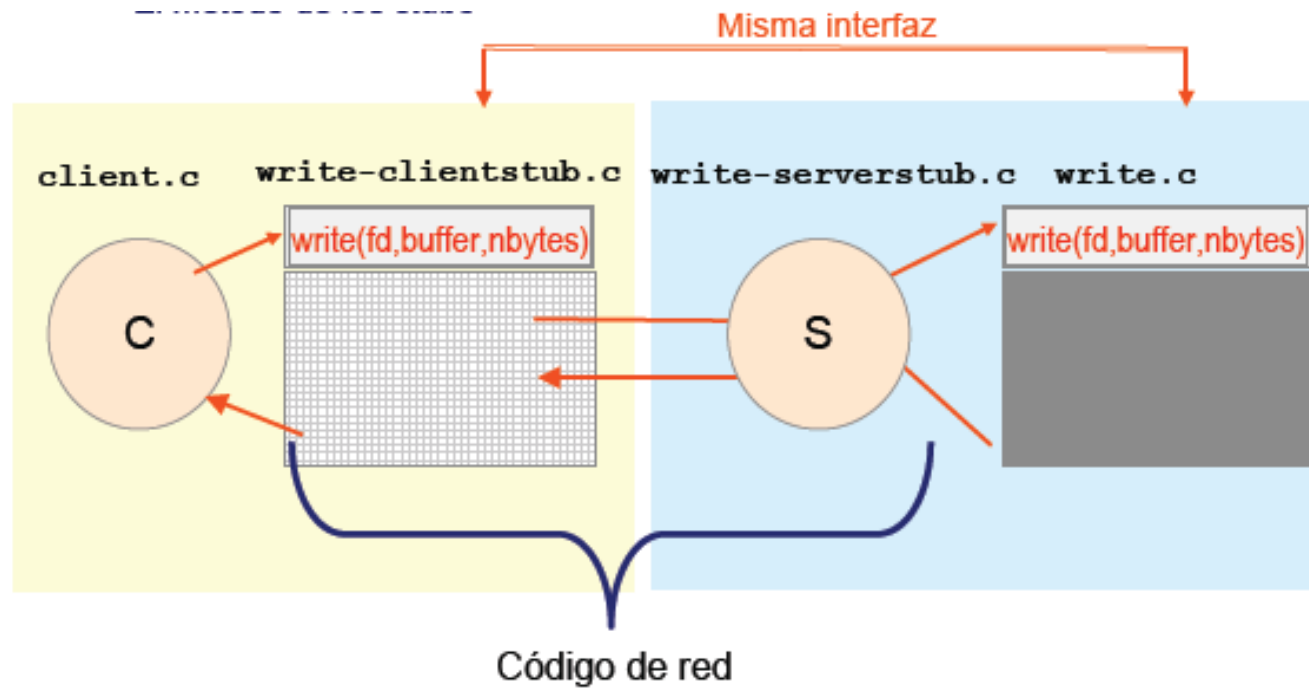
RPC



RPC



RPC

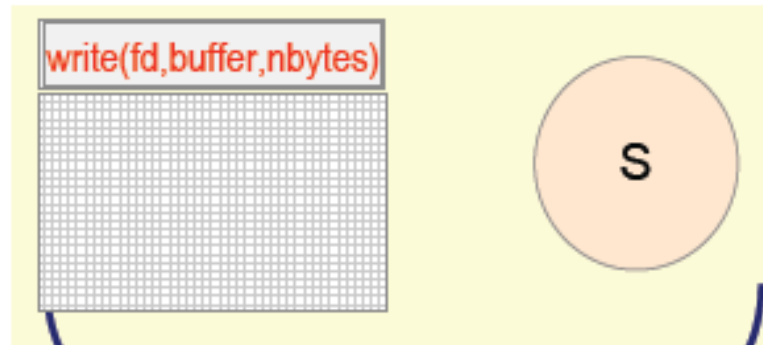


RPC

`write.h`

`write(fd,buffer,nbytes)`

`write-clientstub.c` `write-serverstub.c`



Código de red

RPC: stubs

- Concepto de stub: función de biblioteca que oculta el sistema de mensajes y proporciona a la comunicación remota una semántica de llamada a procedimiento.
- El stub del cliente: es un representante del procedimiento remoto en el nodo del cliente que tiene su misma interfaz (nombre, argumentos, semántica) pero que no ejecuta el procedimiento (no realiza la funcionalidad del procedimiento).
 - La funcionalidad del stub del cliente es la siguiente:
 - Empaquetar los parámetros en un mensaje (parameter marshalling)
 - Enviar el mensaje al servidor
 - Esperar la respuesta
 - Desempaquetar la respuesta

RPC: stubs

- El stub del servidor: es el soporte de ejecución de un procedimiento o conjunto de procedimientos remotos. Normalmente es proceso servidor en vez de una biblioteca (como en el caso del cliente).
- La funcionalidad del stub del servidor es similar a la del cliente pero, adicionalmente, se encarga de:
 - Creación de hilos: en caso de servidor con múltiples hilos, crear un hilo para gestionar cada invocación.
 - Dispatching: seleccionar el procedimiento remoto solicitado (si hay varios) e invocarlo, actuando como hilo representante del cliente.

RPC: stubs

Existen dos enfoques básicos para la generación de stubs:

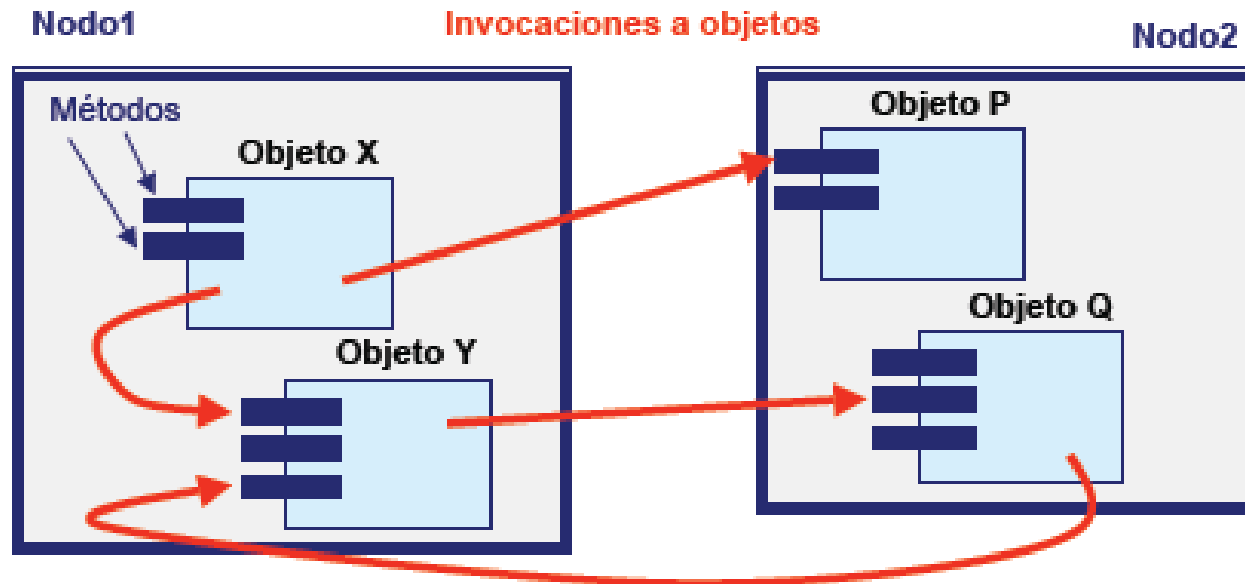
- Generación automática por un preprocesador de RPC
 - Ejemplo: rpcgen de Sun
- Generación “a mano” por el propio programador
 - Constituye un “buen estilo” de programación separar el código independiente de red y el código dependiente de la red en procedimientos separados; éste último código lo constituyen los stubs y skeletons.

Objetos Distribuido

Se basa en extender el paradigma de objetos a sistemas distribuidos:

- Objetos como forma de encapsular los recursos
- Objetos como unidad de distribución
 - La particion de una aplicación y su distribución se hace en base a objetos
- Comunicación remota basada en invocaciones a métodos de objetos.
 - Transparencia de invocación.

Objetos Distribuido



Objetos Distribuido

Aspectos clave del modelo de objetos distribuido

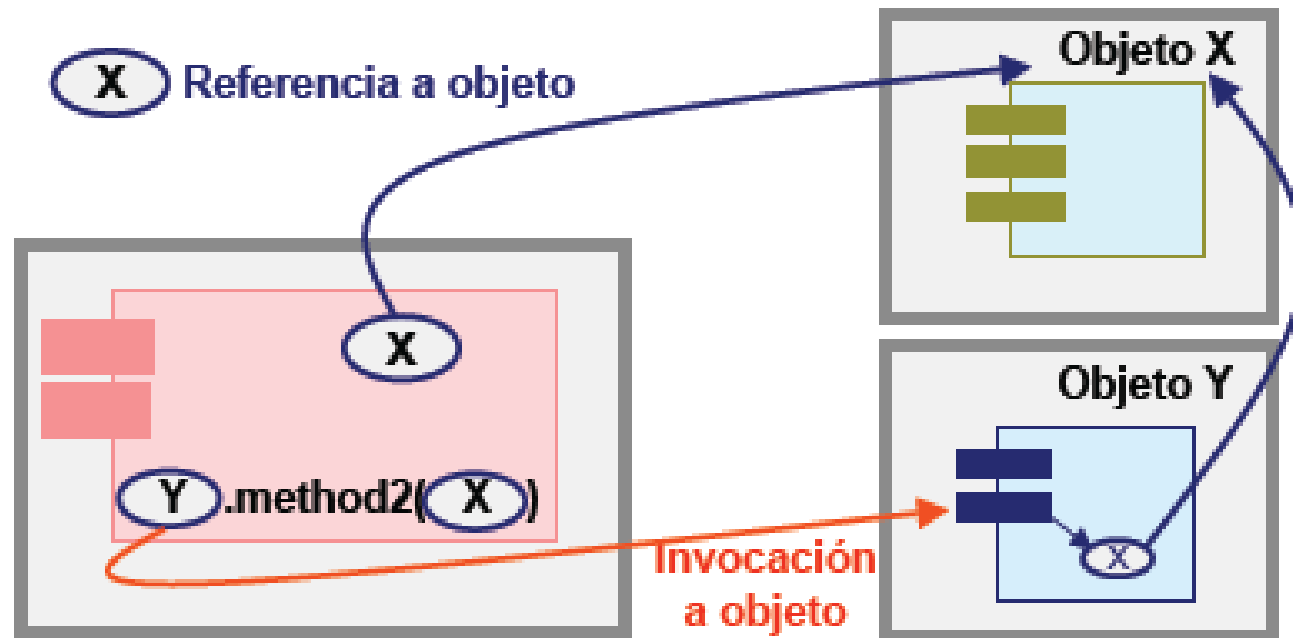
- Referencias a objetos remotos
 - Extender el concepto de referencia a objeto a objetos remotos
 - Servicio de nombres
- Invocación de métodos remotos
 - Extender el concepto de RPC a objetos
 - Implementar paso por valor de objetos (“migrar objetos”)
 - Invocaciones estáticas e invocaciones dinámicas
- Definición y procesamiento de interfaces
 - Definición de interfaces en sistemas heterogéneos
 - Generación de stubs

Objetos Distribuido

Referencias a objetos

- Concepto de referencia a objeto:
 - Determina de forma unívoca la ubicación de un objeto en un sistema distribuido.
 - Se usa para referenciar un “objeto distribuido”.
 - Extiende el concepto de puntero para que también se puedan referenciar objetos remotos.
- Implementación de referencias a objetos.
 - Las referencias se manejan de forma abstracta, siendo oculta su implementación.
 - La implementación es dependiente de la plataforma, pero algunas plataformas como CORBA la estandarizan en aras de la interoperabilidad.
 - La implementación de una referencia normalmente contiene la siguiente información:
 - Dirección IP de máquina
 - Número de port dedicado al sistema de objetos
 - Número de objeto dentro de los objetos de un nod

Referencias a objetos

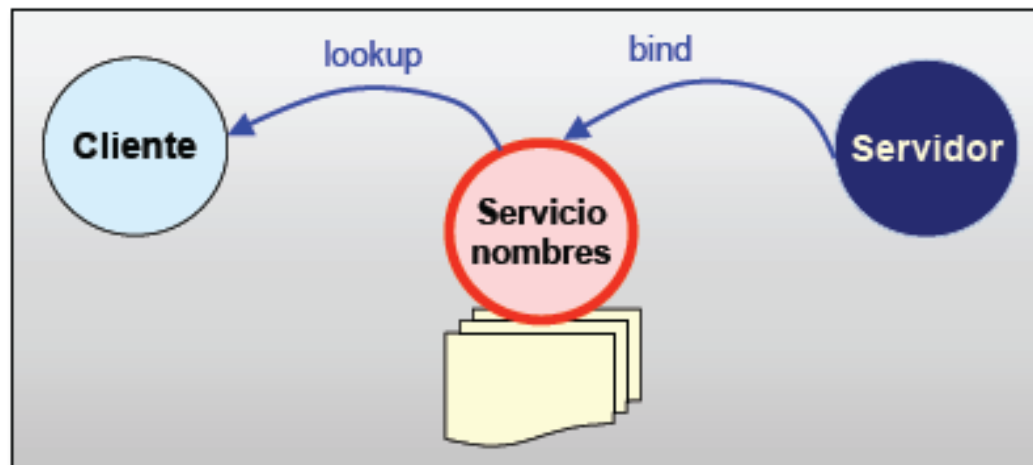


Referencias a objetos

- El problema de binding
 - Problema: ¿Como obtener una referencia a un objeto?
 - Posibles enfoques:
 - Binding estático: se resuelve en tiempo de compilación. La ubicación del objeto es conocida y fija y la referencia es generada por el propio compilador.
 - Binding dinámico (late binding): se resuelve en tiempo de ejecución. Requiere la utilización de un catálogo de servicios que proporcione esta referencia en tiempo de ejecución a partir del nombre simbólico o algún otro atributo del servicio.
- Catálogos de servicios
 - – Existen, fundamentalmente, dos tipos de catálogos:
 - Naming service: servicio de páginas blancas. Proporciona la referencia en función del nombre del objeto.
 - Trading service: servicio de páginas amarillas. Proporciona la referencia en función del servicio que proporciona el objeto.

Servicio de nombres

- Proporciona una base de datos con tuplas:
(nombreObjeto, ReferenciaAObjeto)
- Se implementa como un objeto del sistema con dos métodos fundamentales:
 - Registrar una nueva tupla: `bind(nombreObjeto, ReferenciaAObjeto)`
 - Buscar una referencia: `ReferenciaAObjeto = lookup(nombreObjeto)`

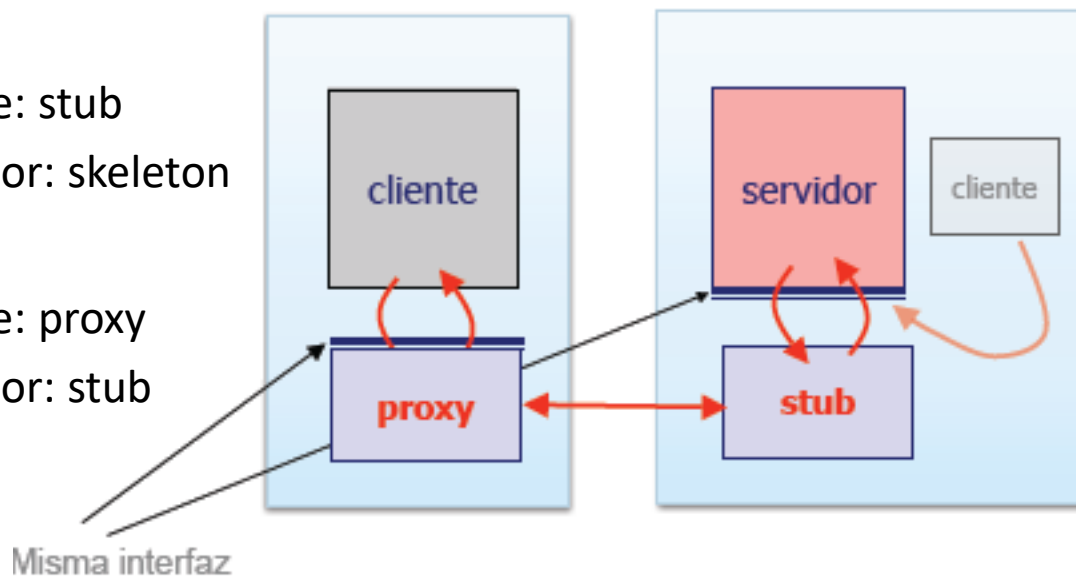


Nombres de objetos

- Existen diferentes opciones para el espacio de nombres de objetos:
 - Nombres no transparentes a la ubicación:
Ejemplo: “unaMaquina:unNombre deObjeto”
 - Nombres transparentes a la ubicación:
 - Nombres planos:
Ejemplo: “object1”, “printer3”, ...
 - Espacio de nombres con estructura de árbol:
Ejemplo: “/europe/spain/upv/anObject”

Invocación de métodos remoto

- Basado en extender el concepto de RPC para invocar métodos remotos (en vez de procedimientos)
 - Método de los stubs.
- Terminología
 - Java-RMI:
 - Stub del cliente: stub
 - Stub del servidor: skeleton
 - CORBA:
 - Stub del cliente: proxy
 - Stub del servidor: stub



Invocación de métodos remoto

- Paso de objetos por valor en invocaciones de métodos
 - Existen dos posibles esquemas:
 - Paso por referencia: se pasa una referencia (puntero) al objeto.
 - Paso por valor: implica
 - Serializar el objeto: convertir el objeto que se pasa como argumento en una cadena de bytes.
 - Migrar el objeto serializado al objeto invocado (el objeto no se “mueve”, se envía una copia serializada)
 - Reconstruir el objeto en el heap del nodo de destino (des-serializarlo).
- Serialización de objetos
 - La serialización de objetos es mucho más compleja cuando existe herencia, ya que ello implica serializar, recursivamente, todos los atributos que el objeto hereda de sus antecesores.

CORBA

– Invocación estática

```
//Obtener una referencia al objeto a través de un servicio de nombres
org.omg.CORBA.Object conversorRef = nc.resolve(path);
// Invocar
conversorRef.cambiar("euros","dollar", 565)
```

– Invocación dinámica

```
//Construir la lista de argumentos
NVList listaArgumentos = myORB.create_list(3);
Any arg1 = myORB.create_any();
arg1.insert_string("euro");
NamedValue nvArg = argList.add_value("monedaOrig", arg1, org.omg.CORBA.ARG_IN.value);

Any arg2 = myORB.create_any();
arg2.insert_string("dollar");
NamedValue nvArg = argList.add_value("monedaDest", arg2, org.omg.CORBA.ARG_IN.value);
// etc...
// Crear un objeto Any para albergar el resultado y empaquetarlo a en un NamedValue
Any resultado = myORB.create_any();
result.insert_string("dummy");
NamedValue resultVal = myORB.create_named_value("resultado", resultado,
                                                org.omg.CORBA.ARG_OUT.value);

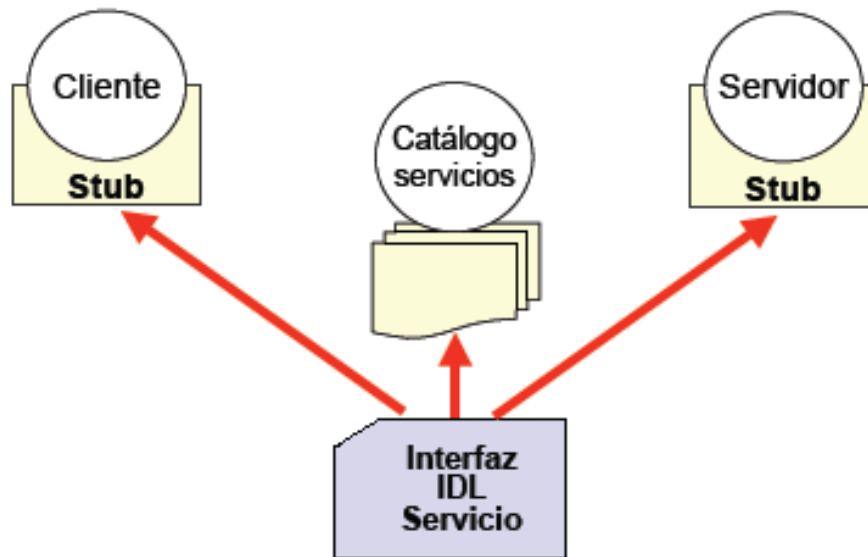
// Invocar
Request thisReq = conversorRef._create_request(ctx,"cambiar", argList, resultVal);
thisReq.invoke();
```

Lenguajes de definición de interfaces

- Lenguajes de definición de interfaces
 - Son lenguajes declarativos que permiten especificar el interfaz de un objeto de una manera estándar e independiente de su lenguaje de implementación.
 - Establecen la naturaleza de las posibles interacciones entre objetos (atributos, métodos, excepciones, ...)
 - Estandariza los tipos de datos (enteros, reales, ...) y su implementación.
- IDL (Interface Definition Language)
 - Lenguaje de especificación de interfaces muy próximo a C++ pero sin punteros
 - Ver transparencias IDL
 - Definido originalmente en el proyecto DCE y estandarizado por CORBA
 - Existen bindings definidos para C, C++, Ada, Smalltalk, ...

Procesamiento de interfaces

- Puede generar de forma automática:
 - Stubs de cliente y servidor
 - Entradas para catálogos de servicios
 - Interfaz en otro lenguaje de programación



Fuente: Joan Vila