URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC
Programming Robot Intelligence

Sistemas Distribuidos y Concurrentes

Comunicación entre procesos distribuidos

Grado en Ingeniería de Robótica Software

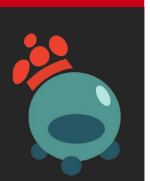
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

Comunicación entre procesos

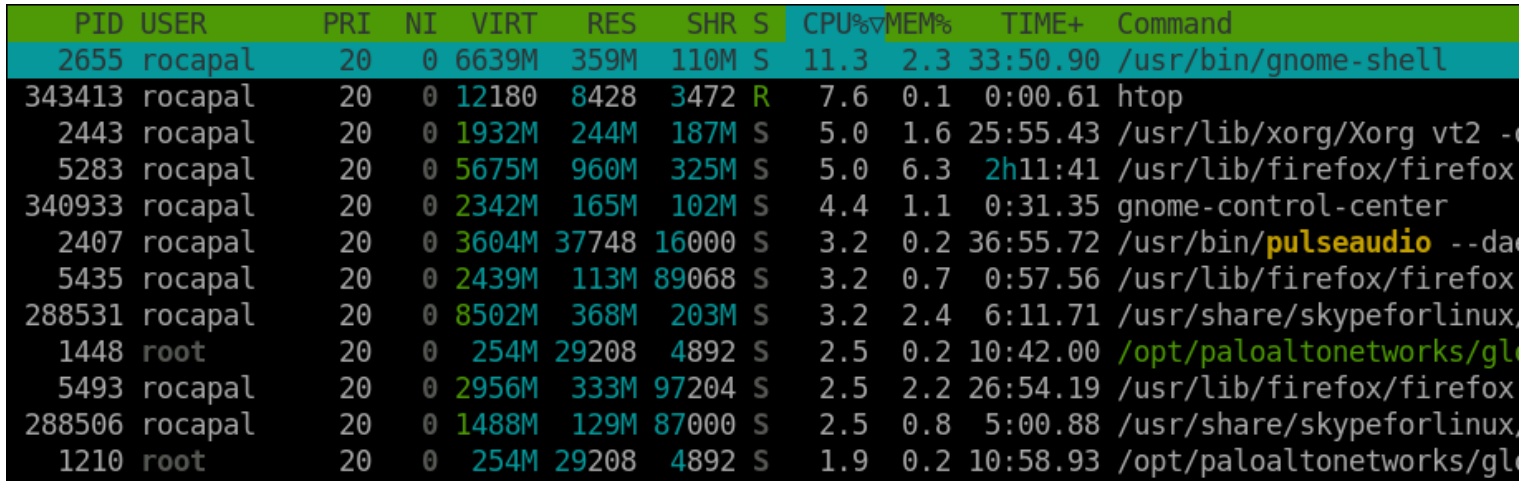
- **Procesos** ejecutando dentro de la misma máquina pueden interactuar entre ellos mediante:
  - Espacios de memoria compartidos
    - Variables, buffers, ficheros, etc.
  - Paso de mensajes
    - Pipes, Colas (problemas de concurrencia)

- En un sistema distribuido la comunicación debe ser **distribuida entre los nodos**, lo que complica la interacción entre los procesos distribuidos.



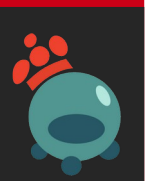
Process / Thread / Fork

- Un proceso/tarea es una abstracción del sistema operativo para ejecutar un conjunto de código
- Es un instancia de un programa en ejecución. En linux, un proceso siempre tendrá un ID (PID)
- Cada proceso tiene la ‘ilusión’ de ser el único ejecutando en el sistema.
- La comunicación entre procesos solo ocurre utilizando los mecanismos de comunicación inter-procesos.



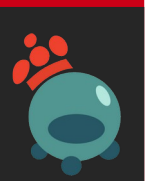
Process / Thread / Fork

- Un thread/hilo/subproceso es un subflujo de ejecución que puede ser manejado de manera independiente por un planificador/scheduler
- Un thread normalmente es parte de un proceso
- Los threads (de un mismo proceso) comparten espacio de direcciones y por tanto memoria.
- Por normal general, los cambios de contexto entre threads es más liviano y rápido que entre procesos.



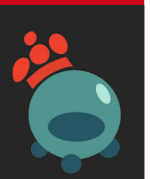
Process / Thread / Fork

- En sistemas unix , fork() genera un proceso hijo (child) con una copia exacta del padre.
- A partir de ahí, cada uno ejecuta independientemente.
- Padre e hijo ejecutan en un espacio diferente de memoria.
- fork() suele usarse más para procesos pesados computacionalmente.



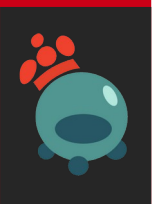
Process / Thread / Fork

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

    pid_t pid = fork();

    if ( pid > 0 ) {
        printf("This is parent section [Process id: %d].\n", getpid());
    }
    else if (pid == 0) {
    /*child process*/
        printf("fork created [Process id: %d].\n", getpid());
        printf("fork parent process id: %d.\n", getppid());
    }
    else {
        printf("Error while executing fork()");
    return -1;
    }

    printf("hello world\n");

    return 0;
}

```



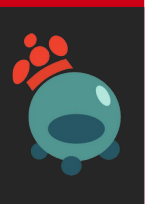
Arquitecturas de comunicación

- Arquitecturas más frecuentes de propósito general:
  - **Cliente / Servidor:**
    - 2 roles en la interacción
    - Servidor es un cuello de botella
    - Comunicación síncrona
  - **Editor / Subscriptor**
    - Manejo de eventos o “temas”
    - Eventos generados por editores y recibidos por subscriptores.
    - Comunicación asíncrona
  - **Peer-to-peer**
    - Arquitectura descentralizada
    - Redes sin topología definida.

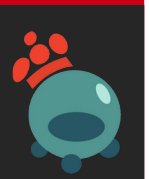


Comunicación Síncrona / Asincrona

- **Comunicación Síncrona**
  - Se realiza una comunicación y se realiza una espera activa hasta obtener el resulto esperado.
  - *Ej: fork()*
- **Comunicación Asíncrona**
  - Se realiza una comunicación y se atienden otras tareas hasta que la respuesta está lista.
  - Ej: suscripción para recibir evento de *“low battery”*

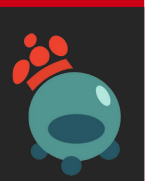


Comunicación entre procesos distribuidos



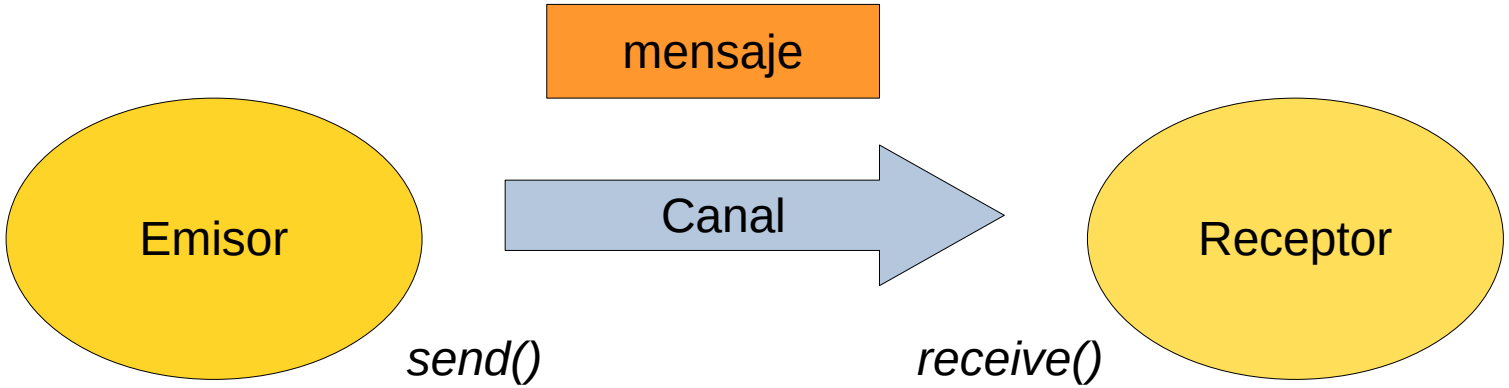
Paso de mensajes

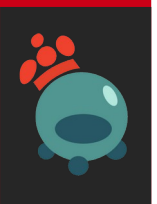
- El **paso de mensajes** es un paradigma de programación que se usa para, desde un proceso, invocar de forma abstracta un comportamiento concreto por parte de otro actor
- Es una capa sobre protocolo de transporte
- Los procesos por los general no comparten memoria física
- Alto nivel de **encapsulamiento** y distribución
- Mantiene la **coherencia** y **sincronización** entre los procesos distribuidos.
- La comunicación se realiza mediante **primitivas** de comunicación: *send()* y *receive()*



Paso de mensajes

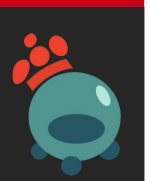
- Modelo general
- Valido para cualquier arquitectura hardware





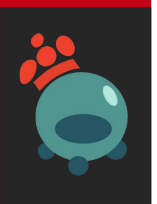
Paso de mensajes: Diseño

- **Sincronización** entre emisor y receptor
  - Comunicación síncrona/asíncrona
- **Identificación** en el proceso de comunicación
  - Comunicación directa/indirecta
  - Comunicación simétrica/asimétrica
- Características del canal
  - Capacidad, uni/bidireccional, etc.
- Gestión de **fallos**



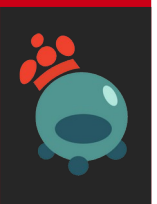
Paso de mensajes: Comunicación

- Comunicación directa
  - Cada proceso que desea comunicarse debe nombrar explícitamente el destinatario o el remitente de la comunicación
    - ***send (P, mensaje)***
      - Enviar un mensaje al proceso P
    - ***receive (Q, mensaje)***
      - Recibir un mensaje del proceso Q
  - Los dos procesos se tienen que *conocer* previamente.
  - Solo existe un único enlace de comunicación.



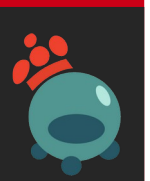
Paso de mensajes: Comunicación

- Comunicación indirecta
  - Con la comunicación indirecta, los mensajes se envían a, y se reciben de, buzones
    - ***send (A, mensaje)***
      - Enviar un mensaje al buzón A
    - ***receive (A, mensaje)***
      - Recibir un mensaje del buzón A
  - Permite esquemas uno a uno, uno a muchos, muchos a uno y muchos a muchos.
  - 2 o más procesos pueden utilizar el buzón.
  - El tipo de esquema “muchos a uno” se conoce como “puertos”



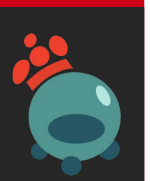
Paso de mensajes: Comunicación

- Comunicación simétrica
  - Los procesos tanto receptor como emisor necesitan nombrar al otro para comunicarse
    - ***send (P, mensaje)***
    - ***receive (Q, mensaje)***
- Comunicación asimétrica
  - Sólo el emisor nombra al destinatario
  - Resuelve el problema en aplicaciones cliente/servidor
    - ***send (P, mensaje)***
    - ***receive (id, mensaje)***



Paso de mensajes: Comunicación

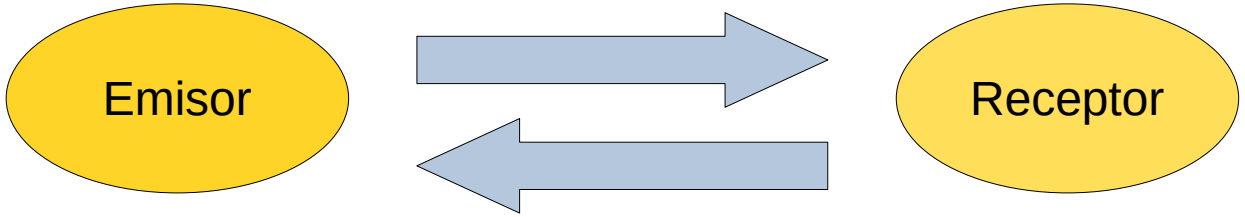
- Llamadas pueden ser
  - **Bloqueantes**
    - La comunicación entre emisor y receptor se bloquea hasta que el paso de mensaje se ha realizado.
    - Son utilizadas para comunicación síncrona
  - **No bloqueantes**
    - La comunicación entre emisor y receptor no bloquea al componente que la solicita.
    - Envio-no-bloqueante: el emisor delega el envío a un sub-componente y reanuda su ejecución
    - Recepción-no-bloqueante: Si existe un dato disponible el receptor lo lee, en caso contrario notifica que no hay datos.

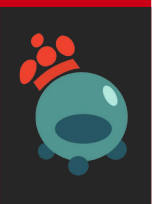


Paso de mensajes: Canal

- El canal es el medio de comunicación usado por el cual el emisor y receptor pueden interaccionar.
  - **Unidireccional**: La información siempre fluye en el mismo sentido.
    - Televisión o Radio

  - **Bidireccional**: La información fluye en ambos sentidos
    - Conversación telefónica, chat.

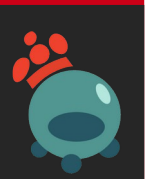




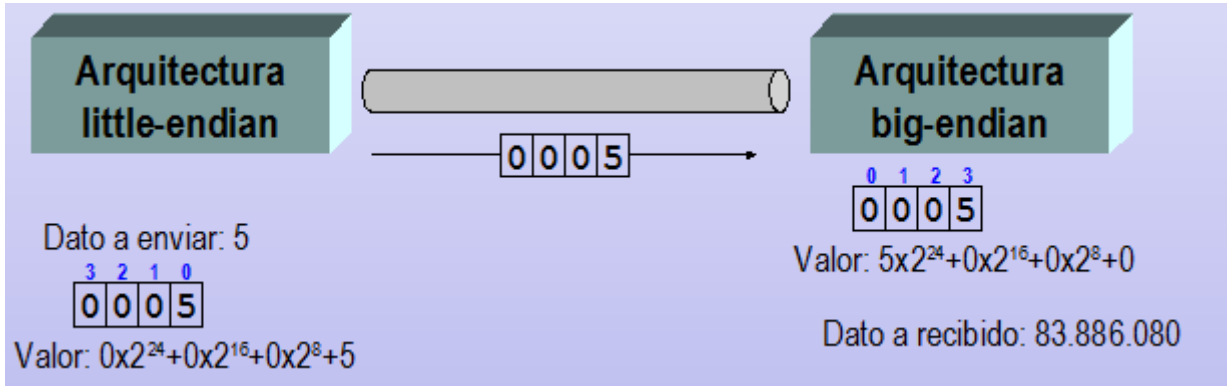
Paso de mensajes: Canal

Características del Canal:

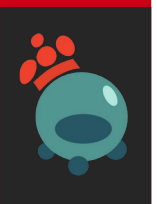
- **Flujo** de datos
  - Enlaces unidireccionales (síncrona) o bidireccionales (síncrona o asíncrona)
- **Capacidad** del canal
  - cero, limitada, infinita (teórico)
- **Tamaño** de los mensajes
  - Longitud fija o variable
- Canales con **tipo** o sin tipo
- Paso por **copia** o por **referencia**



Paso de mensajes: Mensaje

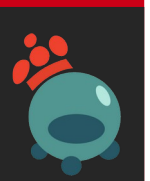


- Para la interpretación correcta del mensaje tanto emisor como receptor, ambos deben conocer la codificación correcta
  - Tamaño datos numéricos
  - Ordenación de bytes
  - Formatos de texto: ASCII, ...



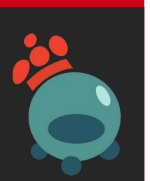
Paso de mensajes: Errores

- Mayor **probabilidad** de error en transmisiones entre procesos distribuidos que cuando ejecutan en la misma máquina.
- Mensajes **perdidos** entre los procesos que se comunican utilizando un canal (códigos de verificación)
- Ruidos en la transmisión puede provocar **alteración** en los datos del mensaje original.
- **Bloqueos** en el emisor/receptor
- La gestión de **fallos y recuperación** en sistemas distribuidos se cae en el ámbito de Sistemas Tolerantes a Fallos.



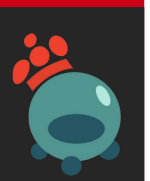
Radio

- Emisor
- Receptor
- Canal
  -
  -
  -
- Fallos
  -
  -



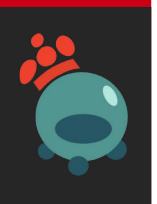
WhatsApp

- Emisor
- Receptor
- Canal
  -
  -
  -
- Fallos
  -
  -



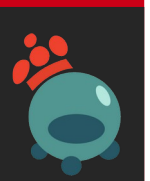
Marshaling

- Es un proceso de **transformación de la representación** de datos a un formato adecuado para su almacenamiento o transmisión.
- Ampliamente usado en comunicaciones distribuidas
  - Procesos puede correr en diferentes arquitecturas.
  - Con diferente representación de los datos.
- Transparencia de acceso
- La operación inversa se denomina *unmarshaling*



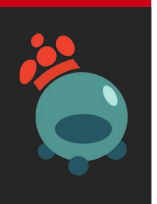
Comunicación entre procesos

- La comunicación entre procesos distribuidos puede realizarse mediante: Sockets, RPC y RMI
- **Sockets:**
  - Permite la comunicación entre procesos distribuidos a bajo nivel.
  - Es necesario conocer los detalles de la red y comunicaciones.
  - Completo control de las comunicaciones
  - Dificil extender el sistema si se quieren suportar nuevas arquitecturas.

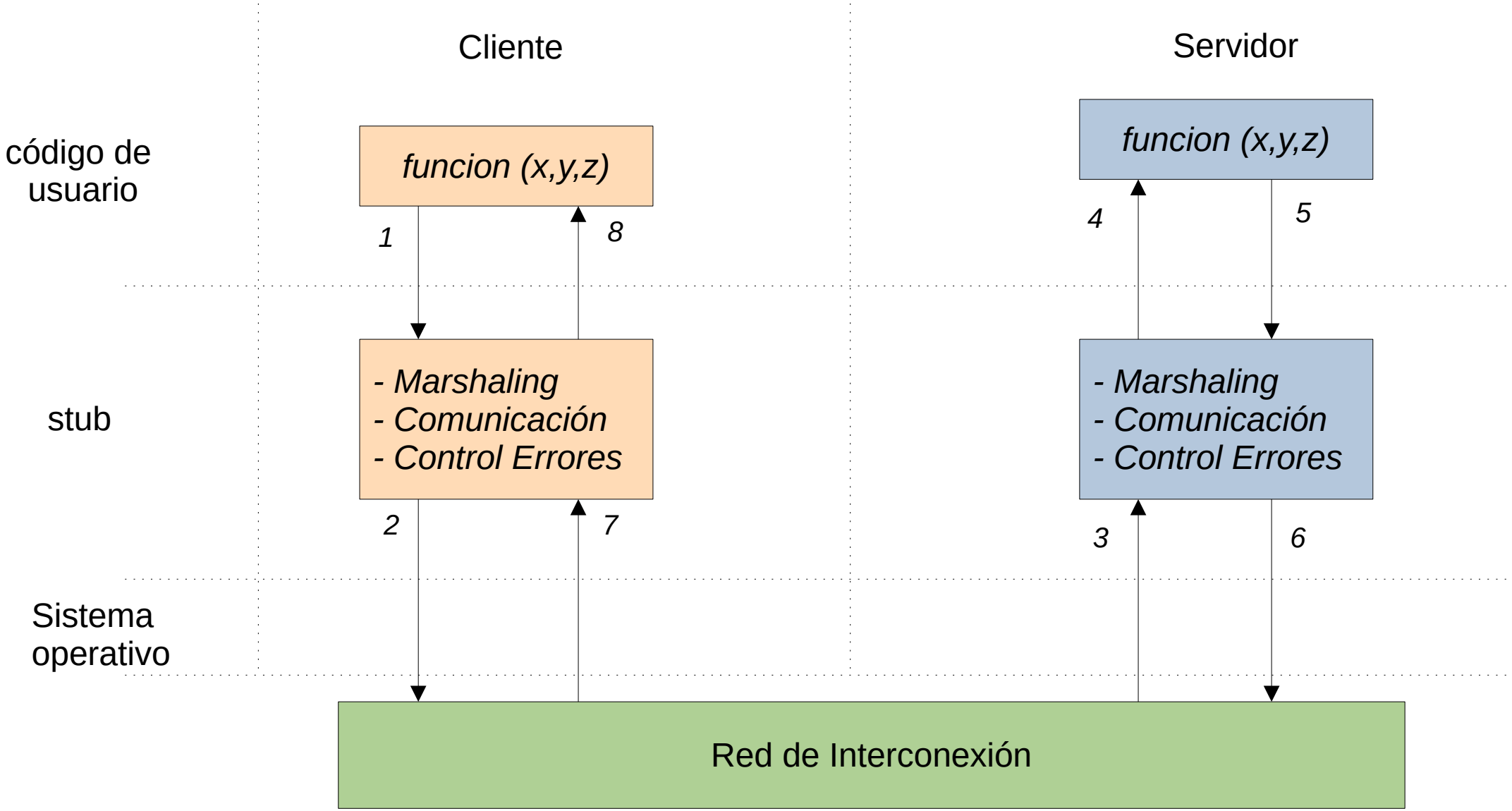


Comunicación entre procesos: RPC

- **RPC** (Remote Procedure Call).
  - Permite realizar llamadas a funciones localizadas en otra máquina.
  - Se encapsula toda la programación de red y comunicaciones.
  - Ofrece transparencia de acceso.
  - La complejidad de la comunicación se encapsula en componentes denominados **stub** (tanto para el cliente, como para el servidor)
  - Permite modular la lógica del sistema distribuido.
  - Excelente *escalabilidad*.

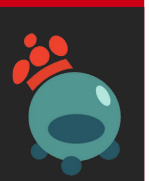


Comunicación entre procesos: RPC

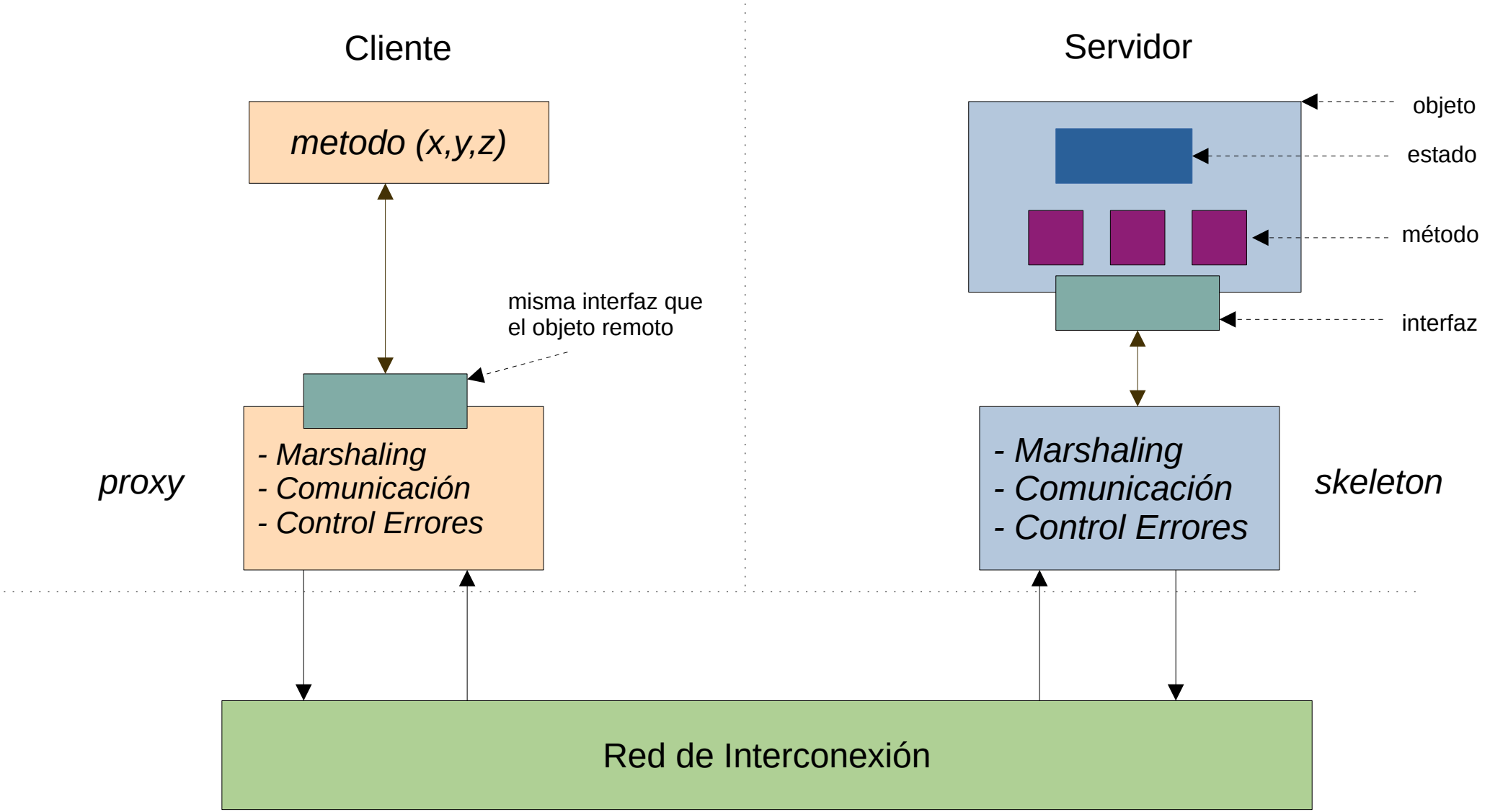


Comunicación entre procesos (RMI)

- **RMI** (Remote Method Invocation).
  - Paradigma de orientación a objetos aplicado a comunicación distribuida.
  - Permite modificar el estado de objetos localizados remotamente.
  - Separación clara entre interfaces de comunicación y objetos.
  - El *stub* del cliente se denomina **proxy**
  - El *stub* del servidor se denomina **skeleton**
  - Ofrece también modularización y escalabilidad.



Comunicación entre procesos (RMI)



URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC

Programming Robot Intelligence