Sistemas Informáticos Distribuidos

Tema 2
Comunicación y Sincronización de Procesos

Contenidos

1. Paso de mensajes
2. Comunicación Cliente/Servidor
3. Llamada remota a procedimiento (RPC)
4. Invocaciones remotas o citas (“*rendez-vous*”)

Paso de mensajes

- Conceptos:
  - **Formar** ("*marshalling*"): pone una colección de datos en un formato adecuado para transmitirlos en un mensaje. Consiste en:
    - **Poner en plano** ("*to flat*") las estructuras de datos en secuencias básicas
    - Traducción de los elementos básicos a una **representación de datos estándar** (Ej.: "*eXternal Data Representation*" o XDR de SUN)
  - Las operaciones para formar mensajes se pueden generar **automáticamente**
  - **Canal:** abstracción de una red de comunicación física que proporciona un camino de **comunicación** entre procesos y **sincronización** mediante 2 primitivas: *send* y *receive*

Paso de mensajes

- Las notaciones difieren en:
  - **Ámbito y denominación** de los canales (e.g., globales a procesos o asociados a subconjunto de ellos)
  - **Uso** (e.g., flujos de información bidireccionales o no)
  - Cómo se **sincroniza** la comunicación (e.g., síncrona-bloqueante)
- Todas las propuestas son **equivalentes**, ya que un programa en una notación se puede escribir en otra, pero cada propuesta es más adecuada dependiendo del tipo de problema

Paso de mensajes

- Notación aceptada, aunque existen variantes (“*timeouts*”, etc.):
  - *send <puerto>(<mensaje>)*
  - *receive <puerto>(<mensaje>)*
  - *empty <puerto>*
- Tipo de comunicación:
  - **Síncrona**: ambas primitivas bloqueantes
  - **Asíncrona**: sólo es bloqueante *receive*
  - **Buffer finito**: *receive* bloqueante y *send* cuando el buffer está lleno

Paso de mensajes

- Destino de los mensajes:
  - Debe ser **conocido** por el emisor e independiente de la localización (transparencia)
  - **Tipos de destinos**:
    - **Proceso**: comunicación punto a punto
    - **Enlace** (“*link*”): punto a punto con indirección
    - **Puerto** (“*port*”): muchos a uno con indirección
    - **Buzón** (“*mailbox*”): muchos a muchos con indirección
    - **Difusión** (“*broadcast*”): muchos a muchos con indirección
    - **Selección** (“*multicast*”): muchos a muchos con indirección

Paso de mensajes

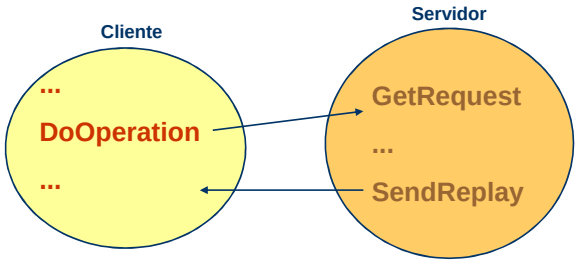
- **Protocolos de comunicación de grupos**:
  - Uso:
    - **Tolerancia a fallos** en servicios replicados
    - **Localización** de objetos en servicios distribuidos
    - **Mejor rendimiento** con servicios replicados
    - **Actualización múltiple** notificando eventos
  - Propiedades:
    - **Atomicidad**: el mensaje es recibido por todos o por ninguno
    - **Ordenación**: ejecución de operaciones en el mismo orden
- **Fiabilidad**: paso de mensajes fiable se puede construir partiendo de uno no fiable

Comunicación Cliente/Servidor

- **Protocolo petición/respuesta**
  - Comunicación típicamente **síncrona y segura/fiable** (la respuesta del servidor sirve como acuse de recibo)
  - Alternativas de implementación:
    - **Primitivas de comunicación** (*send* y *receive*).
      Inconvenientes:
      - Sobrecarga por más canales utilizados explícitamente
      - Correspondencia entre *send* y *receive*
      - Garantía de reparto de mensajes si los servicios de red no la proporcionan
      - Ejemplo

Comunicación Cliente/Servidor

- **Protocolo petición/respuesta**
  - Alternativas de implementación:
    - **Operaciones de comunicación**:
      - Combinan aspectos de monitores y paso de mensajes
      - Tres primitivas: DoOperation, GetRequest y SendReplay



Comunicación Cliente/Servidor

- **Implementación del protocolo (gestión de fallos)**
  - **Plazo de tiempo** ("*time-out*"). Alternativas cuando se cumple:
    - Devolver fallo
    - Repetir petición y, eventualmente, devolver fallo si es el caso
  - **Filtrar mensajes** de petición duplicados. Formato mensaje:
    - Tipo: petición o respuesta
    - Identificador petición: proceso y número petición
    - Identificador procedimiento
    - Argumentos

tipoMensaje – int (0=petición, 1=Respuesta)
idPetición
refObjeto/procedimiento
idMétodo
argumentos

Comunicación Cliente/Servidor

- **Implementación del protocolo**
  - **Mensajes de respuesta perdidos**
    - **Operaciones idempotentes** en servidores permiten reejecución proporcionando los mismos resultados
  - **Historia**
    - Retransmisión de respuestas sin reejecución de operaciones
    - Manejo eficiente de la estructura de historia:
      - Nueva petición como reconocimiento de la respuesta previa
      - Reconocimientos del cliente (*acknoledgments*) ayudan a descartar entradas en la historia
      - También pueden descartarse transcurrido un periodo de tiempo

Comunicación Cliente/Servidor

- Modelo de fallos protocolo petición/respuesta: garantías de envío/informar sobre errores

Garantías de envío			*Semántica RPC*
Reintentar petición	*Filtrar duplicados*	*Reejecutar procedimiento/ Retransmitir repuesta*	
No	No aplicable	No aplicable	**Quizás**
Sí	No	Reejecutar procedimiento	**Al menos una vez**
Sí	Sí	Retransmitir Respuesta	**Exactamente una vez**

RPC

- Introducción:
  - En el modelo C/S los servicios **proporcionan varias operaciones**
  - La comunicación C/S se basa en el protocolo **petición/respuesta**
  - Los mecanismos de RPC integran esta organización con lenguajes de programación procedurales convencionales
  - Se modela y diseña como una llamada a procedimiento local, pero esta se ejecuta remotamente
  - El servidor es visto como un módulo con **una interfaz que exporta operaciones** y con tiempo de vida distinto
  - **Biblioteca soportando servicios para aislar cuestiones como**: diferencias entre procedimientos locales y remotos, localización servidor, mejora de rendimiento por medio de cachés

RPC

- Semántica:
  - Parámetros de entrada/salida (comunicación bidireccional)
  - Sólo uso de variables locales
  - No tienen sentido punteros, por tanto, se hacen **clonaciones** (estructura interna oculta por modularidad)
  - Servidores pueden devolver **referencias opacas** que no pueden ser interpretadas en el entorno del cliente

RPC

- Cuestiones de diseño:
  - **Clases de sistemas**: mecanismo integrado en el lenguaje de programación (permite que algunos requisitos, p. ej. excepciones puedan tratarse con construcciones del lenguaje) **o** de propósito general (no dependen de un entorno particular)
  - **Características** del lenguaje de definición de interfaces (**IDL**): nombres procedimientos, tipo de parámetros y dirección
  - **Manejo de excepciones**: Notificar errores debidos a distribución, plazos de tiempo, errores de ejecución del procedimiento

RPC

- **Protocolo petición/respuesta**
  - *(Normalmente)* Tres protocolos diferentes para informar sobre errores en RPC
  - Diferentes semánticas en presencia de fallos

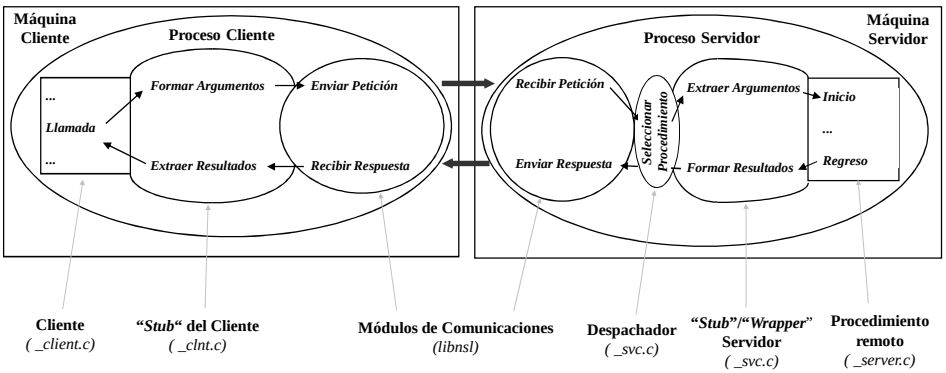
Nombre	***Mensaje enviado por***		
	Cliente	***Servidor***	***Cliente***
R	Petición		
RR	Petición	Respuesta	
RRA	Petición	Respuesta	Reconocimiento

RPC

- Cuestiones de diseño:
  - **Transparencia:**
    - **Manejar errores** debido a que RPC:
      - **Más vulnerable** (red, otra computadora, otro proceso)
      - Toma **más tiempo** que una local
    - Por tanto, **no debería ser transparente,** sino explícita al programador
    - Aunque debería **ocultar detalles de bajo nivel** de paso de mensajes, pero no retardos o fallos

RPC

- Implementación:



RPC

1. Procesamiento Interfaz

- **Integra** el mecanismo RPC con programas cliente y servidor formando y extrayendo argumentos y resultados
- Se **compila una especificación** escrita en un lenguaje de definición de interfaces (IDL) y genera cabeceras, plantillas y *stubs*:
  - En el cliente convierte llamada local a remota
  - En el servidor selecciona y llama al procedimiento adecuado

2. Módulo comunicaciones

- Implementa protocolo petición-respuesta

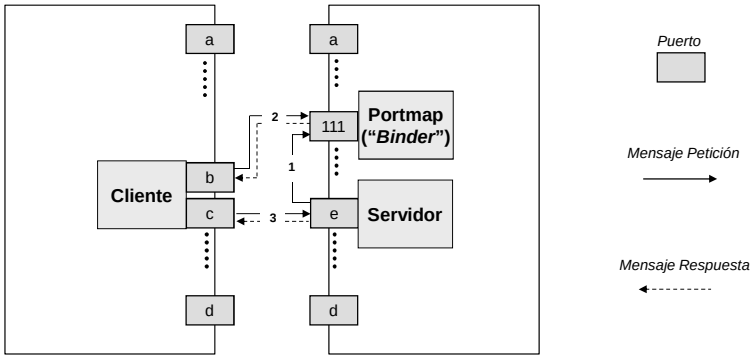
RPC

3. Servicio de ligadura ("*binding*")

- Mecanismo para localización del servidor
- **Asociación** de un nombre a un identificador de comunicación
- Mensaje de petición se dirige a un puerto concreto
- Se evalúa cada vez que el cliente lo requiera, ya que el servidor puede ser **relocalizado**
- Servicio del cual dependen otros, por tanto, debe ser **tolerante a fallos**
- Los servidores exportan (**registran**) sus servicios y los clientes los importan (**buscan**)

RPC

3. Servicio de ligadura ("*binding*")



Operaciones

- **Registro** (<*nombre_servicio*>,<*puerto_servicio*>,<*versión*>);
- **Retirar** (<*nombre_servicio*>,<*puerto_servicio*>,<*versión*>);
- **Buscar** (<*nombre_servicio*>,<*versión*>) : <*puerto*>;

RPC

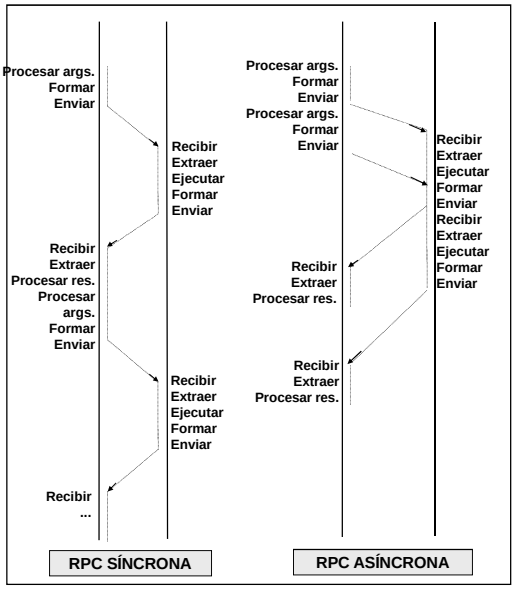
3. Servicio de ligadura ("*binding*")

- Alternativas para localizar el ligador:
  1. **Dirección conocida**: El cliente/servidor han de ser recompilados cuando el ligador ser relocaliza
  2. El **sistema operativo** proporciona la información en tiempo de ejecución (e.g. variables de entorno)
  3. Cuando cliente/servidor se lanzan, envían **mensajes de difusión** para que así el ligador responda con la dirección

RPC Asíncrona

- Requisitos comunes:
  - El cliente envía muchas peticiones al servidor
  - No se necesita una respuesta a cada petición
- Ventajas:
  - El servidor puede **planificar operaciones** más eficientemente
  - El cliente **trabaja en paralelo**
  - Se facilita el **cálculo de peticiones paralelas** en el caso de varios servidores

RPC Asíncrona



RPC Asíncrona

- Optimizaciones:
  - Varias peticiones en una sola comunicación:
    Se almacenan mensajes hasta que:
    a) Se cumple un **plazo de tiempo**
    b) Se realice una petición que **requiere respuesta**
  - El cliente puede proceder si no espera una respuesta que puede obtener más tarde

Citas

- A veces nombradas como **citas extendidas**
- Las invocaciones remotas se sirven mediante una **instrucción de aceptación**
- **RPC** es una **comunicación intermódulo**
- Las instrucciones de comunicación están **limitadas**:
  - *A menudo un proceso desea comunicarse con más de un proceso, quizás en puertos diferentes, y no se sabe el orden en el cual los otros procesos desean comunicarse con él*

Citas

- **No determinismo** mediante **instrucciones guardadas**:

```
Var  a, b, c;
a, b, c := A, B, C;
do
        a > b   ì   a, b :=  b, a;
½       b > c   ì   b, c :=  c, b;
od
```


Citas

- **Comunicación no determinista** = instrucciones guardadas + instrucciones de comunicación

Citas

- **Semántica** de la guarda:
  1. Tiene **éxito** si ***B*** es verdad y la ejecución de ***C*** no produce retardo
  2. **Falla** si ***B*** es falso
  3. **Bloquea** si ***B*** es verdad, pero ***C*** no se puede ejecutar sin producir retardo
- ***B*** no puede cambiar hasta ejecutar otras instrucciones de asignación, ya que no hay **variables globales**
- La guardas pueden incluir instrucciones de comunicación de entrada o salida

Citas

- Las **instrucciones guardadas de comunicación** se combinan en construcciones:
  - **Alternativas** (***IF***):
    - Si al menos una guarda tiene éxito, una de ellas se escoge de forma no determinista ejecutando ***C*** y ***S***
    - Si todas las guardas fallan, entonces ***IF*** falla o termina
    - Si no hay guardas con éxito y algunas están bloqueadas, la ejecución se retrasa hasta que la primera tenga éxito
  - **Repetitivas** (***DO***) igual que ***IF*** con ejecución iterativa hasta que todas las guardas fallen

Citas

- **Ejemplo:** Servidor de ficheros. Hasta *n* ficheros abiertos a la vez. El acceso a cada fichero se proporciona por un proceso servidor de fichero distinto
- Operaciones sobre archivos: abrir, leer, escribir, cerrar
- Canales de comunicación:
  - `mailbox abrir (string, int);`
  - `chan respuesta_abrir [1..m](int); //tantos como clientes`
  - `chan leer [1:n](…);`
  - `chan respuesta [1..m](…);`
  - `chan escribir [1:n](…);`
  - `chan cerrar [1:n](…);`

Citas

- **Ejemplo:** Servidor de ficheros. Hasta *n* ficheros abiertos a la vez. El acceso a cada fichero se proporciona por un proceso servidor de fichero distinto
- Operaciones sobre archivos: abrir, leer, escribir, cerrar

```
Ficheros [i:1..n]::
var nombref:string; args: otros tipos;
  índice_cliente:int; resultados:int;
  mas: bool;
  buffer_local, caché, dirección_de_disco, …;

do receive abrir (…
```

Citas *(Citas extendidas)*

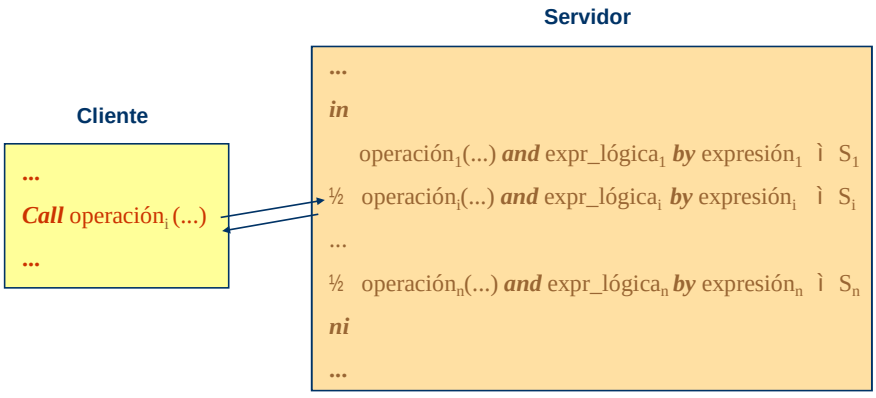
- **Órdenes guardadas de comunicación**:
  - Un proceso **exporta operaciones** de forma similar a RPC
  - Otro proceso invoca operaciones exportadas
    - **call <nombre_proceso>.<nombre_operación>(<argumentos>);**
  - El proceso servidor atiende invocaciones en su contexto de ejecución mediante **instrucciones de aceptación** (**in <nombre_operación> (<parámetros_formales>) → S ni**)
  - El **ámbito** de los parámetros formales es el de la operación guardada
  - Una guarda de una operación **tiene éxito** cuando:

    a) Se ha **invocado la operación**

    b) La expresión lógica **se evalúa a verdad**
  - La ejecución **se retrasa** hasta que una guarda tiene éxito, no determinismo cuando hay varias

Citas extendidas

- **Construcción alternativa con órdenes guardadas de comunicación:**



Citas extendidas

- **Características**:
  - Operaciones en el **contexto del proceso** que especifican puntos de comunicación de muchos a uno
  - Sin parámetros hay **sincronización** y no comunicación
  - Una misma operación puede producir **efectos diferentes**
  - Las invocaciones se sirven en los **instantes que desee el servidor**