Sistemas Distribuidos

MC Jesus Humberto Abundis Patiño

UNIVERSIDAD AUTÓNOMA DE SINALOA
Facultad de Informática Culiacán

Referencias Básicas

- **Distributed Systems: Concepts and Design**
- G. Coulouris, J. Dollimore, T. Kindberg; Addison-Wesley, 2001
- **Distributed Systems: Principles and Paradigms**

A. S. Tanenbaum, M. Van Steen; Prentice-Hall, 2002

- **Distributed Operating Systems: Concepts & Practice**
- D. L. Galli; Prentice-Hall, 2000
- **Distributed Operating Systems & Algorithms**
- R. Chow, T. Johnson; Addison-Wesley, 1997
- **Distributed Computing: Principles and Applications**
- M.L. Liu; Addison-Wesley, 2004

Contenido

1. Planificación de procesos
2. Comunicación
3. Servicio de nombres
4. Sincronización

Definición de un proceso

Un proceso es una entidad activa que tiene asociada un conjunto de atributos: código, datos, pila, registros e identificador único.

Representa la entidad de ejecución utilizada por el Sistema Operativo. Frecuentemente se conocen también con el nombre de tareas (“tasks”).

Un programa representa una entidad pasiva. Cuando un programa es reconocido por el Sistema Operativo y tiene asignado recursos, se convierte en proceso

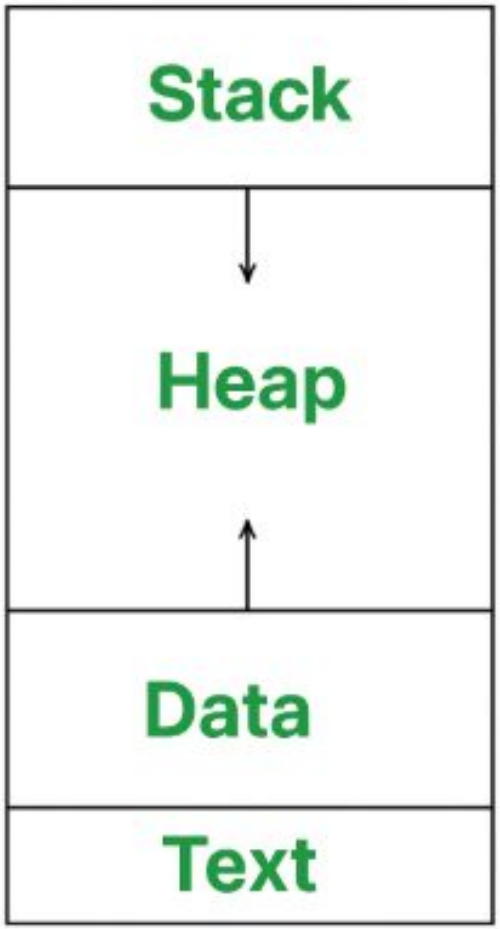
Definición de un proceso

Es la unidad de asignación de recursos: el Sistema Operativo va asignando los recursos del sistema a cada proceso.

Es una unidad de despacho: un proceso es una entidad activa que puede ser ejecutada, por lo que el Sistema Operativo conmuta entre los diferentes procesos listos para ser ejecutados o despachados.

Proceso visto desde la memoria

Datos temporales: funciones, variables locales, direcciones de retorno



Memoria alocada dinámicamente

Variables globales

Instrucciones ejecutables (código) (sección de sólo lectura)

Atributos de un Proceso

Almacenados en una estructura
llamada Bloque de Control de Proceso
(PCB):

PID: Número único para cada proceso

Estado del Proceso: running, waiting, ready to execute

Prioridad y planificación de CPU (CPU scheduling): nivel de prioridad y apuntador a qué proceso sigue en el queue

Información I/O: Info sobre los dispositivos I/O que el proceso está utilizando

Atributos de un Proceso

Descriptor de archivos: Información sobre archivos abiertos y conexiones de red

Contabilización de la información: tiempo que ha corrido el proceso, tiempo de CPU utilizado, uso de otros recursos

Información manejo de memoria: detalles sobre espacio de memoria alojado al proceso; en dónde se encuentra cargado en memoria, la estructura de su memoria (stack, heap)

Ideas generales de un proceso

- Los programas se ejecutan en un contexto llamado proceso
- Un proceso es un programa en ejecución
- Componentes de un proceso: código y una estructura de datos
- Contexto de un proceso: memoria, archivos abiertos, sockets, etc.
- Los procesos tienen un identificador (PID)
- Procesos se alternan en el uso del CPU

Planificación de procesos

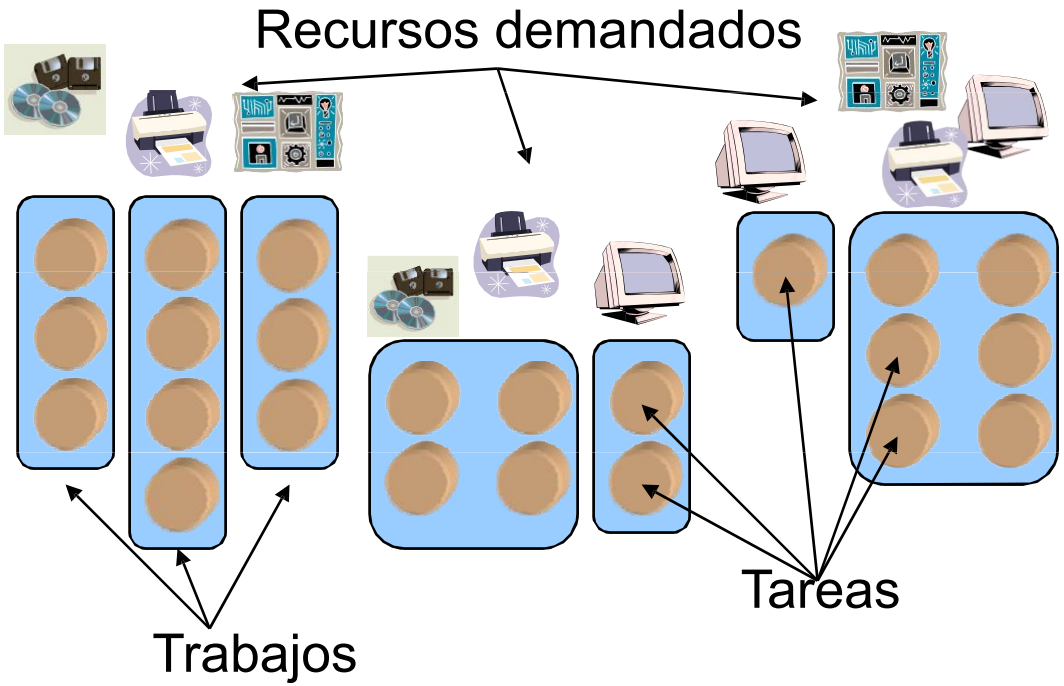
Gestión de procesos

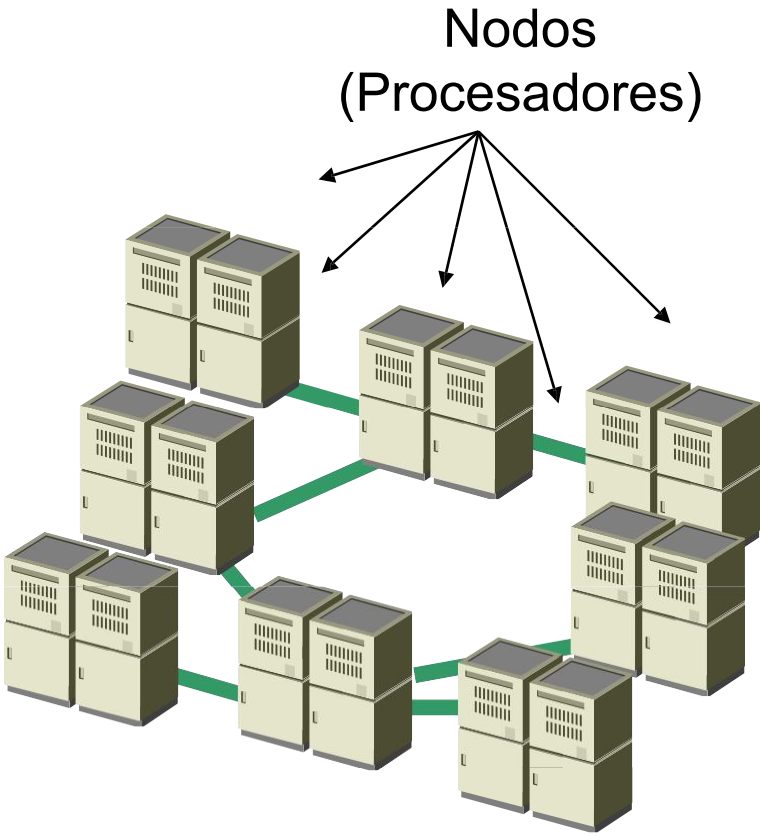
1. Conceptos y taxonomías: Trabajos y sistemas paralelos
2. Planificación estática:
    1. Planificación de tareas dependientes
    2. Planificación de tareas paralelas
    3. Planificación de múltiples tareas
3. Planificación dinámica:
    1. Equilibrado de carga
    2. Migración de procesos
    3. Migración de datos

Escenario de partida: Términos

1. Trabajos: Conjuntos de tareas (procesos) que demandan: (recursos - tiempo)
   - Recursos: Datos, dispositivos, CPU u otros requisitos (finitos) necesarios para la realización de trabajos.
   - Tiempo: Periodo durante el cual los recursos están asignados (de forma exclusiva o no) a un determinado trabajo.
   - Relación entre las tareas: Las tareas se deben ejecutar siguiendo unas restricciones en relación a los datos que generan o necesitan (dependientes y concurrentes)
2. Planificación: Asignación de trabajos a los nodos de proceso correspondientes. Puede implicar revisar, auditar y corregir esa asignación.

Escenario de partida: Términos





OBJETIVO

Asignación de los trabajos de los usuarios a los distintos procesadores, con el objetivo de **mejorar prestaciones** frente a la solución tradicional

Características de un Sistema Distribuido

- Sistemas con memoria compartida
  - Recursos de un proceso accesibles desde todos los procesadores
    - Mapa de memoria
    - Recursos internos del SO (ficheros/dispositivos abiertos, puertos, etc.)
  - Reparto/equilibrio de carga (*load sharing/balancing*) automático
    - Si el procesador queda libre puede ejecutar cualquier proceso listo
  - Beneficios del reparto de carga:
    - Mejora uso de recursos y rendimiento en el sistema
    - Aplicación paralela usa automáticamente procesadores disponibles
- Sistemas distribuidos
  - Proceso ligado a procesador durante toda su vida
  - Recursos de un proceso accesibles sólo desde procesador local
    - No sólo mapa de memoria; También recursos internos del SO
  - Reparto de carga requiere migración de procesos

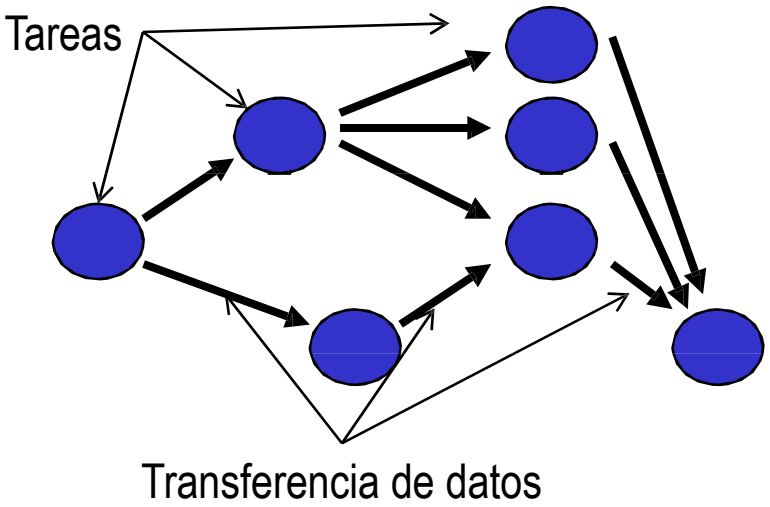
Escenario de partida: Trabajos

- ¿Qué se tiene que ejecutar?
- Tareas en las que se dividen los trabajos:
- Tareas disjuntas
  - Procesos independientes
  - Pertenecientes a distintos usuarios
- Tareas cooperantes
  - Interaccionan entre sí
  - Pertenecientes a una misma aplicación
  - Pueden presentar dependencias
  - O Pueden requerir ejecución en paralelo

Tareas Cooperantes

Dependencias entre tareas

- Modelizado por medio de un grafo dirigido acíclico (DAG).



- Ejemplo: Workflow

Ejecución paralela

- Implican un número de tareas concurrentes ejecutando simultáneamente:
  - De forma síncrona o asíncrona.
  - En base a una topología de conexión.
  - Siguiendo un modelo maestro/esclavo o distribuido.
  - Con unas tasas de comunicación y un intercambio de mensajes.

- Ejemplo: Código MPI

Escenario de partida: objetivos

¿Qué “mejoras de prestaciones” se espera conseguir?

Tipología de sistemas:

- Sistemas de alta disponibilidad
  - HAS: *High Availability Systems*
  - Que el servicio siempre esté operativo
  - Tolerancia a fallos
- Sistemas de alto rendimiento
  - HPC: *High Performance Computing*
  - Que se alcance una potencia de cómputo mayor
  - Ejecución de trabajos pesados en menor tiempo
- Sistemas de alto aprovechamiento
  - HTS: *High Troughput Systems*
  - Que el número de tareas servidas sea el máximo posible
  - Maximizar el uso de los recursos o servir a más clientes (puede no ser lo mismo).

Tipología de Clusters

- High Performance Clusters
  - Beowulf; programas paralelos; MPI; dedicación a un problema
- High Availability Clusters
  - ServiceGuard, Lifekeeper, Failsafe, heartbeat
- High Throughput Clusters
  - Workload/resource managers; equilibrado de carga; instalaciones de supercomputación
- Según servicio de aplicación:
  - Web-Service Clusters
    - LVS/Piranha; equilibrado de conexiones TCP; datos replicados
  - Storage Clusters
    - GFS; sistemas de ficheros paralelos; visión idéntica de los datos desde cada nodo
  - Database Clusters
    - Oracle Parallel Server;

Planificación

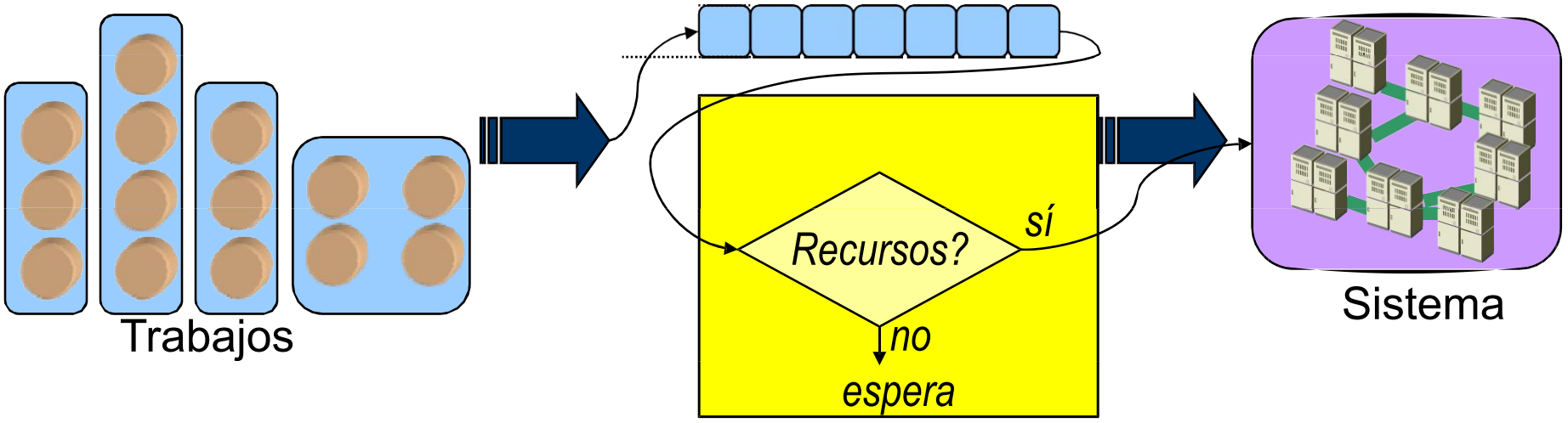
- La planificación consiste en el despliegue de las tareas de un trabajo sobre unos nodos del sistema:
  - Atendiendo a las necesidades de recursos
  - Atendiendo a las dependencias entre las tareas
- El rendimiento final depende de diversos factores:
  - Concurrencia: Uso del mayor número de procesadores simultáneamente.
  - Grado de paralelismo: El grado más fino en el que se pueda descomponer la tarea.
  - Costes de comunicación: Diferentes entre procesadores dentro del mismo nodo y procesadores en diferentes nodos.
  - Recursos compartidos: Uso de recursos (como la memoria) comunes para varios procesadores dentro del mismo nodo.

Planificación

- Dedicación de los procesadores:
  - Exclusiva: Asignación de una tarea por procesador.
  - Tiempo compartido: En tareas de cómputo masivo con E/S reducida afecta dramáticamente en el rendimiento. Habitualmente no se hace.
- La planificación de un trabajo puede hacerse de dos formas:
  - Planificación estática: Inicialmente se determina dónde y cuándo se va a ejecutar las tareas asociadas a un determinado trabajo. Se determina antes de que el trabajo entre en máquina.
  - Planificación dinámica: Una vez desplegado un trabajo, y de acuerdo al comportamiento del sistema, se puede revisar este despliegue inicial. Considera que el trabajo ya está en ejecución en la máquina.

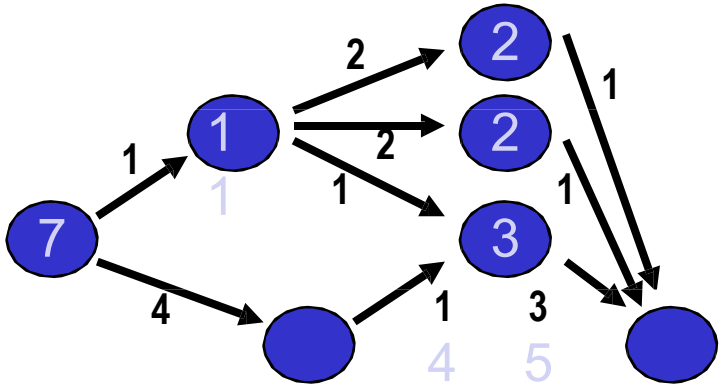
Planificación estática

- Generalmente se aplica antes de permitir la ejecución del trabajo en el sistema.
- El planificador (a menudo llamado *resource manager*) (YARN en Hadoop) selecciona un trabajo de la cola (según política) y si hay recursos disponibles lo pone en ejecución, si no espera.



Planificación de tareas dependientes

- Considera los siguientes aspectos:
  - Duración (estimada) de cada tarea.
  - Volumen de datos transmitido al finalizar la tarea (e.g. fichero)
  - Precedencia entre tareas (una tarea requiere la finalización previa de otras).
  - Restricciones debidas a la necesidad de recursos especiales.



Representado por medio de un grafo acíclico dirigido (DAG)

Una opción consiste en transformar todos los datos a las mismas unidades (tiempo):

- **Tiempo de ejecución (tareas)**
- **Tiempo de transmisión (datos)**

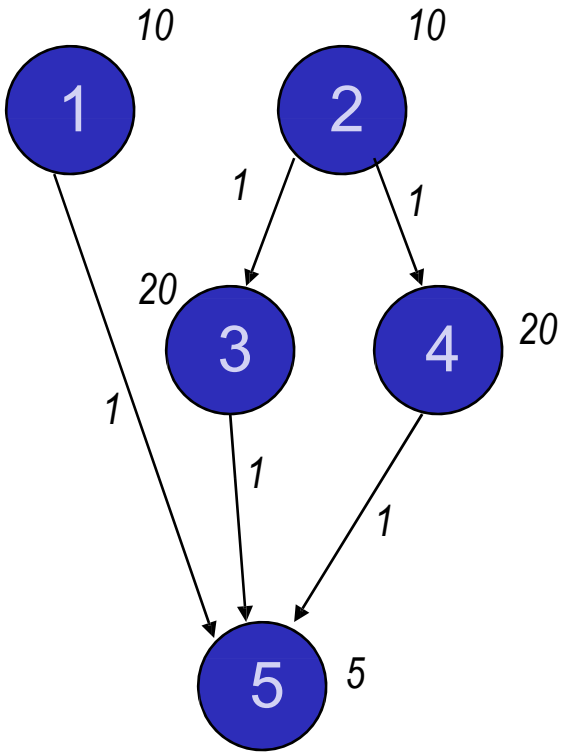
La Heterogeneidad complica estas estimación:

- **Ejecución dependiente de procesador**
- **Comunicación dependiente de conexión**

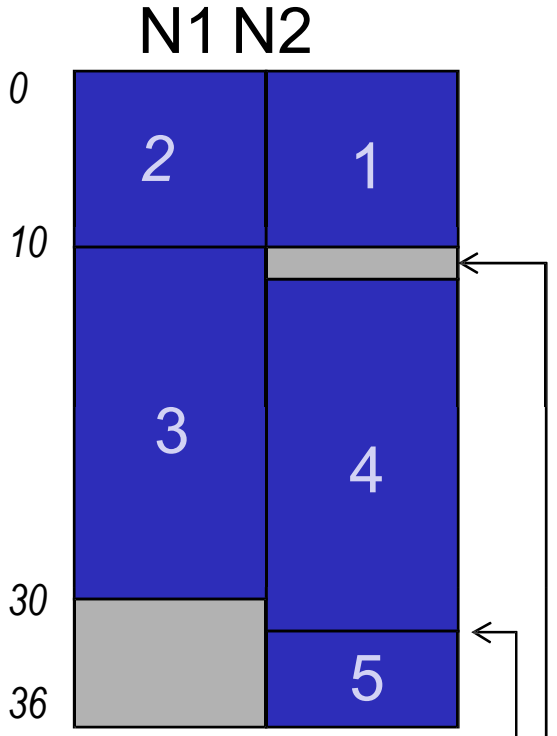
Planificación de tareas dependientes

- Planificación consiste en la asignación de tareas a procesadores en un instante determinado de tiempo:
  - Para un solo trabajo existen heurísticas eficientes: buscar camino crítico (camino más largo en grafo) y asignar tareas implicadas al mismo procesador.
  - Algoritmos con complejidad polinomial cuando sólo hay dos procesadores.
  - Es un problema NP-completo con N trabajos.
  - El modelo teórico se denomina *multiprocessor scheduling*.

Tareas dependientes



Planificador



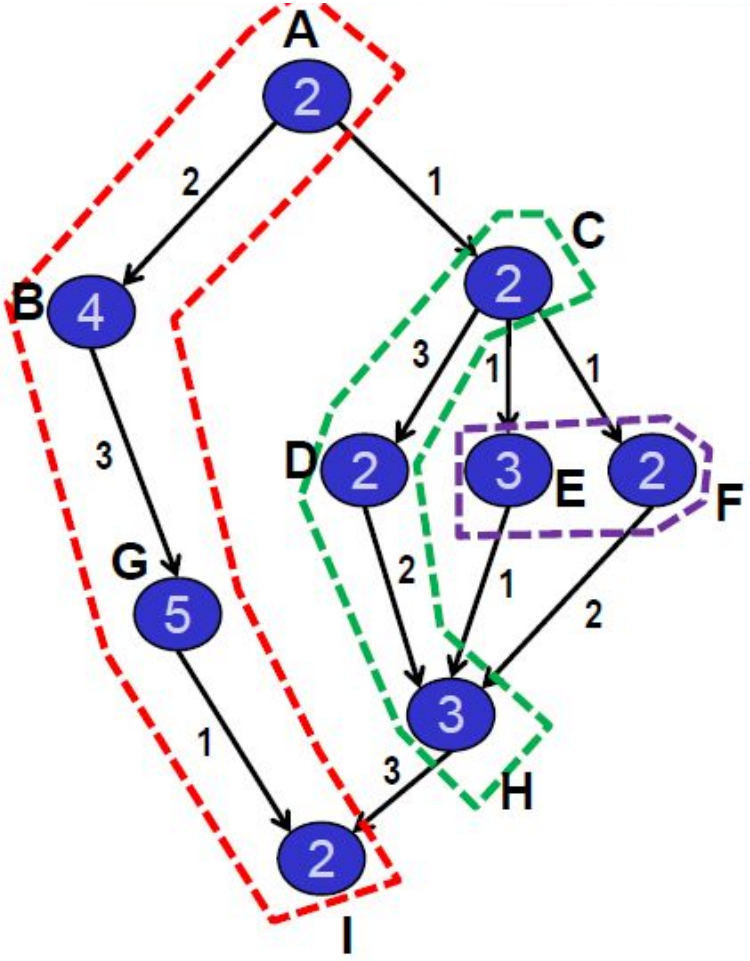
El tiempo de comunicación entre procesos depende de su despliegue en el mismo nodo:

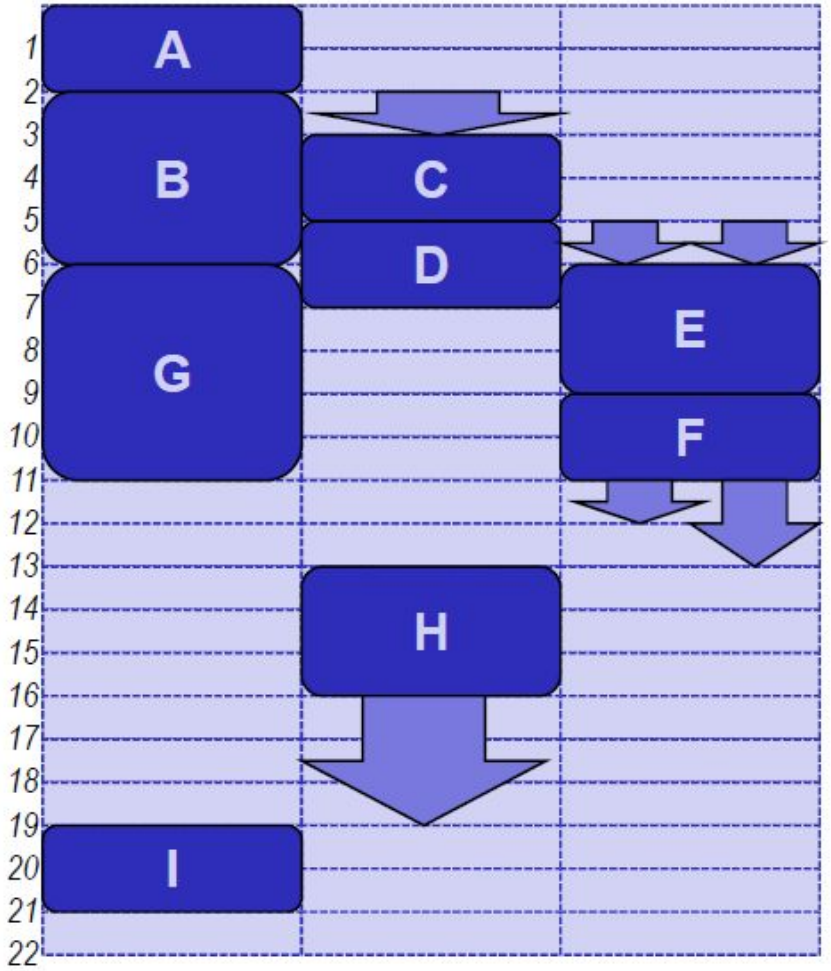
- **Tiempo ~0 si están en el mismo nodo**
- **Tiempo *n* si están en diferentes nodos**

Algoritmos de Clustering

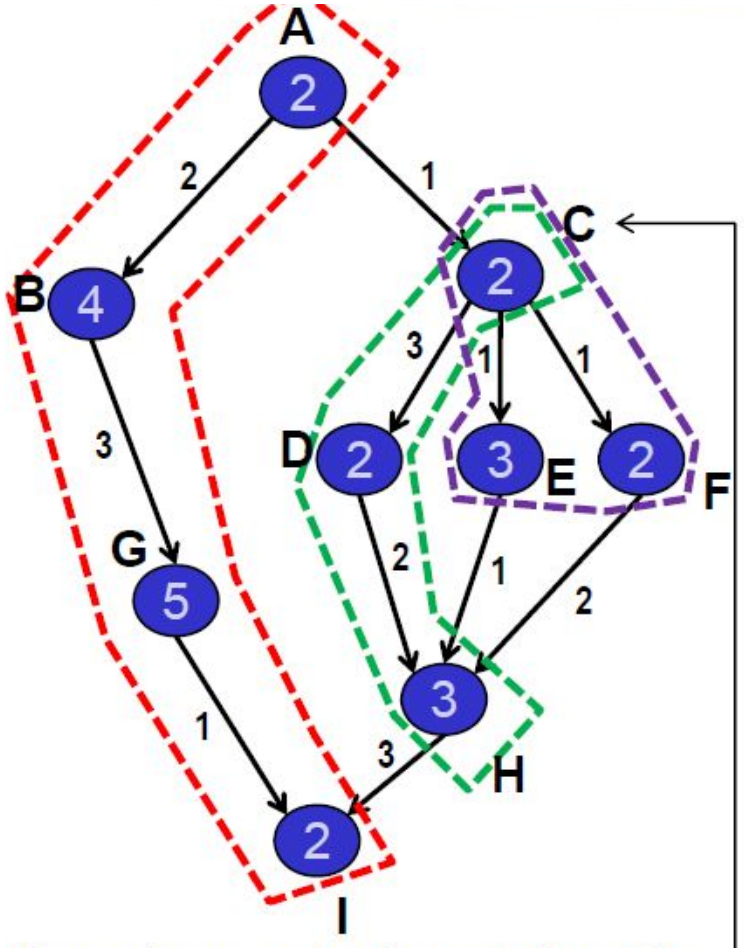
- Para los casos generales se aplican algoritmos de *clustering* que consisten en:
  - Agrupar las tareas en grupos (*clusters*).
  - Asignar cada *cluster* a un procesador.
  - La asignación óptima es NP-completa
  - Este modelo es aplicable a un solo trabajo o a varios en paralelo.
  - Los *clusters* se pueden construir con:
    - Métodos lineales
    - Métodos no-lineales
    - Búsqueda heurística/estocástica

Planificación Clustering

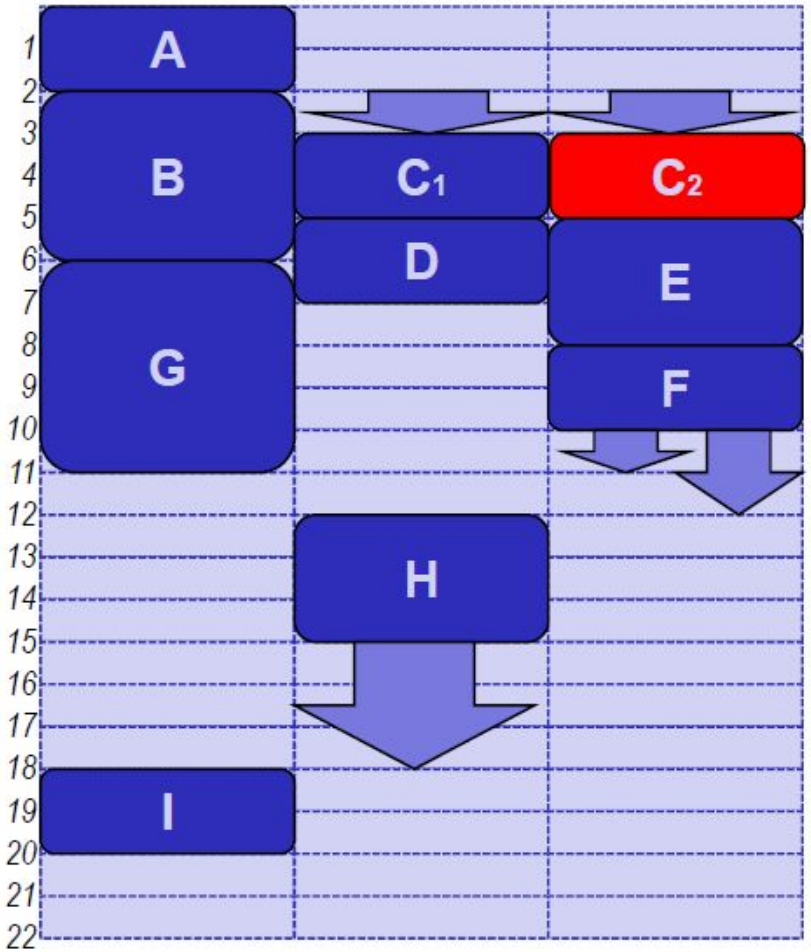




Planificación Clustering



Algunas tareas se ejecutan en varios nodos para ahorrar en la comunicación



Planificación Dinámica

- La planificación estática decide si un proceso se ejecuta en el sistema o no, pero una vez lanzado no se realiza seguimiento de él.
- La planificación dinámica:
  - Evalúa el estado del sistema y toma acciones correctivas.
  - Resuelve problemas debidos a la paralelización del problema (desequilibrio entre las tareas).
  - Reacciona ante fallos en nodos del sistema (caídas o falos parciales).
  - Permite un uso no dedicado o exclusivo del sistema.
  - Requiere una monitorización del sistema (políticas de gestión de trabajos):
    - En la planificación estática se contabilizan los recursos comprometidos.

Load Balancing vs. *Load Sharing*

- *Load Sharing*:
  - Que el estado de los procesadores no sea diferente
  - Un procesador ocioso
  - Una tarea esperando a ser servida en otro procesador

Asignación

- *Load Balancing*:
  - Que la carga de los procesadores sea igual.
  - La carga varía durante la ejecución de un trabajo
  - ¿Cómo se mide la carga?

- Son conceptos muy similares, gran parte de las estrategias usadas para LS vale para LB (considerando objetivos relativamente diferentes). LB tiene unas matizaciones particulares.

Medición de carga

- ¿Qué es un nodo inactivo?
  - Estación de trabajo: “lleva varios minutos sin recibir entrada del teclado o ratón y no está ejecutando procesos interactivos”
  - Nodo de proceso: “no ha ejecutado ningún proceso de usuario en un rango de tiempo”.
  - Planifiación estática: “no se le ha asignado ninguna tarea”.
- ¿Qué ocurre cuando deja de estar inactivo?
  - No hacer nada → El nuevo proceso notará mal rendimiento.
  - Migrar el proceso a otro nodo inactivo (costoso)
  - Continuar ejecutando el proceso con prioridad baja.
- Si en lugar de considerar el estado (LS) se necesita conocer la carga (LB) hacen falta métricas específicas.

Políticas de gestión de trabajos

Toda la gestión de trabajos se basa en una serie de decisiones (políticas) que hay que definir para una serie de casos:

- Política de información: cuándo propagar la información necesaria para la toma de decisiones.
- Política de transferencia: decide cuándo transferir.
- Política de selección: decide qué proceso transferir.
- Política de ubicación: decide a qué nodo transferir.

Planificación Dinámica vs. Estática

- Los sistemas pueden usar indistintamente una, otro o las dos.

	Estática	*No Estática*
Dinámica	***Planificación adaptativa**: Se mantiene un control central sobre los trabajos lanzados al sistema, pero se dispone de los mecanismos necesarios para reaccionar a errores en las estimaciones o reaccionar ante problemas.*	***Estrategias de equilibrado de carga**: Los trabajos se arrancan libremente en los nodos del sistema y por detrás un servicio de equilibrado de carga/estado ajusta la distribución de tareas a los nodos.*
No Dinámica	***Gestor de recursos** (con asignación batch): Los procesadores se asignan de forma exclusiva y el gestor de recursos mantiene información sobre los recursos comprometidos.*	***Cluster de máquinas sin planificación alguna**: Que sea lo que Dios quiera…*

Comunicación de procesos

Contenido

Arquitectura de comunicaciones

Características de la comunicación

Paso de mensajes

- *Sockets*

Llamadas a procedimientos remotos (RPC)

- RPC de Sun

Entornos distribuidos de objetos

- CORBA
- Java RMI

Introducción

Sistema de comunicación: Espina dorsal del SD.

Modelos de comunicación entre procesos:

- Memoria compartida: no factible, en principio (DSM), en SD
- Paso de mensajes.

Nivel de abstracción en comunicación con paso de mensajes:

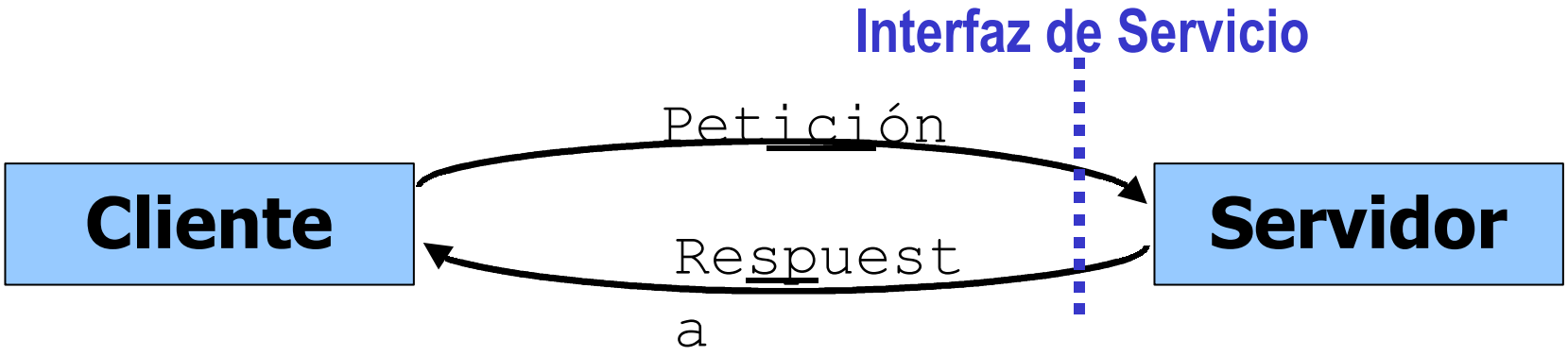
- Paso de mensajes puro.
- Llamadas a procedimientos remotos.
- Modelos de objetos distribuidos.

Arquitectura de comunicaciones

- Modelo cliente/servidor
  - con proxy o caché
  - múltiples capas
  - código móvil
- Modelo *peer-to-peer*

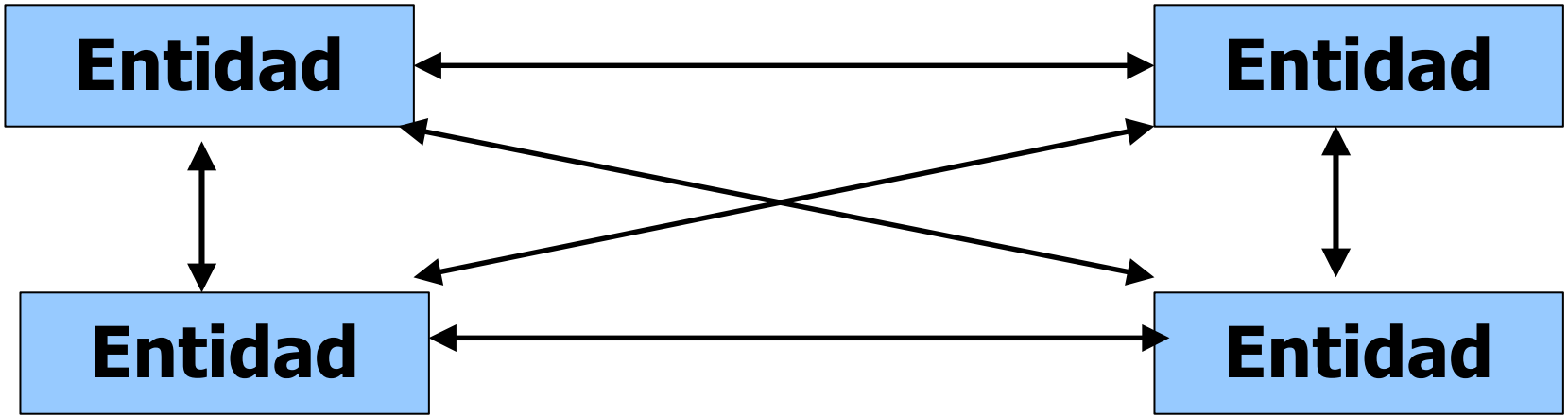
Modelo cliente/servidor

- Dos roles diferentes en la interacción
  - Cliente: Solicita servicio. Petición: Operación + Datos
  - Servidor: Proporciona servicio. Respuesta: Resultado



Modelo peer to peer

- Un único rol: Entidad
- Protocolo de diálogo
  - Entidades se coordinan entre sí
  - Ejemplo: simulación paralela, al final de cada etapa las entidades se sincronizan e intercambian información



Características de la comunicación

Alternativas de diseño

- Modo de operación bloqueante o no bloqueante
- Fiabilidad
- Comunicación síncrona vs. asíncrona

Bloqueantes vs. no bloqueantes

Envío:

- Envío no bloqueante: El emisor almacena el dato en un buffer del núcleo (que se encarga de su transmisión) y reanuda su ejecución.
- Envío bloqueante: El emisor se bloquea hasta que ha sido enviado correctamente al destino.

Recepción:

- Recepción no bloqueante: Si hay un dato disponible el receptor lo lee, en otro caso indica que no había mensaje.
- Recepción bloqueante: Si no hay un dato disponible el receptor se bloquea.

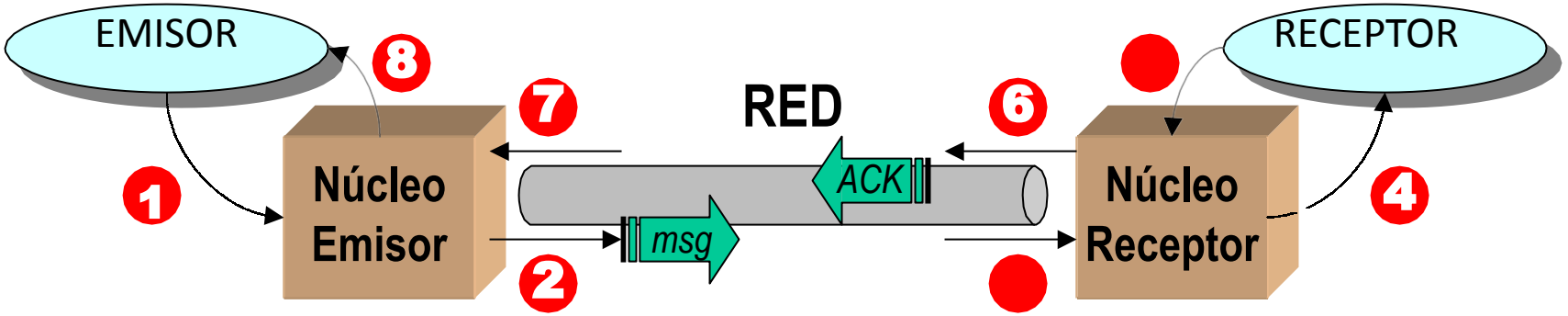
Fiabilidad

Aspectos relacionados con la fiabilidad de la comunicación:

- Garantía de que el mensaje ha sido recibido en nodo(s) destino(s)
- Mantenimiento del orden en la entrega de los mensajes
- Control de flujo para evitar "inundar" al nodo receptor

Si el sistema de comunicación no garantiza algunos de estos aspectos, debe hacerlo la aplicación

Comunicación síncrona vs. asíncrona



- **Envío no bloqueante (comunicación asíncrona):** [1:8] El emisor continúa al pasar el mensaje al núcleo.
- **Envío bloqueante no fiable (comunicación asíncrona):** [1:2:7:8] El emisor espera a que el núcleo transmita por red el mensaje.
- **Envío bloqueante fiable (comunicación asíncrona):** [1:2:3:6:7:8]: El emisor espera a que el núcleo receptor recoge el mensaje.
- **Envío bloqueante explícito (comunicación síncrona):** [1:2:3:4:5:6:7:8]: Idem al anterior, pero es la aplicación receptora la que confirma la recepción.
- **Petición-Respuesta (cliente/servidor) :** [1:2:3:4:<servicio>:5:6:7:8]: Emisor espera a que el receptor procese la operación. Respuesta puede servir de ACK.

Paso de mensajes

Sockets

Paso de mensajes

Primitivas de comunicación hipotéticas:

- Envío: `send(destino,mensaje)`
- Recepción: `receive(&origen,&mensaje)`

Mecanismo puede estar orientado a conexión o no:

- Con conexión: mayor fiabilidad
  - orden de mensajes, control de flujo, fragmentación automática, ...
- Sin conexión: menor sobrecarga inicial

Modo de comunicación típico:

- `Send` no bloqueante y `Receive` bloqueante
- Comunicación asíncrona

Las aplicaciones definen el protocolo de comunicación:

- En arquitectura cliente/servidor: Petición-respuesta

Cliente/servidor con paso de mensajes

Los modelos de comunicación basados en cliente-servidor con paso de mensajes responden al esqueleto:


```
Mensaje petic,resp;

petic.cod_op=OPER_1
;
petic.param1=<...>;
...
petic.paramN=<...>;
send(servidor,petic)
;
receive(NULL,&resp);
...
```

```
Mensaje petic,resp;
receive(&cliente,&petic)
;  switch(petic.cod_op)
{
  case   OPER_1:
    Realiza  petición
    con paráms.  y
    genera   resultados
    resp.resul=<...>;
  case   OPER_2: ...
}
send(cliente,resp);
```

Mensajes de texto

Los mensajes pueden ser de texto.

Estructura del Mensaje:

- Cadenas de caracteres.
- Por ejemplo HTTP:

```
"GET //www.fi.upm.es HTTP/1.1"
```

Envío del Mensaje:

```
send(destino, "GET //www.fi.upm.es HTTP/1.1");
```

Mensajes binarios

Los mensajes pueden tener formato binario

estructura del Mensaje:

```
struct mensaje_st {
 unsigned int msg_tipo;  unsigned
 int msg_seq_id;  unsigned char
 msg_data[1024];
};
```

Envío del Mensaje:

```
struct mensaje_st confirm;
confirm.msg_tipo = MSG_ACK;
confirm.msg_seq_id = 129;

send(destino, confirm);
```

Sockets

Aparecieron en 1981 en UNIX BSD 4.2

- Intento de incluir TCP/IP en UNIX.
- Diseño independiente del protocolo de comunicación.

Un socket es punto final de comunicación (dirección IP y puerto).

Abstracción que:

- Ofrece interfaz de acceso a los servicios de red en el nivel de transporte.
- Representa un extremo de una comunicación bidireccional con una dirección asociada.

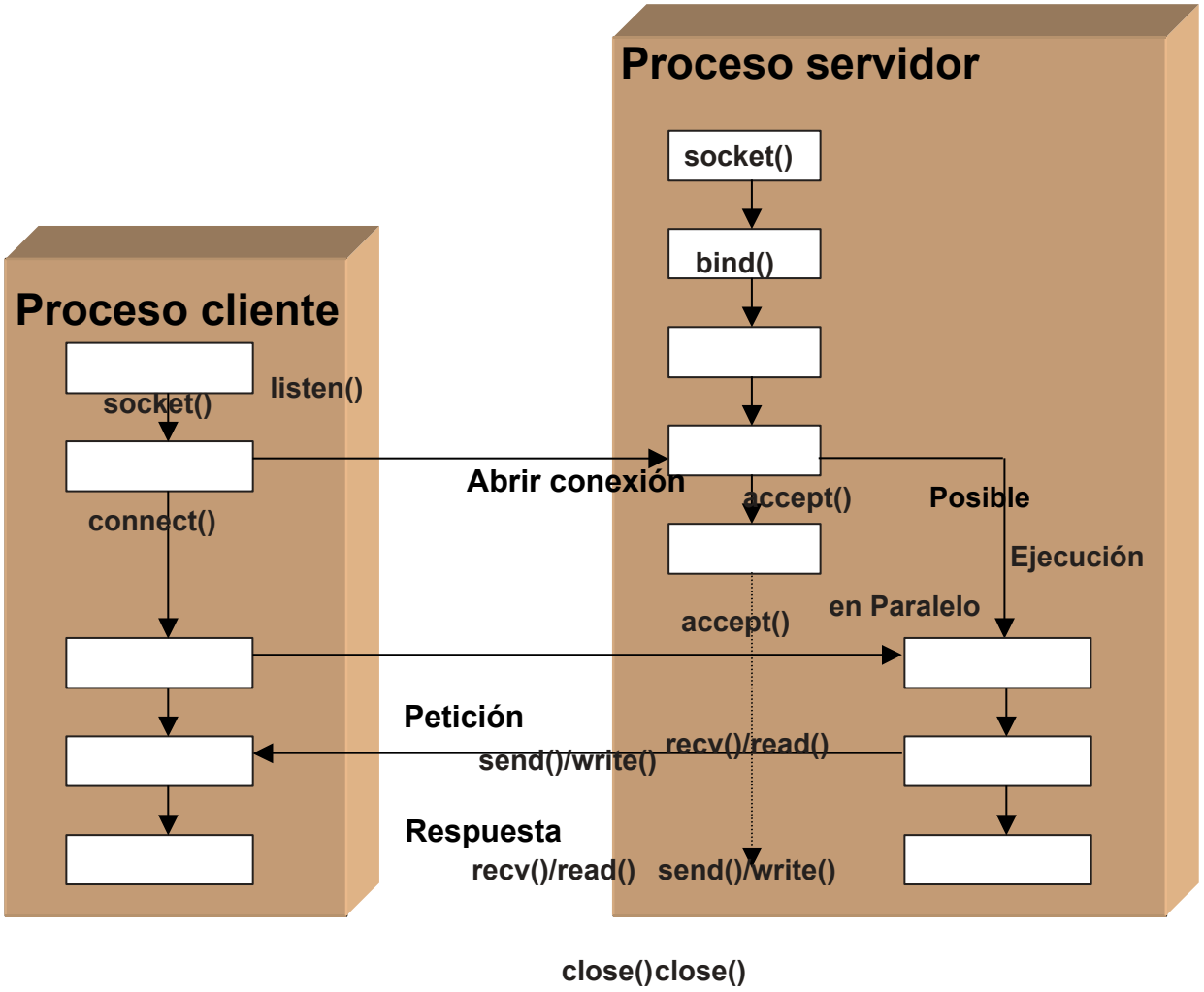
Sockets

Sujetos a proceso de estandarización dentro de POSIX (POSIX 1003.1g).

Actualmente:

- Disponibles en casi todos los sistemas UNIX.
- En prácticamente todos los sistemas operativos:
  - WinSock: API de sockets de Windows.
- En Java como clase nativa.

Idea general



Conceptos básicos sobre sockets

Dominios de comunicación.

Tipos de *sockets*.

Direcciones de *sockets*.

Creación de un *socket*.

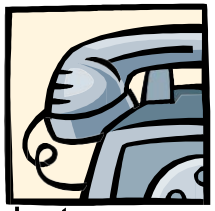
Asignación de direcciones.

Solicitud de conexión.

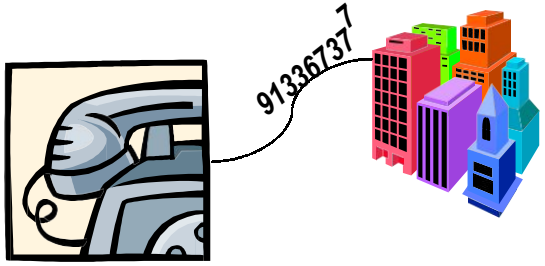
Preparar para aceptar conexiones.

Aceptar una conexión.

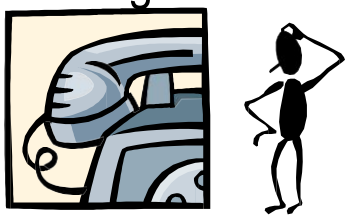
Transferencia de datos.



1.- Creación del socket



2.- Asignación de dirección



3.- Aceptación de conexión

Dominios de comunicación

- Un dominio representa una familia de protocolos.
- Un *socket* está asociado a un dominio desde su creación.
- Sólo se pueden comunicar *sockets* del mismo dominio.
- Los servicios de *sockets* son independientes del dominio.

Algunos ejemplos:

- **`PF_UNIX`** (o **`PF_LOCAL`**): comunicación dentro de una máquina.
- **`PF_INET`**: comunicación usando protocolos TCP/IP.

Tipos de sockets

Stream (SOCK_STREAM):

- Orientado a conexión.
- Fiable, se asegura el orden de entrega de mensajes.
- No mantiene separación entre mensajes.
- Si **PF_INET** se corresponde con el protocolo TCP.

Datagrama (SOCK_DGRAM):

- Sin conexión.
- No fiable, no se asegura el orden en la entrega.
- Mantiene la separación entre mensajes.
- Si **PF_INET** se corresponde con el protocolo UDP.

Raw (SOCK_RAW):

- Permite el acceso a los protocolos internos como IP.

Direcciones de sockets

Cada *socket* debe tener asignada una dirección única.

Dependientes del dominio.

Las direcciones se usan para:

- Asignar una dirección local a un socket (**`bind`**).
- Especificar una dirección remota (**`connect`** o **`sendto`**).

Se utiliza la estructura genérica de dirección:

- **`struct sockaddr mi_dir;`**

Cada dominio usa una estructura específica.

- Uso de *cast* en las llamadas.
- Direcciones en **`PF_INET`** (**`struct sockaddr_in`**).
- Direcciones en **`PF_UNIX`** (**`struct sockaddr_un`**).

Direcciones de sockets en PF_INET

Una dirección destino viene determinada por:

- Dirección del *host*: 32 bits.
- Puerto de servicio: 16 bits.

Estructura **`struct sockaddr_in`**:

- Debe iniciarse a 0 (**`bzero`**).
- **`sin_family`**: dominio (AF_INET).
- **`sin_port`**: puerto.
- **`sin_addr`**: dirección del *host*.

Una transmisión está caracterizada por cinco parámetros únicos:

- Dirección *host* y puerto origen.
- Dirección *host* y puerto destino.
- Protocolo de transporte (UDP o TCP).

Obtención de la dirección del host

Usuarios manejan direcciones en forma de texto:

- decimal-punto: 138.100.8.100
- dominio-punto: laurel.datsi.fi.upm.es

Conversión a binario desde decimal-punto:

```
int inet_aton(char *str,struct in_addr *dir)
```

- **`str`**: contiene la cadena a convertir.
- **`dir`**: resultado de la conversión en formato de red.

Conversión a binario desde dominio-punto:

```
struct hostent *gethostbyname(char *str)
```

- **`str`**: cadena a convertir.
- Devuelve la estructura que describe al *host*.

Creación de un socket

La función **`socket`** crea uno nuevo:

```
int socket(int dom,int tipo,int proto)
```

- Devuelve un descriptor de fichero (igual que un **`open`** de fichero).
- Dominio (**`dom`**): **`PF_`***XXX*
- Tipo de socket (**`tipo`**): **`SOCK_`***XXX*
- Protocolo (**`proto`**): Dependiente del dominio y del tipo:
  - 0 elige el más adecuado.
  - Especificados en **`/etc/protocols`**

El socket creado **no** tiene dirección asignada.

Asignación de direcciones

La asignación de una dirección a un *socket* ya creado:

`int bind(int s,struct sockaddr* dir,int tam)`

- Socket (**`s`**): Ya debe estar creado.
- Dirección a asignar (**`dir`**): Estructura dependiendo del dominio
- Tamaño de la dirección (**`tam`**): **`sizeof()`** .

Si no se asigna dirección (típico en clientes) se le asigna automáticamente (puerto efímero) en la su primera utilización **`connect`** o **`sendto`**).

Asignación de direcciones (PF_INET)

Direcciones en dominio **`PF_INET`**

- Puertos en rango 0..65535.
- Reservados: 0..1023.
- Si se le indica el 0, el sistema elige uno.
- Host: una dirección IP de la máquina local.
  - **`INADDR_ANY`**: elige cualquiera de la máquina.

Si el puerto solicitado está ya asignado la función **`bind`** devuelve un valor negativo.

El espacio de puertos para *streams* (TCP) y datagramas (UDP) es independiente.

Solicitud de conexión

Realizada en el cliente por medio de la función:

`int connect(int s,struct sockaddr* d,int tam)`

- Socket creado (**`s`**).
- Dirección del servidor (**`d`**).
- Tamaño de la dirección (**`tam`**).

Si el cliente no ha asignado dirección al socket, se le asigna una automáticamente.

Normalmente se usa con *streams*.

Preparar para aceptar conexiones

Realizada en el servidor *stream* después de haber creado (**`socket`**) y reservado dirección (**`bind`**) para el socket: **`int listen(int sd, int baklog)`**

- Socket (**`sd`**): Descriptor de uso del socket.
- Tamaño del buffer (**`backlog`**): Número máximo de peticiones pendientes de aceptar que se encolarán (algunos manuales recomiendan 5)

Hace que el socket quede preparado para aceptar conexiones.

Aceptar una conexión

Realizada en el servidor *stream* después de haber preparado la conexión (**`listen`**):

```
int accept(int s,struct sockaddr *d,int *tam)
```

- Socket (**`sd`**): Descriptor de uso del socket.
- Dirección del cliente (**`d`**): Dirección del socket del cliente devuelta.
- Tamaño de la dirección (**`tam`**): Parámetor valor-resultado
  - Antes de la llamada: tamaño de dir
  - Después de la llamada: tamaño de la dirección del cliente que se devuelve.

Aceptar una conexión

La semántica de la función **`accept`** es la siguiente:

Cuando se produce la conexión, el servidor obtiene:

- La dirección del socket del cliente.
- Un nuevo descriptor (socket) que queda conectado al socket del cliente.

Después de la conexión quedan activos dos sockets en el servidor:

- El original para aceptar nuevas conexiones
- El nuevo para enviar/recibir datos por la conexión establecida.

Idealmente se pueden plantear servidores *multithread* para servicio concurrente.

Otras funcionalidades

Obtener la dirección a partir de un descriptor:

- Dirección local: **`getsockname()`**.
- Dirección del socket en el otro extremo: **`getpeername()`**.

Transformación de valores:

- De formato *host* a red:
  - Enteros largos: htonl().
  - Enteros cortos: htons().
- De formato de red a *host*:
  - Enteros largos: ntohl().
  - Enteros cortos: ntohs().

Cerrar la conexión:

- Para cerrar ambos tipos de sockets: **`close()`**.
  - Si el socket es de tipo stream cierra la conexión en ambos sentidos.
- Para cerrar un único extremo: **`shutdown()`**.

Transferencia de datos con streams

Envío:

Puede usarse la llamada **`write`** sobre el descriptor de socket.

```
int send(int s,char *mem,int tam,int flags)
```

- Devuelve el nº de bytes enviados.

Recepción:

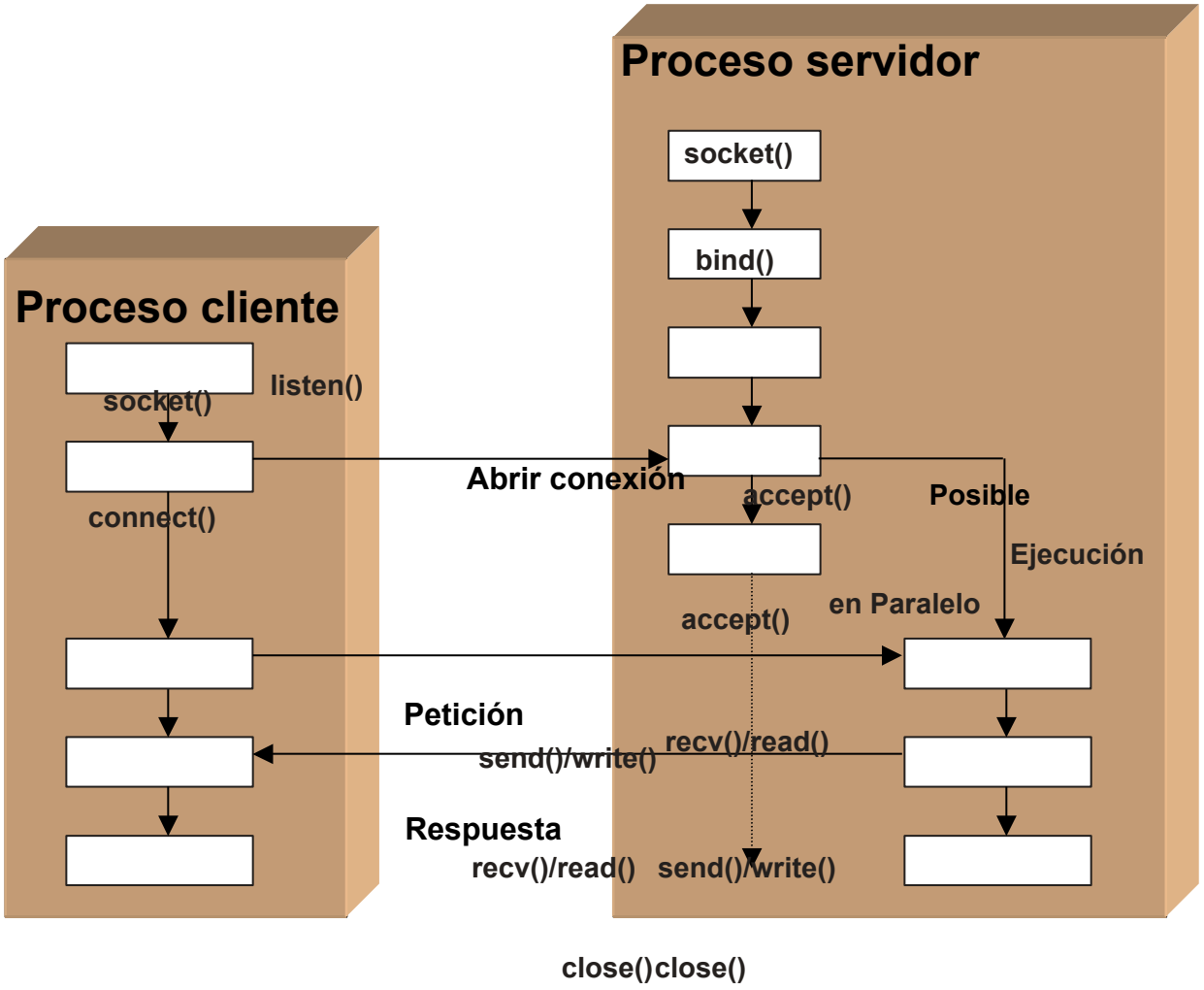
Puede usarse la llamada **`read`** sobre el descriptor de socket.

```
int recv(int s,char *mem,int tam,int flags)
```

- Devuelve el nº de bytes recibidos.

Los flags implican aspectos avanzado como enviar o recibir datos urgentes (*out-of-band*).

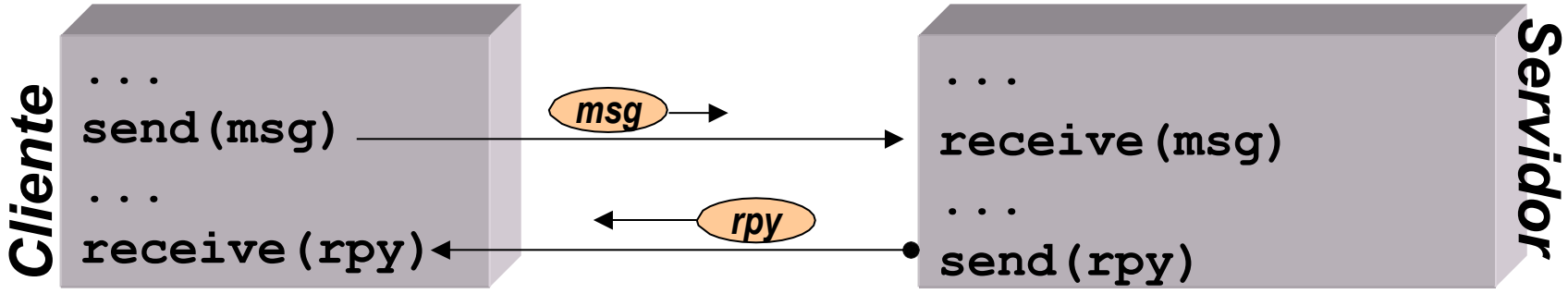
Escenario de uso de sockets



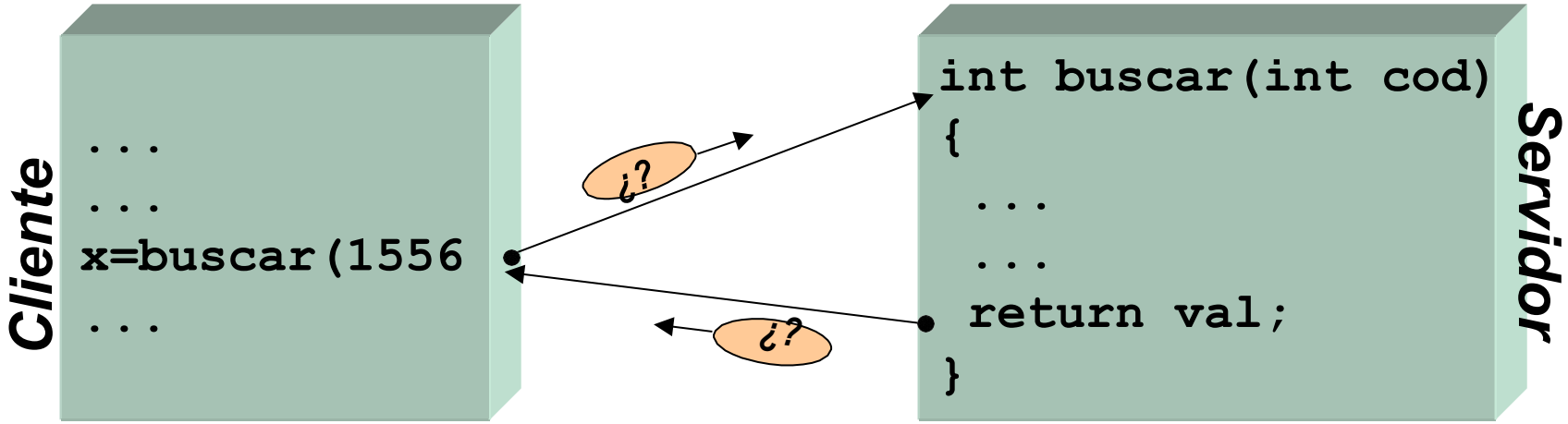
Llamadas a procedimientos remotos

RPC

Llamadas a procedimientos remotos



Paso de mensajes (visión de bajo nivel)



Llamadas a procedimientos remotos (más alto nivel) → *Comodidad*

Llamadas a procedimientos remotos

Remote Procedure Call: RPC.

Evolución:

- Propuesto por Birrel y Nelson en 1985.
- Sun RPC es la base para varios servicios actuales (NFS o NIS
- Llegaron a su culminación en 1990 con DCE (*Distributed Computing Environment*) de OSF.
- Han evolucionado hacia orientación a objetos: invocación de métodos remotos (CORBA, RMI).

Llamadas a procedimientos remotos

Cliente:

- El proceso que realiza una la llamada a una función.
- Dicha llamada empaqueta los argumentos en un mensaje y se los envía a otro proceso.
- Queda la espera del resultado.

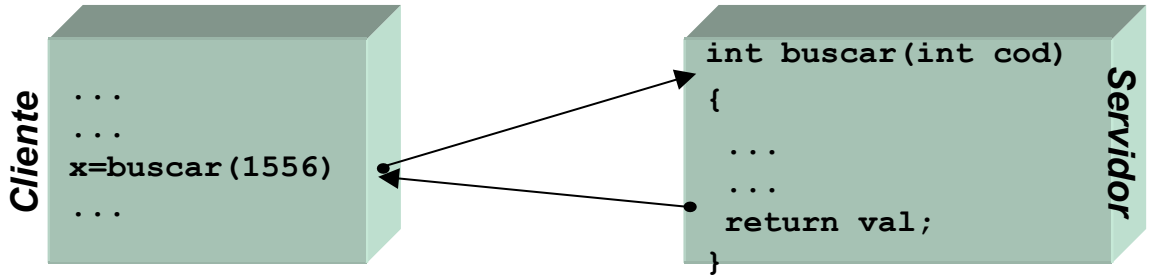
Servidor:

- Se recibe un mensaje consistente en varios argumentos.
- Los argumentos son usados para llamar una función en el servidor.
- El resultado de la función se empaqueta en un mensaje que se retransmite al cliente.

Objetivo: acercar la semántica de las llamadas a procedimiento convencional a un entorno distribuido (transparencia).

Elementos necesario

- Código cliente.
- Código del servidor.
- Formato de representación.
- Definición del interfaz.
- Localización del servidor.
- Semánticas de fallo.



Entornos distribuidos de objetos

CORBA

Java RMI

Middleware

Software de conectividad que consiste en un conjunto de servicios que permiten interactuar a múltiples procesos que se ejecutan en distintas máquinas a través de una red.

Ocultan la heterogeneidad, abstrae la complejidad subyacente y proveen de un modelo de programación conveniente para los desarrolladores de aplicaciones.

Middleware

En la actualidad se cuenta con muchos productos y estándares Middlewares que dan soporte a los sistemas orientados a objetos, entre ellos podemos señalar a :

1. CORBA,
2. Java RMI,
3. Web Services,
4. DCOM (Distributed Component Object Model) de Microsoft y
5. los modelos de referencia para procesamiento distribuido abierto (RM-ODP) del ISO/ITU-T.

CORBA

Concepto ¿Qué es?

- **CORBA** (***C****ommon* ***O****bject* ***R****equest* ***B****roker* ***A****rchitecture*).
- Es una herramienta que facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos (HW y SW)
  - Distintos sistemas operativos (Unix, Windows, MacOS, OS/2)
  - Distintos protocolos de comunicación (TCP/IP, IPX, …)
  - Distintos lenguajes de programación (Java, C, C++, …)
- Define la infraestructura para la arquitectura **OMA** (***O****bject* ***M****anagement* ***A****rchitecture*) especificando los entándares necesarios para la invocación de métodos sobre objetos en entornos heterogéneos

CORBA

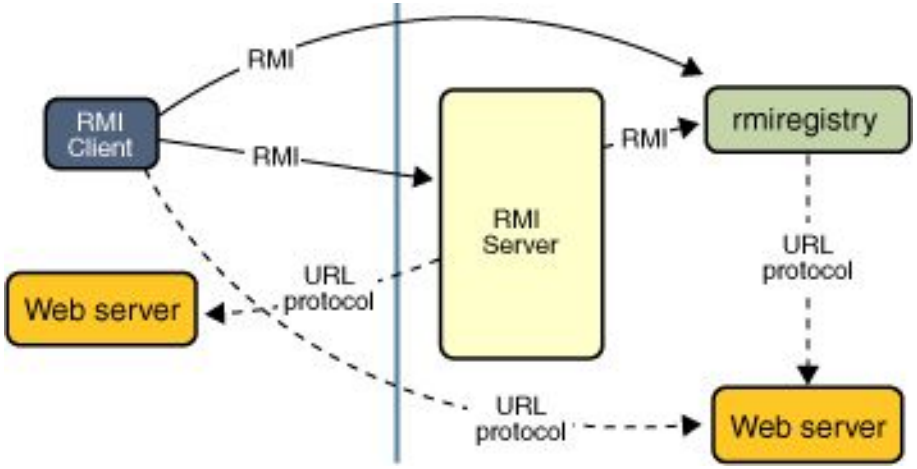
¿Para qué sirve?

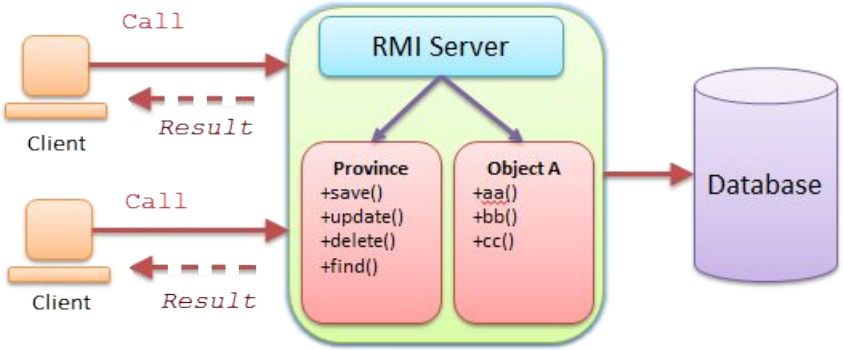
- Permitir invocación de métodos de un objeto por objetos que residen en diferentes máquinas en entornos heterogéneos
  - Los objetos pueden estar desarrollados en diferentes lenguajes.
  - Los equipos pueden tener diferente:
    - Hardware
    - Sistema operativo
  - Los equipos pueden estar conectados entre sí usando distintos protocolos de comunicación
- Facilitar el desarrollo de aplicaciones distribuidas

JAVA RMI (Remote Method Invocation)

- La invocación remota de métodos de Java es un modelo de objetos distribuidos, diseñado específicamente para ese lenguaje, por lo que mantiene la semántica de su modelo de objetos locales, facilitando de esta manera la implantación y el uso de objetos distribuidos.
- En el modelo de objetos distribuidos de Java, un objeto remoto es aquel cuyos métodos pueden ser invocados por objetos que se encuentran en una máquina virtual (MV) diferente.
- Los objetos de este tipo se describen por una o más interfaces remotas que contienen la definición de los métodos del objeto que es posible invocar remotamente.

JAVA RMI (Remote Method Invocation)





Servicios de nombres

Servicios de nombres

- Introducción
- Servicio de nombres
  - Estudio de un ejemplo práctico: DNS
- Servicio de directorio
  - Estudio de un ejemplo práctico: LDAP
- Descubrimiento de servicios

Historia

- Quiero contactar con persona en un contexto para pedirle algo
  - Contexto: una organización, una ciudad, un país, el mundo, ...
  - Necesito su dirección de contacto (p.e. nº teléfono en ese contexto)
  - *Nombre* (*quién*) [permanente] → *Dirección* (*dónde*) [transitorio]
    - Veremos que nombres y direcciones no son tan diferentes…

- Servicio telefónico "páginas blancas" (Servicio de nombres)
  - Necesito conocer nº teléfono de servicio de guía del contexto dado
  - Y especificar la persona con "nombre" unívoco en ese contexto
    - Nombre/apellidos | nº empleado | nº DNI
  - Guía proporciona nivel de indirección respecto a dirección contacto
    - Permite que persona cambie nº tfno (cuidado con agenda-caché)
  - Puede requerirse cadena de consultas; ¿nº tfno empleado?:
    - 1º obtengo nº tfno empresa; 2º centralita empresa me da tfno empleado

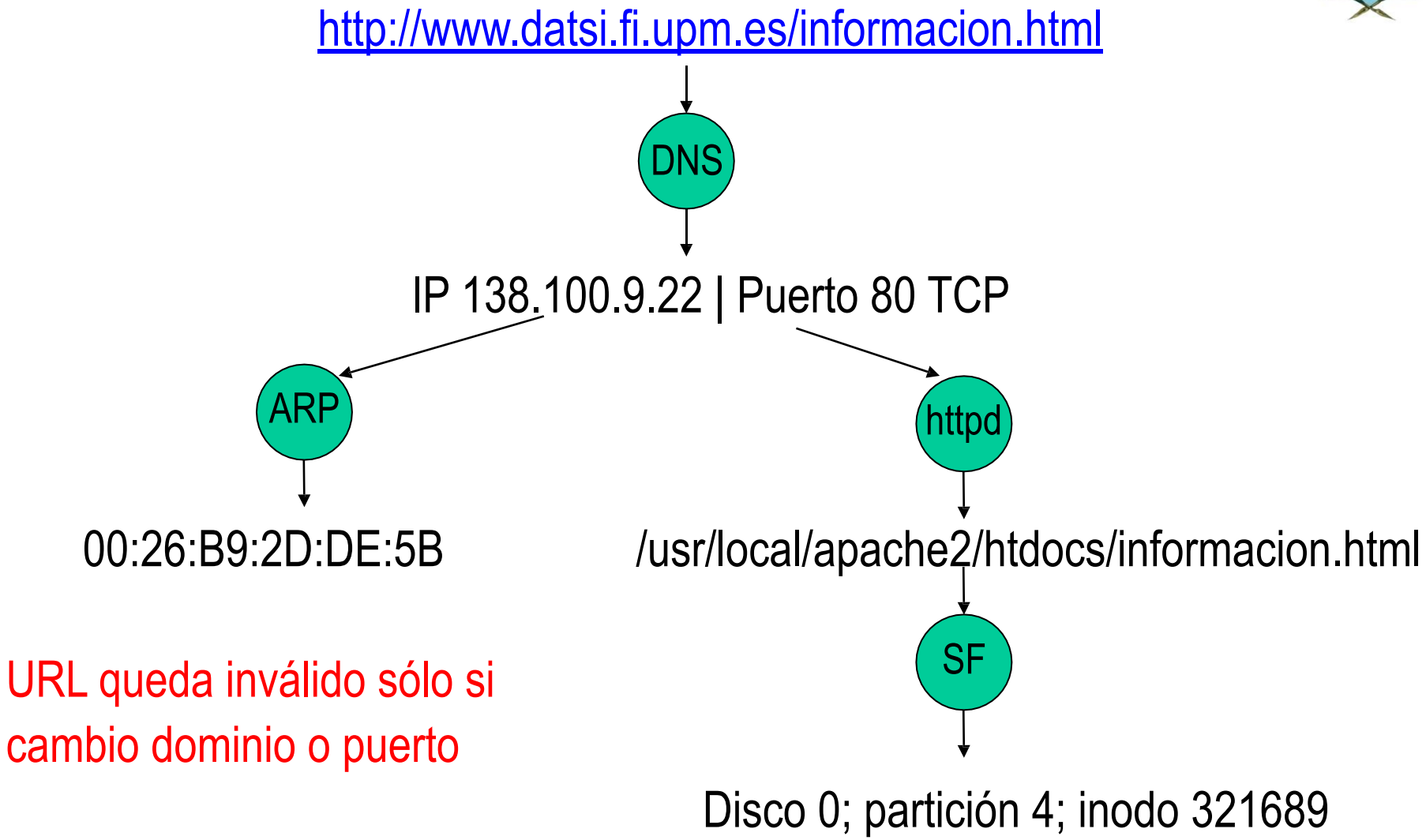
Historia

- Nombres y direcciones suelen tener carácter jerárquico
  - Facilita su administración y gestión
  - Ejs. Nombres: ID empleado internacional (ISBN, cuenta bancaria, ...)
    - Cambio de recurso en jerarquía puede invalidar el nombre
  - Ejs. Direcciones: nº teléfono o dirección postal
  - Encaminamiento jerárquico
- A veces quiero contactar con cualquiera que dé un servicio
  - Necesito conocer condiciones de servicio para elegir
- Servicio telefónico “páginas amarillas” (**Servicio de directorio**)
- ¿Y si ni siquiera sé nº tfno. de servicios de guía (o no los hay)?
  - Quizás debería gritar pidiendo ayuda
  - **Descubrimiento de servicios** (Computación ubicua)

URI: Uniform Resource Identifier

- 2 tipos de identificadores de recursos URIs en Internet:
  - Nombres URNs y direcciones URLs
- *Uniform Resource Name*: *Nombre* (*qué*) [permanente]
  - Identifican recurso sin incluir información de localización
  - Requiere un proceso de traducción
- *Uniform Resource Locator*: *Dirección* (*dónde*) [¿transitorio?]
  - Pueden verse afectados si recurso “se mueve”
  - Algunos URL se pueden considerar permanentes
- Ejemplos wikipedia URN vs. URL
  - *urn:ietf:rfc:3187*
  - *http://tools.ietf.org/html/rfc3187.html*

Ejemplo: Niveles de traducción de URL



URL queda inválido sólo si cambio dominio o puerto

Servicio de nombres

- Nombre de entidad en SD → punto(s) de acceso a la entidad
  - Sockets: Dir(s) IP+ puerto(s)+ protocolo(s)
  - RMI: referencia(s) a objeto(s)
- Nombre permite referirse a una entidad única en SD
  - Aunque puede estar replicada (p.e. fichero en Coda)
  - y puede haber varios nombres para la misma entidad (alias)
- Hay diversos tipos de entidades en SD
  - ficheros, usuarios, grupos, procesos, dispositivos, máquinas, ...
- Serv. de nombres específicos para algunos tipos de entidades
  - para ficheros (SFD), para máquinas (DNS), ...
- Ideal: servicio de nombres integral para todas las entidades
  - Excepto ficheros por su gran número y frecuencia de actualizaciones

Sincronización, concurrencia y transacciones

Sincronización en Sistemas Distribuidos

Más compleja que en los centralizados

Propiedades de algoritmos distribuidos:

- La información relevante se distribuye entre varias máquinas.
- Se toman decisiones sólo en base a la información local.
- Debe evitarse un punto único de fallo.
- No existe un reloj común.

Problemas a considerar:

- Tiempo y estados globales.
- Exclusión mutua.
- Algoritmos de elección.
- Operaciones atómicas distribuidas: Transacciones

Sincronización de relojes físicos

Relojes hardware de un sistema distribuido no están sincronizados.

Necesidad de una sincronización para:

- En aplicaciones de tiempo real.
- Ordenación natural de eventos distribuidos (fechas de ficheros).

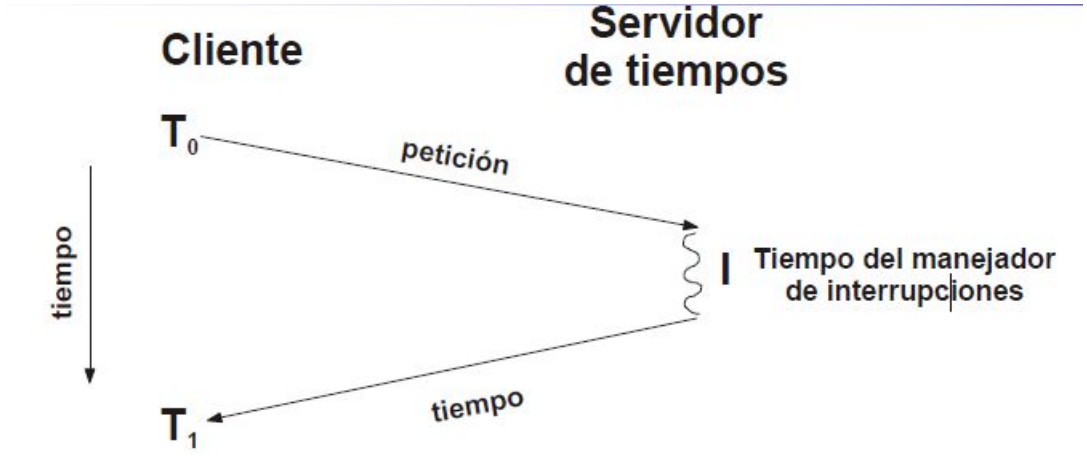
Concepto de sincronización:

- Mantener relojes sincronizados entre sí.
- Mantener relojes sincronizados con la realidad.

UTC: Universal Coordinated Time

- Transmisión de señal desde centros terrestres o satélites.
- Una o más máquinas del sistema distribuido son receptoras de señal UTC.

Algoritmo de Cristian

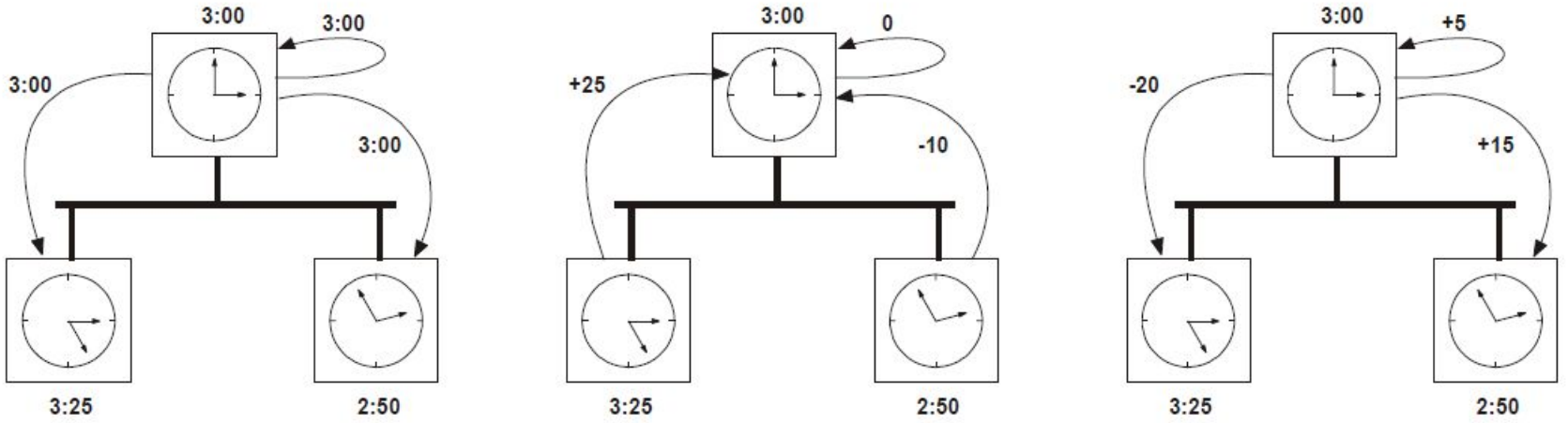


Adecuado para sincronización con UTC.

- Tiempo de transmisión del mensaje: (T1-T0)/2
- Tiempo en propagar el mensaje: (T1-T0-I)/2
- Valor que devuelve el servidor se incrementa en (T1-T0-I)/2
- Para mejorar la precisión se pueden hacer varias mediciones y descartar cualquiera en la que T1-T0 exceda de un límite

Algoritmo de Berkeley

- El servidor de tiempo realiza un muestreo periódico de todas las máquinas para pedirles el tiempo.
- Calcula el tiempo promedio e indica a todas las máquinas que avancen su reloj a la nueva hora o que disminuyan la velocidad.
- Si cae servidor: selección de uno nuevo (alg. de elección)



Protocolo de Tiempo de Red

NTP (Network Time Protocol).

- Aplicable a redes amplias (Internet).
- La latencia/retardo de los mensajes es significativa y variable.

Objetivos:

- Permitir sincronizar clientes con UTC sobre Internet.
- Proporcionar un servicio fiable ante fallos de conexión.
- Permitir resincronizaciones frecuentes.
- Permitir protección ante interferencias del servicio de tiempo.

Organización:

- Jerarquía de servidores en diferentes estratos.
- Los fallos se solventan por medio de ajustes en la jerarquía.

Protocolo de Tiempo de Red

La sincronización entre cada par de elementos de la jerarquía:

– Modo *multicast*: Para redes LAN. Se transmite por la red a todos los elementos de forma periódica. Baja precisión.

– Modo de llamada a procedimiento: Similar al algoritmo de Cristian. Se promedia el retardo de transmisión. Mejor precisión.

– Modo simétrico: Los dos elementos intercambian mensajes de sincronización que ajustan los relojes. Mayor precisión.

Los mensajes intercambiados entre dos servidores son datagramas UDP.