



COMUNICACIÓN EN SISTEMAS DISTRIBUIDOS

Tema # VI

Sistemas de operación II

Abril-Julio 2012

Yudith Cardinale

INDICE

- ◆ Protocolos de Comunicación
- ◆ Propiedades Fundamentales de los Protocolos
- ◆ Tipos de Protocolos de Transporte
- ◆ Construcción de Bloques para la Comunicación
- ◆ Modelo Cliente-Servidor (Operaciones Remotas)
- ◆ RPC (Remote Procedure Call)
- ◆ Comunicación en grupo

PROTOSCOLOS DE COMUNICACIÓN

- ♦ Dos de las características claves de los Sistemas Distribuidos son:
 - ♦ La ausencia de memoria compartida
 - ♦ Existencia de múltiples elementos que pueden fallar independientemente
- ♦ Éstas implican que la comunicación se basa en la transferencia de mensajes y la existencia de mecanismos de comunicación entre procesos.

PROTOCOLOS DE COMUNICACIÓN

Funciones de los mecanismos de Comunicación:

- ♦ Permitir la comunicación entre procesos separados sobre una red de computadoras.
- ♦ Proveer protección contra fallas.
- ♦ Proveer ayuda natural para la estructuración modular de grandes aplicaciones distribuidas (interfaces simples).
- ♦ Esconder la distinción entre comunicación remota y local, para permitir reconfiguración estática y dinámica.

PROTOSCOLOS DE COMUNICACIÓN

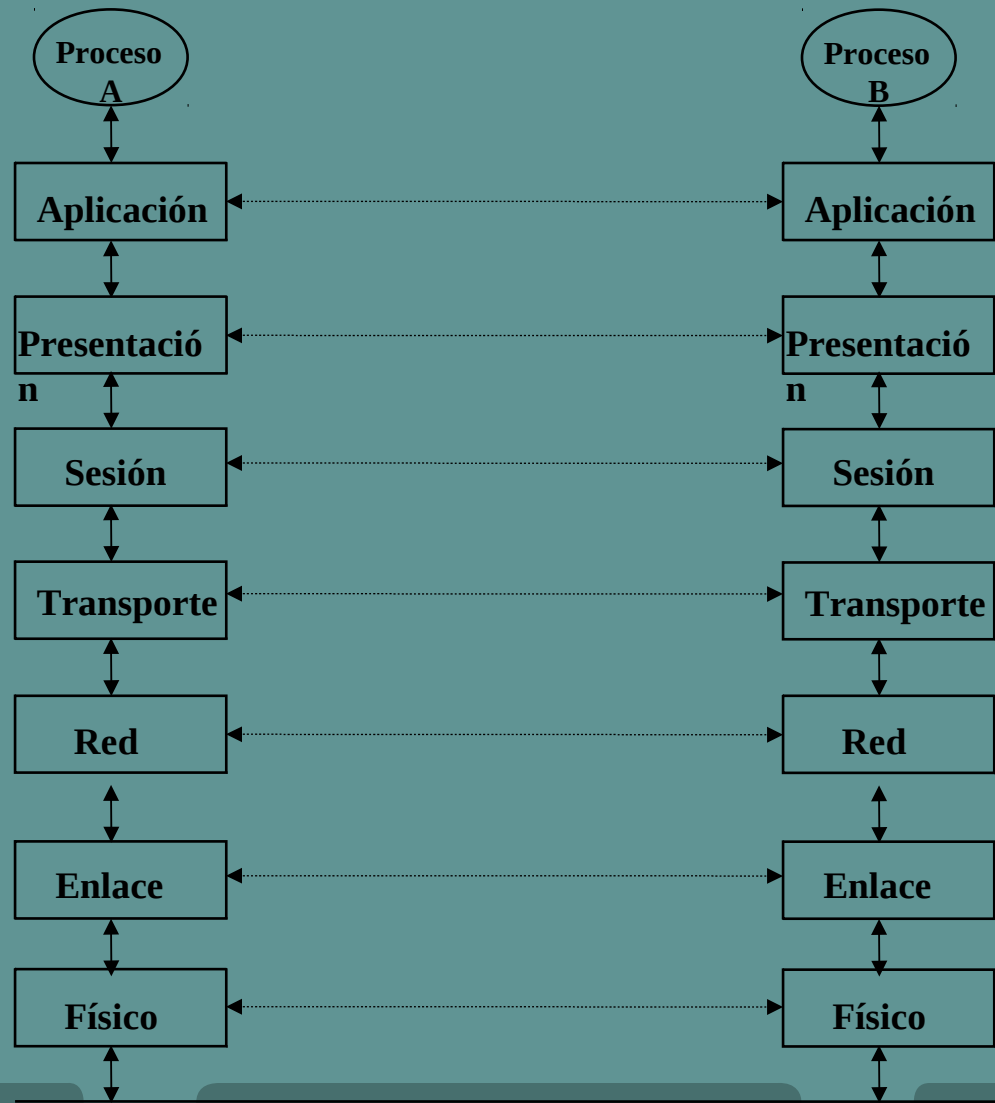
Los mecanismos de comunicación se expresan a través de protocolos de comunicación. Algunos de los más conocidos son:

- Modelo OSI de ISO
 - Modelo ATM (Asynchronous Transfer Mode)
- ♦ Especifica la forma como debe desarrollarse la comunicación (formato, contenido y significado de los mensajes intercambiados)

PROTOSCOLOS DE COMUNICACIÓN

- ♦ El modelo OSI de ISO es una armazón para definir estándares para comunicar computadores heterogéneos (1983). Entre sus características se destacan las siguientes:
 - ♦ Comunicación entre sistemas abiertos
 - ♦ Protocolos orientados a conexión
 - ♦ Protocolos sin conexión

PROTOSCOLOS DE COMUNICACIÓN

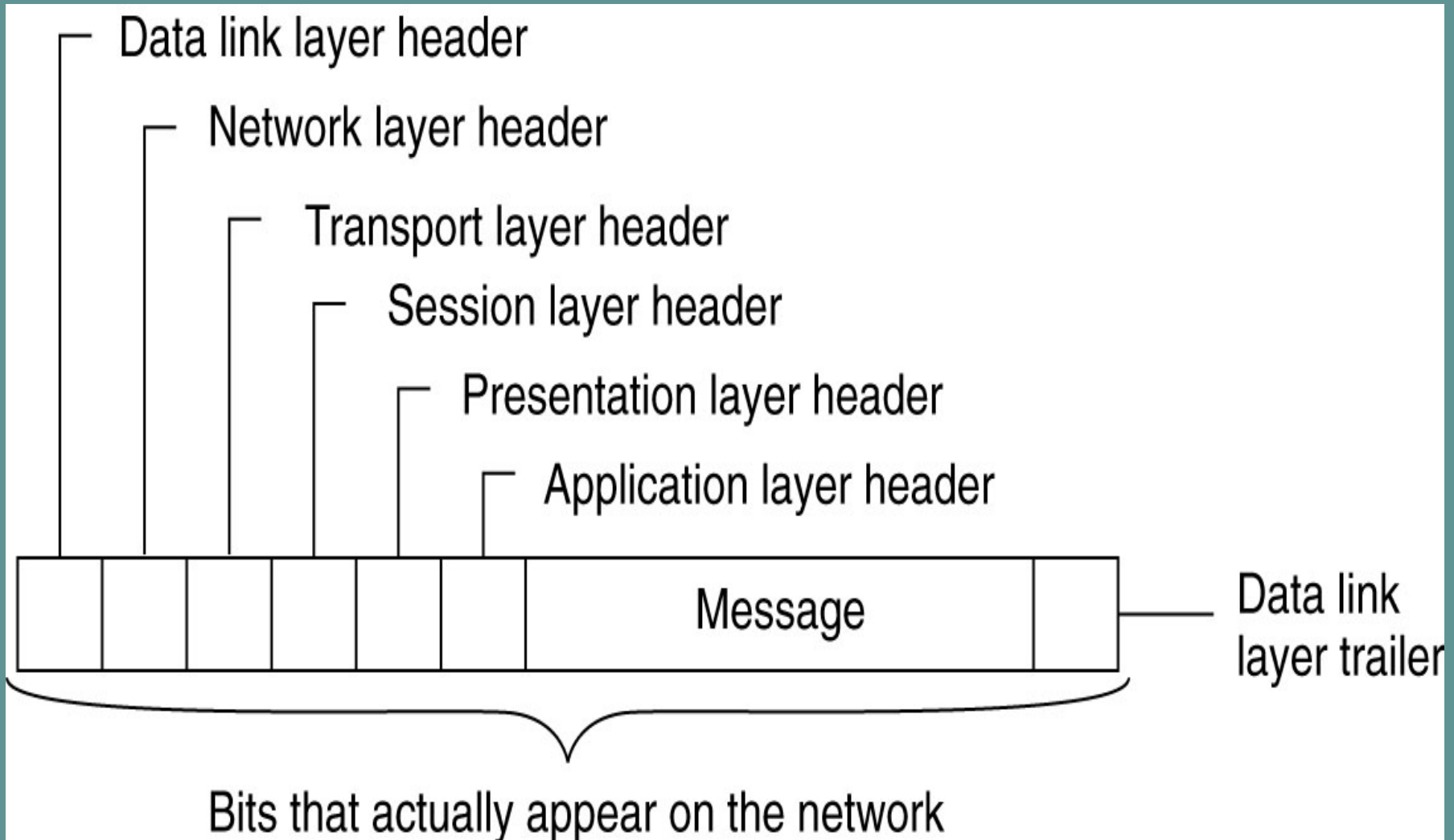


PROTOCOLOS DE COMUNICACIÓN

Hace uso de protocolos que definen la forma cómo debe desarrollarse la comunicación (formato, contenido y significado de los mensajes intercambiados).

- ♦ Protocolo de aplicación (ftp, telnet, correo electrónico)
- ♦ Protocolo de presentación (tipología del mensaje)
- ♦ Protocolo de sesión (facilidades de sincronización)
- ♦ Protocolo de transporte (TP0-TP4, TCP y UDP)
- ♦ Protocolo de red (X.25, IP, ATM)
- ♦ Protocolo de enlace de datos (checksum, control de errores)
- ♦ Protocolo físico (estándar RS-232, 802.3)

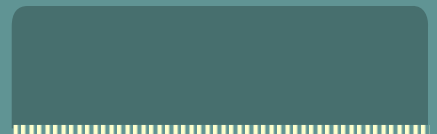
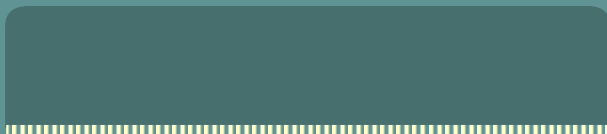
PROTOCOLOS DE COMUNICACIÓN: Componentes de un mensaje



PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

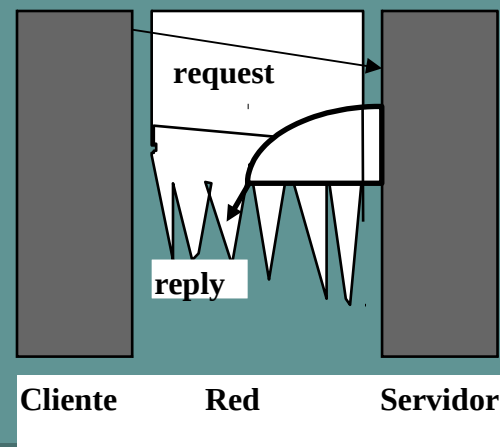
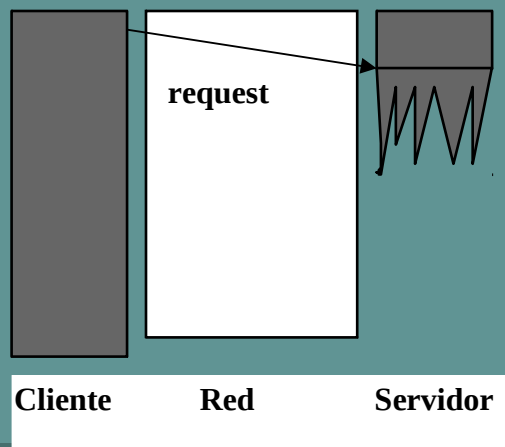
- ♦ Las redes pueden exhibir fallas que impliquen la pérdida de paquetes.
- ♦ Si sólo se pierden algunos paquetes (hay comunicación), estas fallas pueden corregirse usando “feedback” en forma de “ACK” y “timeouts”.

¿Qué implica que todos los paquetes se pierdan?



PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

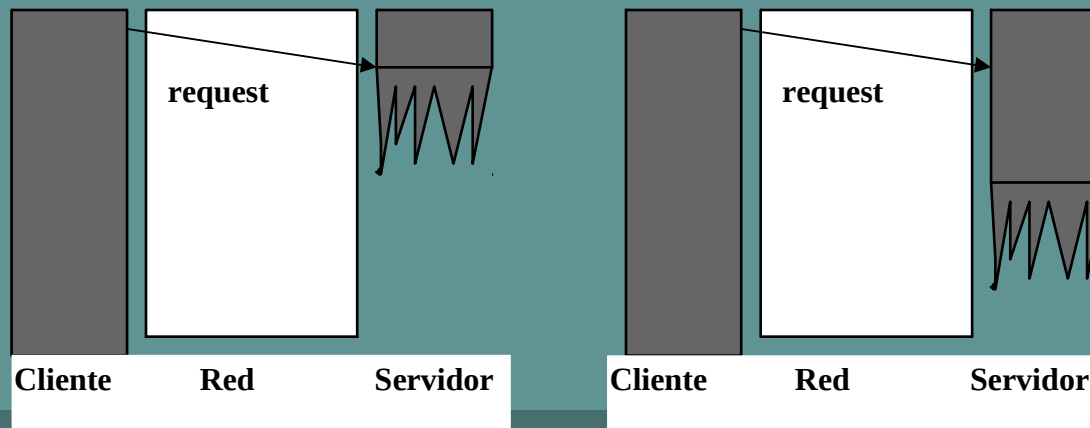
- ♦ Los protocolos no pueden ser totalmente confiables
- ♦ El problema es que el cliente no sabe si su requerimiento fue llevado a cabo o no



PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

- ♦ Si la red fuese totalmente confiable:

¿Cuál es la incertidumbre del cliente ante una falla del servidor?



PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

- ♦ Cuando el servidor revive luego de una caída, su cliente puede intentar una nueva comunicación, retransmitiendo el último “request” no respondido. ¿Qué sucede si el servidor ejecutó el requerimiento la vez anterior? Hay dos posibilidades:
 - ♦ El servidor recuerda lo que realizó antes de la caída, recalcula la respuesta y la envía al cliente.
 - ♦ El servidor sufre amnesia total. Olvida todo su estado, por lo tanto, conocer que es una retransmisión, no ayuda.

PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

- ♦ Ante el modelo de amnesia total, los protocolos de comunicación no pueden tener la propiedad de entregar los mensajes “exactamente una vez”.

Pueden tener propiedad de:

- ♦ Entregar mensaje al-menos-una-vez (at-least-once-protocols)
- ♦ Entregar mensajes a-lo-más-una-vez (at-most-once-protocols)

PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

Entrega al-menos-una-vez

- ♦ En ausencia de fallas, entregan los mensajes exactamente-una-vez.
- ♦ Ante fallas, pueden entregar un mensaje más de una vez.
- ♦ Se usa cuando los requerimientos son ídem potentes.
- ♦ Esconden fallas de comunicación y de servidores.
- ♦ Sun NFS usa este protocolo.

PROPIEDADES FUNDAMENTALES DE LOS PROTOCOLOS

Entrega a-lo-más-una-vez

- ♦ Detectan el hecho de que ha fallado la red o el servidor y lo reportan.
- ♦ Operan en un contexto *sesión* - una asociación entre dos procesos durante la cual ambos mantienen el estado del protocolo.
- ♦ Cuando se pierde el estado del protocolo, se termina la sesión.
- ♦ No se reciben mensajes en una sesión diferente a la sesión en la que se envió.
- ♦ Las sesiones tienen identificadores únicos (timestamps, números aleatorios).

TIPOS DE PROTOCOLOS DE TRANSPORTE

- ♦ La diversidad de necesidades de comunicación imposibilita el uso de un único protocolo de comunicación general. Se distinguen cuatro (4) clases:
 - ♦ Operaciones remotas (cliente-servidor o request-reply)
 - ♦ Protocolos de transferencia de datos en masa (Bulk Data Transfer)
 - ♦ Comunicación uno-a-muchos
 - ♦ Comunicación de medios continuos

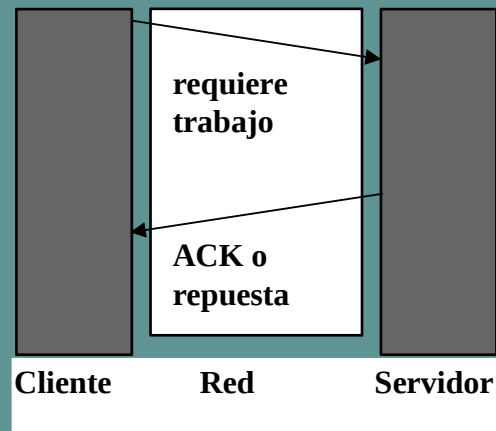
TIPOS DE PROTOCOLOS DE TRANSPORTE

Operaciones remotas:

- ♦ Forma más básica de comunicación en sistemas distribuidos.
- ♦ También llamada: cliente-servidor, comunicación request-response e invocación remota.
- ♦ RPC es un ejemplo de operación remota.

TIPOS DE PROTOCOLOS DE TRANSPORTE

Operaciones remotas:



TIPOS DE PROTOCOLOS DE COMUNICACIÓN

Protocolos de transferencia de datos en masa (Bulk Data Transfer):

- ♦ Se especializan en transferir grandes cuerpos de datos en forma eficiente.
- ♦ Usados como parte de protocolos de transferencia de archivos

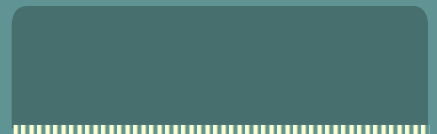
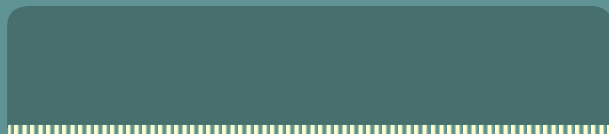
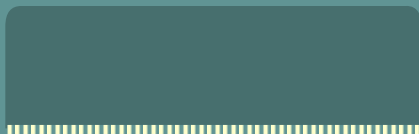
TIPOS DE PROTOCOLOS DE COMUNICACIÓN

Comunicación uno-a-muchos (comunicación en grupo):

- ♦ Usados en replicación.
- ♦ *Broadcast*: enviar a todos los host de una red.
- ♦ *Multicast*: enviar solo a un conjunto de host.

CONSTRUCCIÓN DE BLOQUES PARA LA COMUNICACIÓN

- ♦ Consiste en el “mapping” de estructuras de datos a “ítemes de datos” para mensajes.
- ♦ El problema surge porque no todos los computadores almacenan valores simples en el mismo orden. Así, para que dos computadores intercambien datos:



CONSTRUCCIÓN DE BLOQUES PARA LA COMUNICACIÓN

- ♦ Los valores son convertidos a una forma de representación externa de datos antes de la transmisión y convertida a la forma local al recibirlos.
- ♦ Para comunicación entre computadores del mismo tipo, la conversión a una representación externa de datos, puede ser omitida

CONSTRUCCIÓN DE BLOQUES PARA LA COMUNICACIÓN

- ♦ Cuando se usa comunicación orientada a conexión, los computadores pueden negociar si usan representación externa de datos.
- ♦ Una alternativa es transmitir los valores de datos en su forma nativa junto con un identificador de arquitectura y al recibirlos se convierten, si es necesario.

CONSTRUCCIÓN DE BLOQUES PARA LA COMUNICACIÓN

**Algunos tipos de representaciones
externas de datos son:**

- ♦ Sun XDR (eXternal Data Representation).
- ♦ Courier de Xerox.

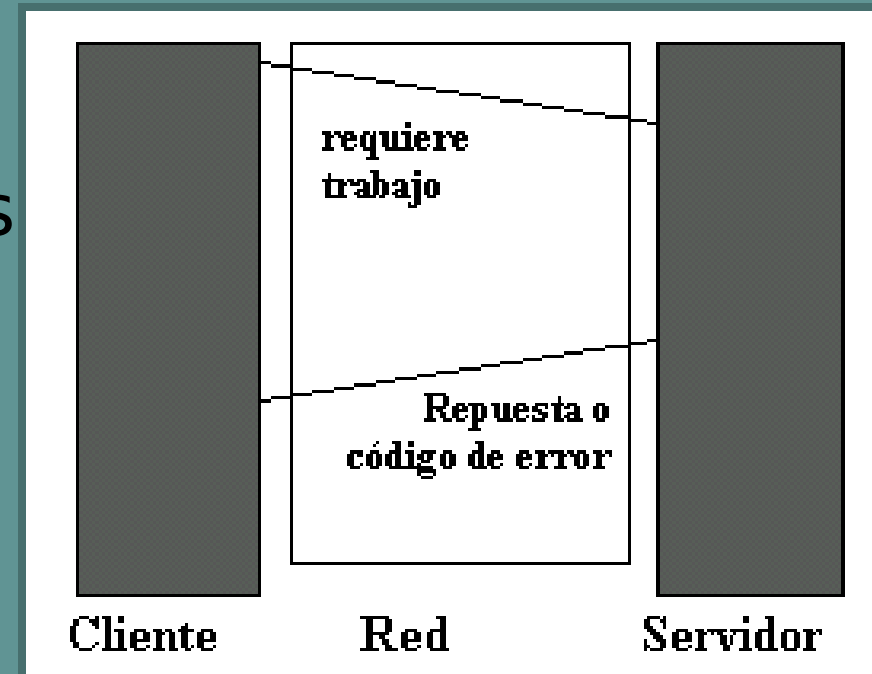
CONSTRUCCIÓN DE BLOQUES PARA LA COMUNICACIÓN

“Marshalling” es convertir ítemes de datos estructurados en una representación externa de datos, “unmarshalling” es el procedimiento inverso. Puede ser hecho:

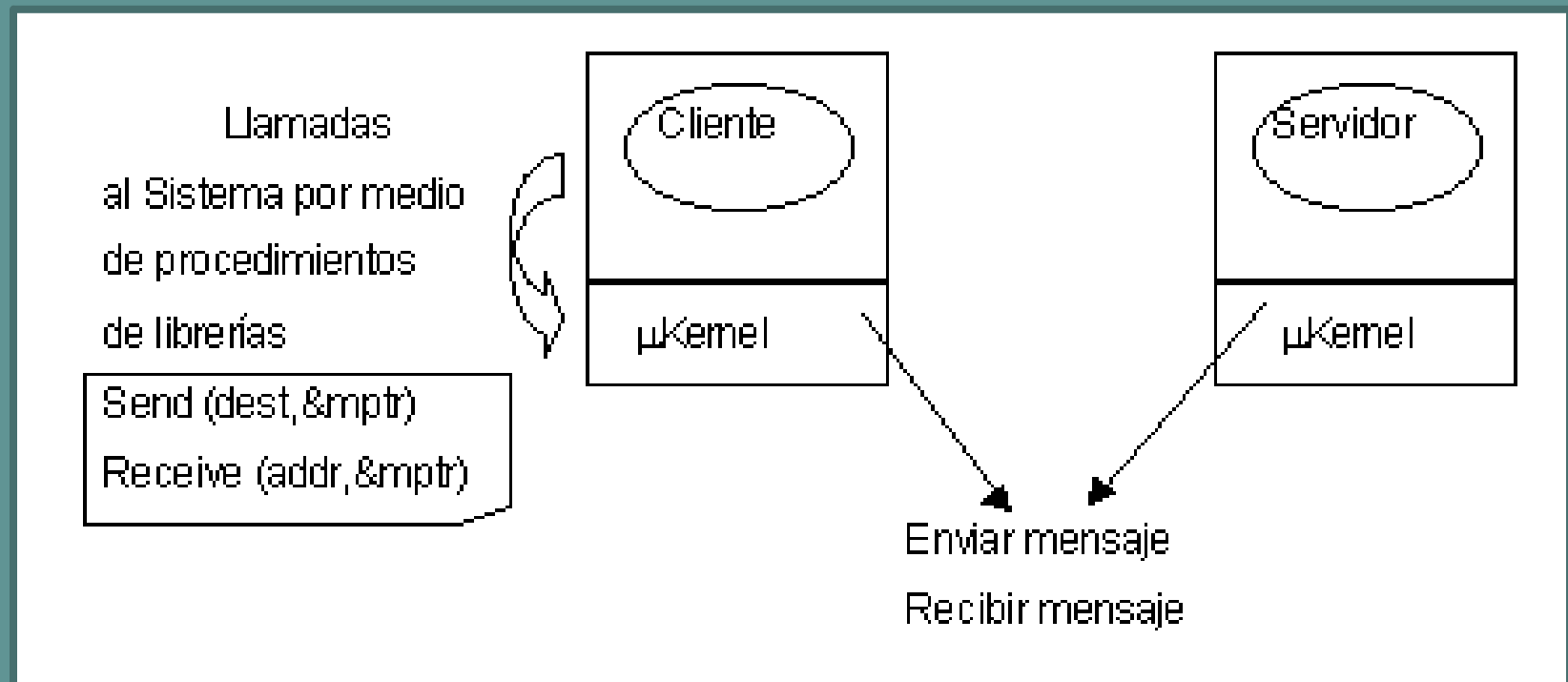
- ♦ A mano (by hand): El programa enviador realiza explícitamente la conversión.
- ♦ Automática: Especificación de los tipos de datos.

MODELO CLIENTE-SERVIDOR

- ◆ Sugiere estructurar al Sistema de Operación como un grupo de procesos cooperantes: Clientes y Servidores.
- ◆ Para evitar el costo excesivo de los protocolos orientados a la conexión, usualmente utiliza un protocolo “requerimiento-respuesta” sin conexión.

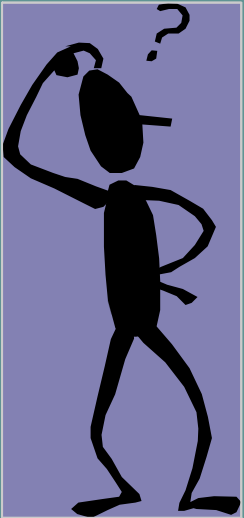


MODELO CLIENTE-SERVIDOR



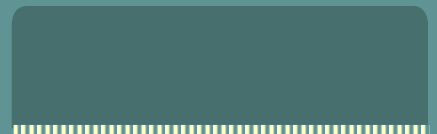
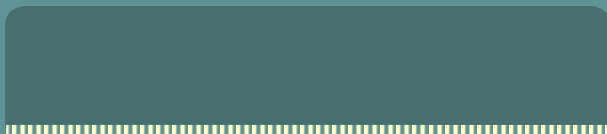
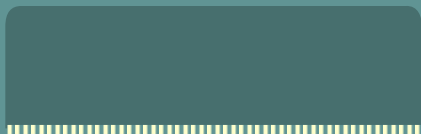
MODELO CLIENTE-SERVIDOR

- ◆ Direccionamiento



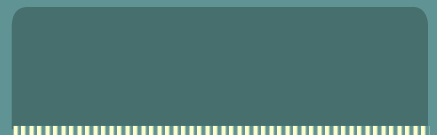
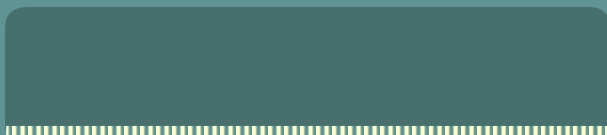
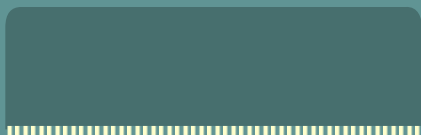
¿Cuál es la dirección del servidor?

Cliente



MODELO CLIENTE-SERVIDOR

- ♦ Direccionamiento: Para que un cliente envíe un mensaje a un servidor necesita conocer su dirección.
- ♦ Existen múltiples formas de hacer al cliente saber cuál es esta dirección, entre las que se encuentran:



MODELO CLIENTE-SERVIDOR

- Colocar la dirección IP de la máquina específica como una constante, dentro del programa emisor.

Problema: Sólo un proceso por máquina y no es transparente.

MODELO CLIENTE-SERVIDOR

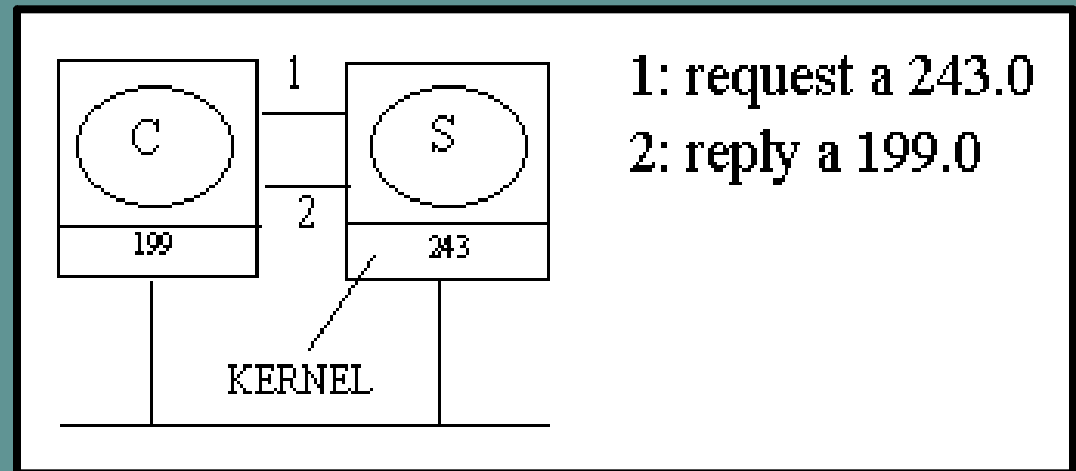
- Enviar mensajes a los procesos en lugar de enviarlo a máquinas específicas.
- ¿Cómo identificar los procesos?.

MODELO CLIENTE-SERVIDOR

(a) Nombres en dos partes:

- maq. id-proceso
- id-proceso @ maq
- maq. local-id

Número aleatorio
de 16 a 32 bits



◆ Problema: No hay transparencia.

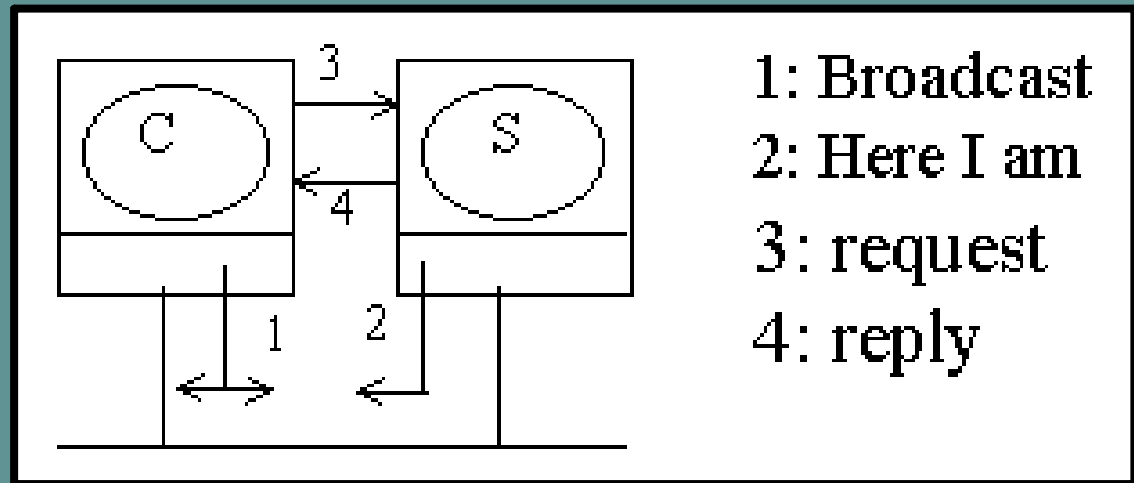
MODELO CLIENTE-SERVIDOR

- (b) A través de un proceso centralizado asignador de direcciones únicas a los procesos.
 - El núcleo del envidador localiza al proceso con un “broadcast”.
- ◆ Problema: Componente centralizado restringiendo la escalabilidad y sobrecarga del sistema por el “broadcast”.

MODELO CLIENTE-SERVIDOR

(c) Dejar que cada proceso tome su identificador como un número aleatorio (binarios enteros de 64 bits) y localizarlo a través de un broadcast.

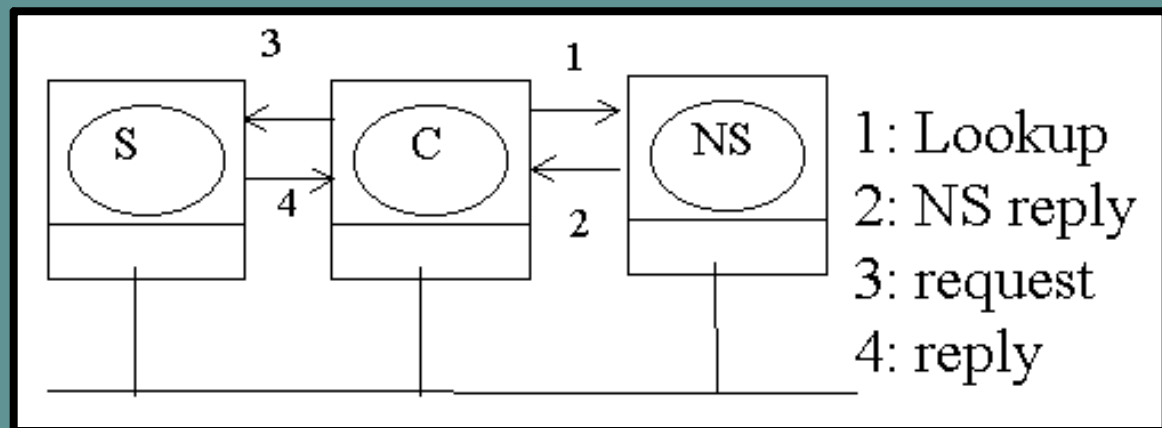
Se usa caching para disminuir el número broadcasts.



> Problema: carga extra al sistema por el “broadcast” y el caching.

MODELO CLIENTE-SERVIDOR

(d) Colocar nombres ASCII de los servidores en los clientes y un servidor de nombres en una máquina dedicada.

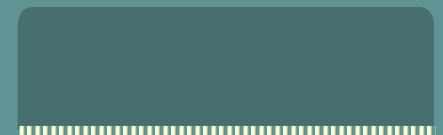
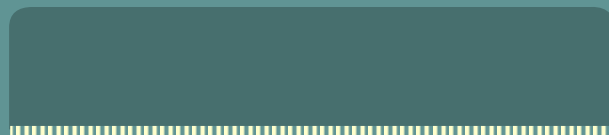
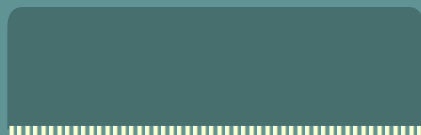


- ◆ Problema: Componente centralizado y no transparente.

MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

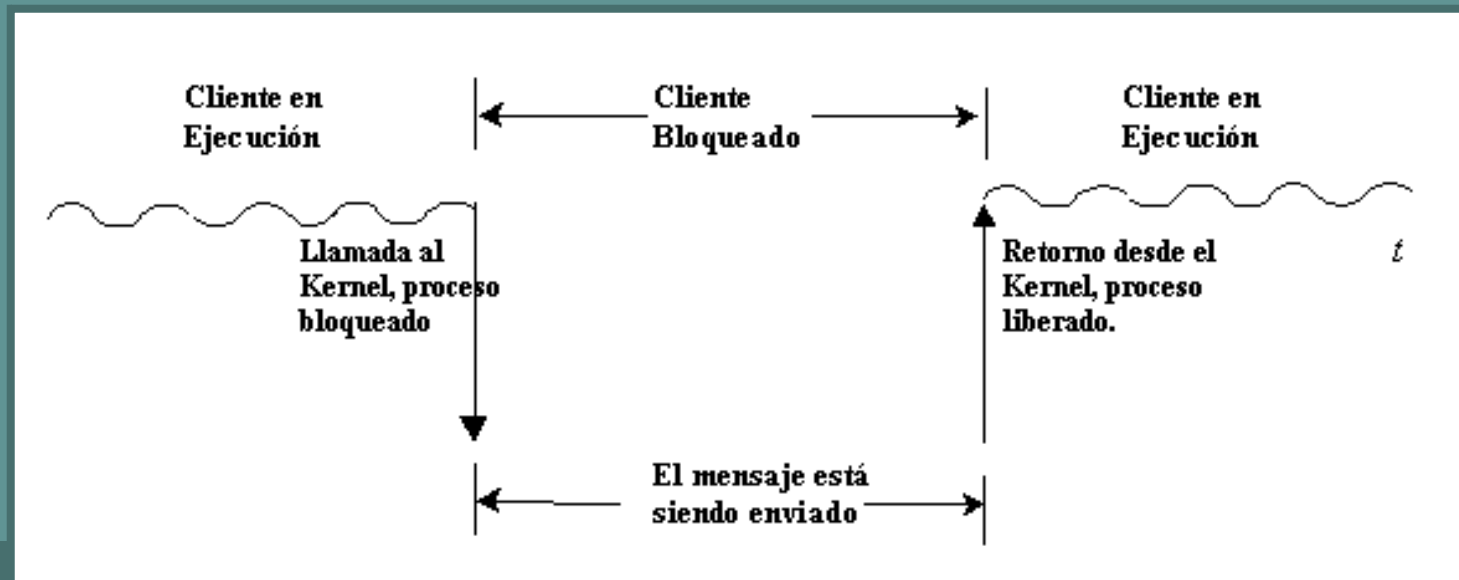
- ◆ Primitivas bloqueantes vs. no bloqueantes



MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

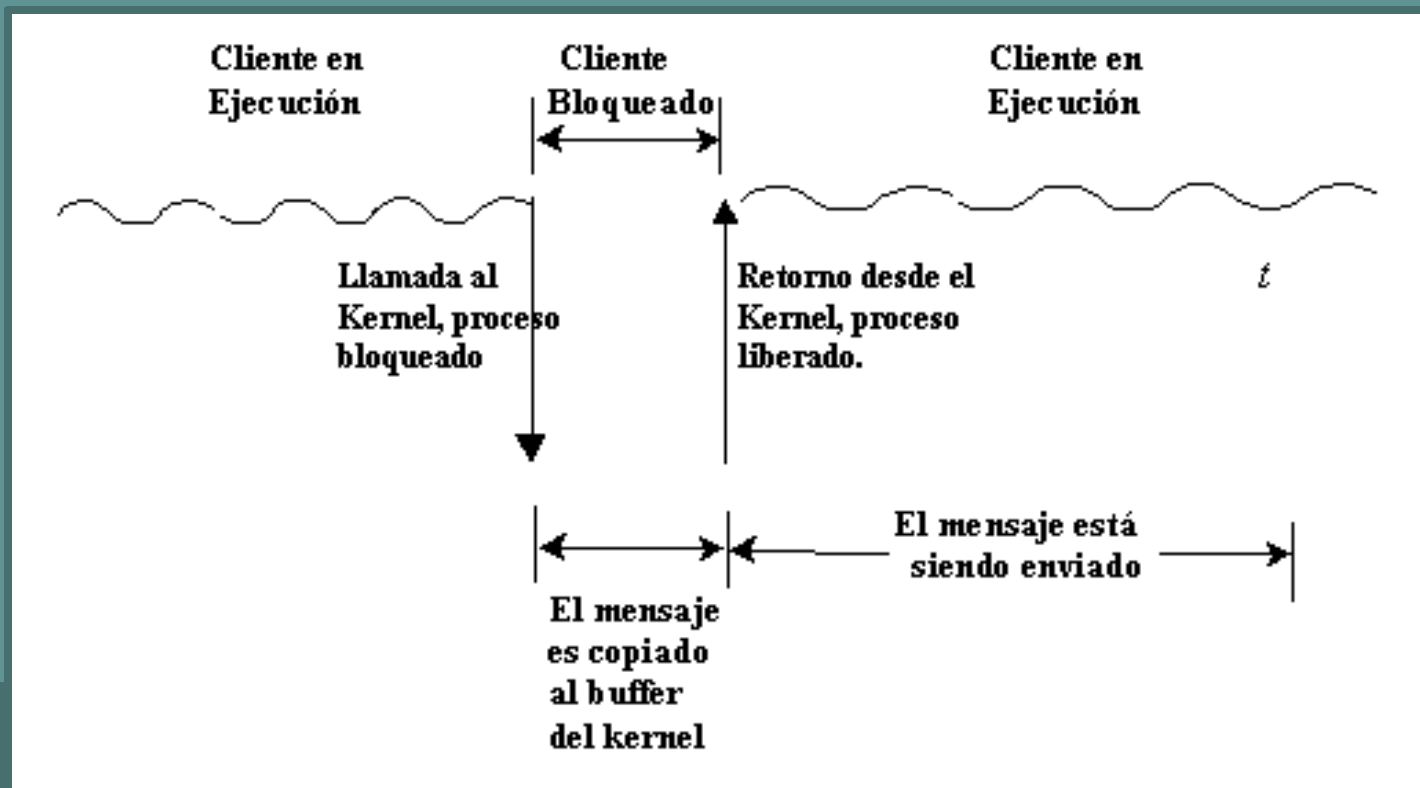
- ◆ Primitivas bloqueantes vs. no bloqueantes
 - ◆ SEND
 - ◆ Sincrónicos.



MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

- ◆ SEND
 - ◆ Asincrónicos.



MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

- Ventaja: Paralelismo en la ejecución del proceso enviador con la transmisión del mensaje.
- Desventaja cuando no hay buffer a nivel del kernel: El envidador no puede modificar su buffer del mensaje hasta que ha sido efectivamente enviado. Podría perderse el mensaje.

MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de

◆ Receive Comunicación

◆ No bloqueante:

- > Solo dice al kernel donde está el buffer de recepción y retorna el control al llamador.

¿ Cuándo sabe el cliente que la operación se ha completado?

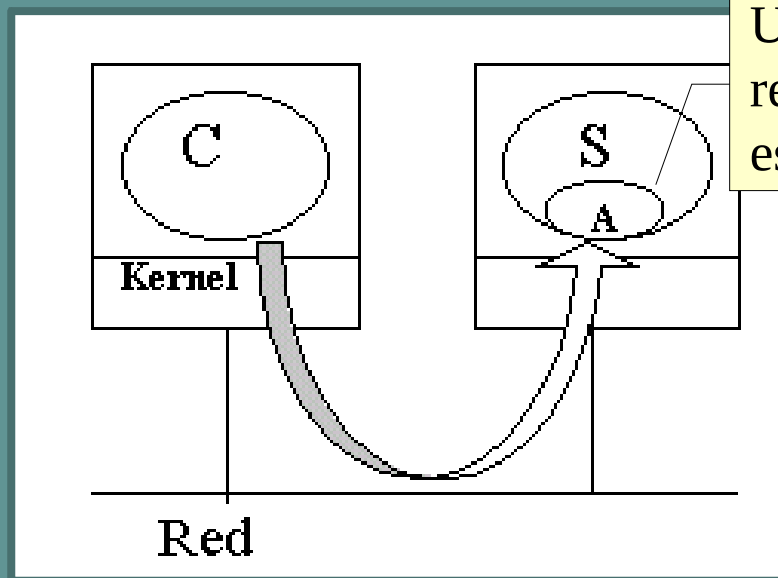
Soluciones:

- > wait (permite esperar hasta que llegue el mensaje)
- > test (para conocer el estado del buffer)
- > conditional-receive (espera el mensaje por un tiempo, pudiendo retornar el mensaje o una falla)
- > Interrupción.

MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

- ◆ Primitivas con “buffer” vs. sin “buffer”
 - ◆ Sin buffer (“Unbuffered”) :

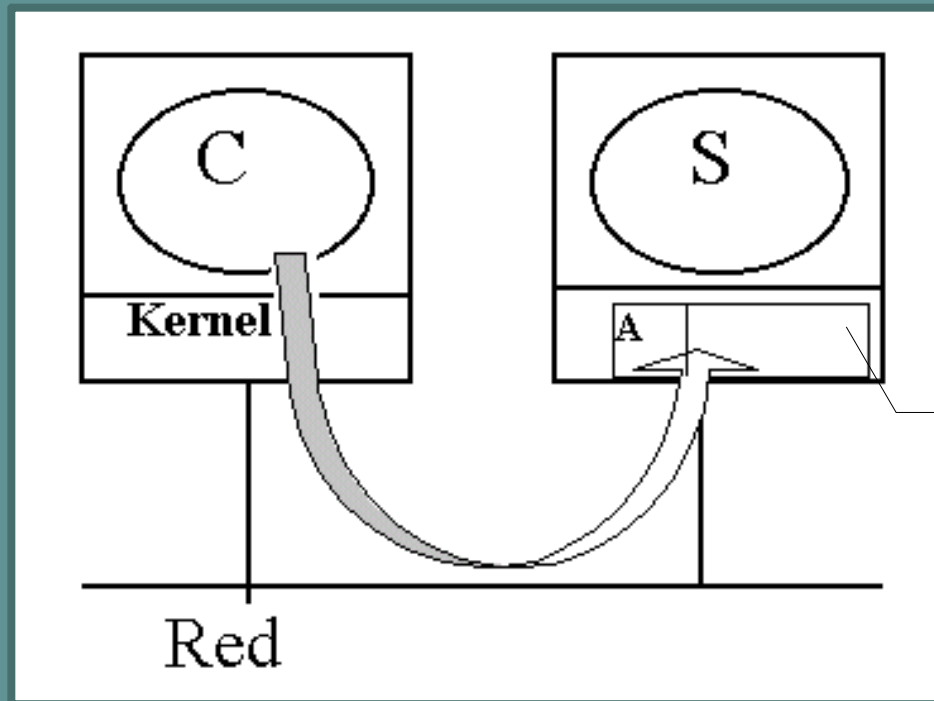


Una dirección se refiere a un proceso específico.

MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

- ♦ Con buffer (“Buffered”)



Buzón: espacio solicitado por un proceso al kernel para que le guarde los mensajes que lleguen.

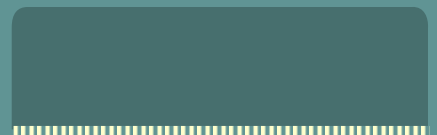
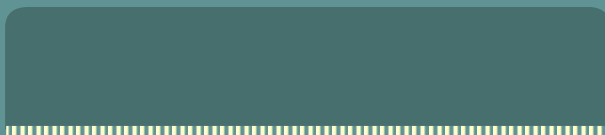
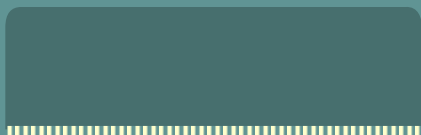
- ♦ Problema: Se puede llenar el buzón

⇒ El kernel, como antes, puede descartar el mensaje ó mantener el mensaje por un corto tiempo.

MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

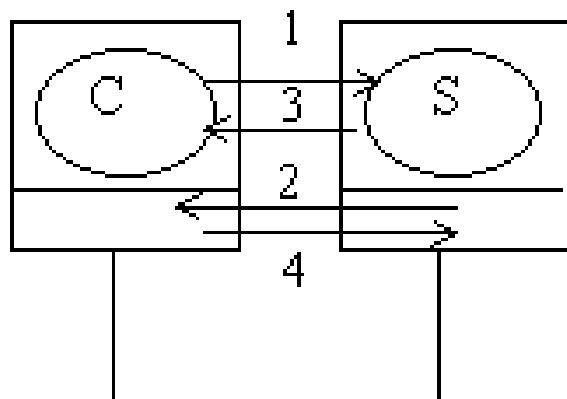
- ◆ Primitivas confiables vs. no confiables.
 - ◆ Se refiere a como resolver el problema de la pérdida de mensajes. El enviador debe asegurarse que el mensaje llegó correctamente a su destino.
- ¿Que garantía tiene un proceso de que el mensaje fue entregado?
- ◆ Existen cuatro (4) enfoques:



MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

1. *Send* no confiable: el sistema no garantiza que el mensaje fue entregado. La implementación de la comunicación confiable se deja en manos de los usuarios.
2. ACK a nivel de kernels emisor y receptor.

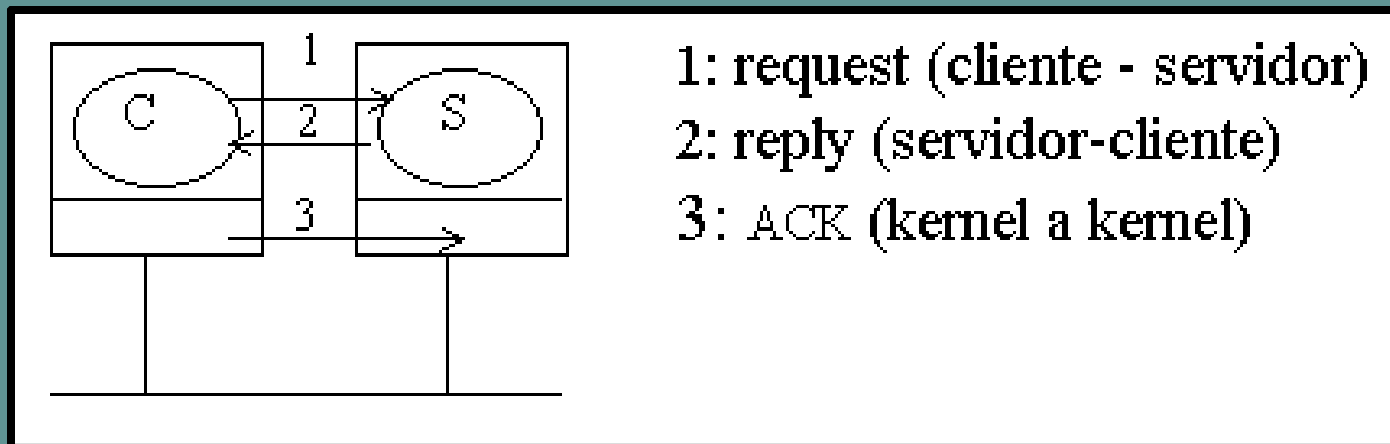


- 1: request (cliente - servidor)
- 2: ACK (kernel - kernel)
- 3: reply (servidor-cliente)
- 4: ACK (kernel a kernel)

MODELO CLIENTE-SERVIDOR:

Aspectos del Diseño de las Primitivas de Comunicación

3. Solo un ACK a nivel de kernels enviador y receptor.



4. Uso de timers para posponer los ACKs.

- ◆ Timers de retransmisión en el cliente.
- ◆ Timers de ACK.

RPC (REMOTE PROCEDURE CALLS)

- ♦ Mecanismo de comunicación basado en el protocolo request-reply.
- ♦ Consiste de un protocolo de comunicaciones, el cual típicamente se sitúa en el tope de un servicio de nivel de transporte, y rutinas de lenguaje concernientes con el ensamblaje de datos a ser pasados por el protocolo.
- ♦ Permitir que los programas llamen a procedimientos ubicados en otras máquinas en forma Transparente

RPC (REMOTE PROCEDURE CALLS)

- ♦ ¿Cómo funciona RPC?

La idea de RPC es que las llamadas remotas sean transparentes a las aplicaciones.

Para esto se requiere de STUBS de cliente y servidor contenidos en librerías, y procesos de empaquetamiento (marshall) y desempaquetamiento (unmarshall) de los parámetros pasados en las llamadas y en el resultado.

RPC (REMOTE PROCEDURE CALLS)


¿Cómo funciona la llamada convencional a procedimientos?

Count = read(fd, buf, nbytes).

fd: Integer.

buf: Array of char.

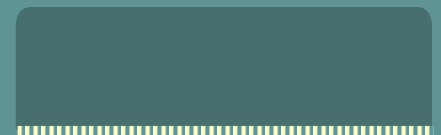
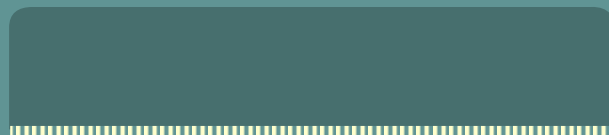
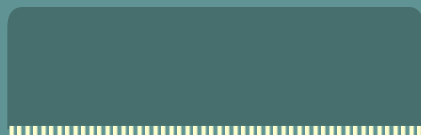
nbytes: Integer.



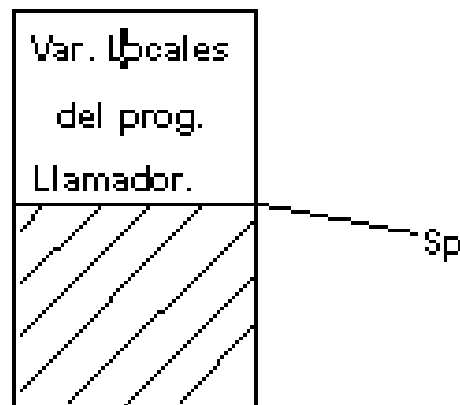
Identificación
del “Stream”
de Datos

RPC (REMOTE PROCEDURE CALLS)

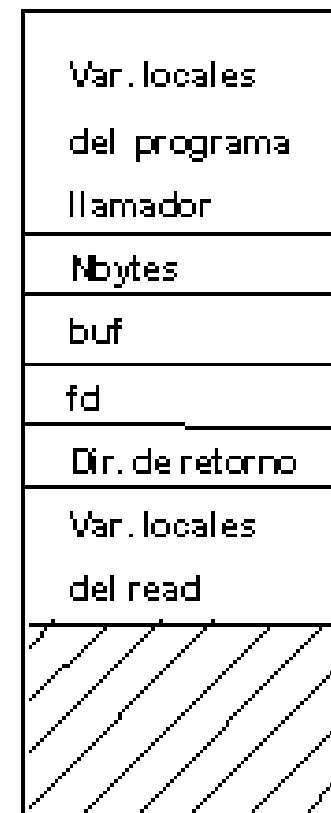
- ◆ ¿Cómo funciona la llamada convencional a procedimientos?
- ◆ El enlazador inserta el código de la rutina “read” al programa.
- ◆ Este código:
 - ◆ Coloca los parámetros en la pila del programa
 - ◆ Hace la llamada al núcleo, quien ejecuta la operación.



Etapa 1. Stack antes de la llamada



Etapa 2. Stack después de ejecutar la llamada.



Etapa 3. Después que el read finaliza.

La rutina llamada (read):

- Coloca el valor de retorno en un registro.
- Remueve la dirección de retorno y
- Retorna el control al llamador.

El prog. Llamador:

- Remueve los parámetros desde el stack.

RPC (REMOTE PROCEDURE CALLS)

El RPC:

- ◆ Debe lucir como una llamada local (Transparencia).

Usar "Stub" (rutina de librería)

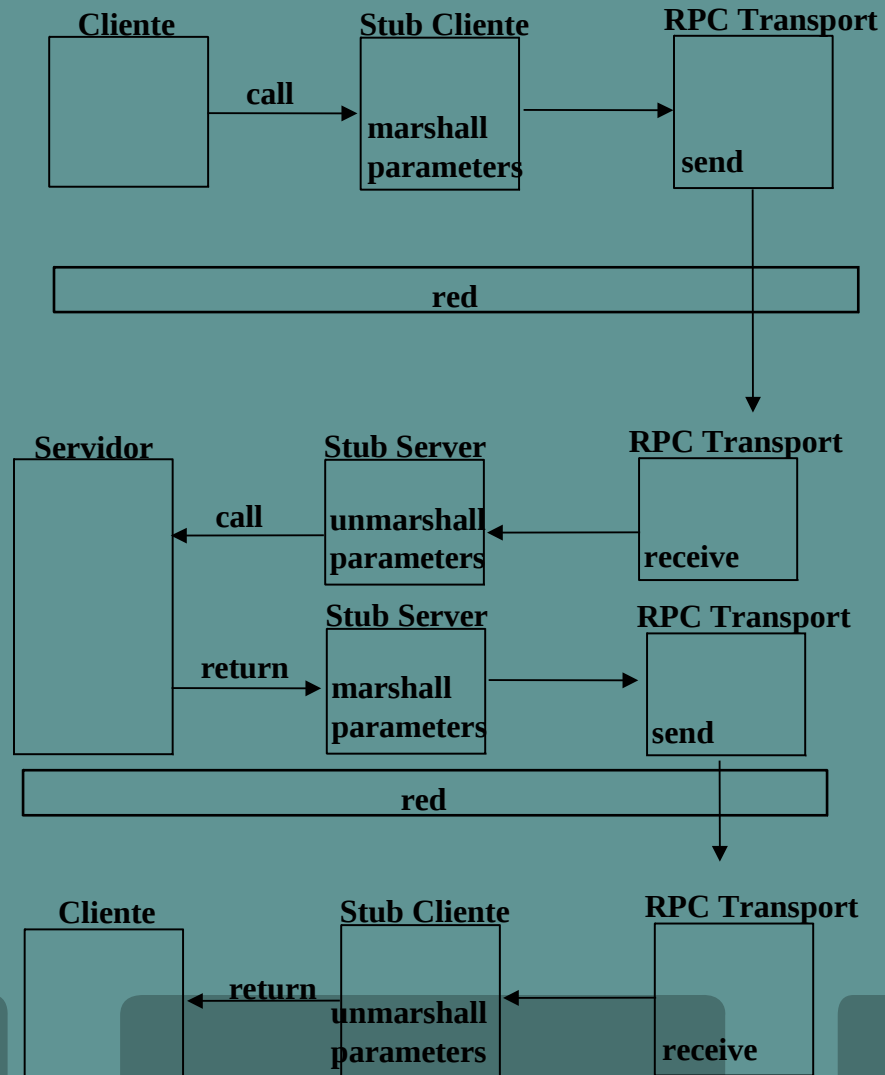
- ◆ El "stub" del cliente es enlazado al programa objeto cuando el procedimiento llamado es remoto.

RPC (REMOTE PROCEDURE CALLS)

¿Qué hace el stub del cliente que NO hace un procedimiento convencional?

- Copia los parámetros desde el stack a un mensaje de requerimientos;
- señala al núcleo pidiendo el envío del mensaje al servidor;
- ejecuta un receive, el cual lo bloquea hasta que reciba una respuesta.

RPC (REMOTE PROCEDURE CALLS)



RPC (REMOTE PROCEDURE CALLS)

Pase de parámetros en RPC:

- ♦ Heterogeneidad:

La existencia de arquitecturas diferentes en los sistemas distribuidos implica diferentes formatos de representación de números y caracteres:

- ♦ Complemento a 1 vs. Complemento a 2
- ♦ EBCDIC vs. ASCII
- ♦ Big endian vs. Little endian

RPC (REMOTE PROCEDURE CALLS)

Solución: Establecer un estándar de red o forma canónica para cada tipo de dato y requerir que los enviadores conviertan su representación interna a esta representación externa de datos. El proceso contrario en los receptores.

Pueden hacerse optimizaciones.

RPC (REMOTE PROCEDURE CALLS)

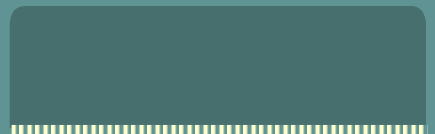
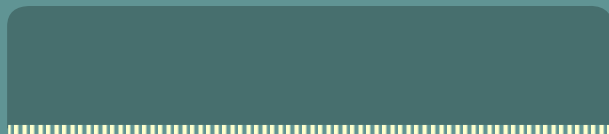
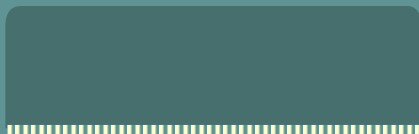
- ◆ Pase de parámetros por referencia:

Algunas soluciones son:

- ◆ Prohibir el pase de parámetros por referencias en RPC.
- ◆ Copiar los datos reales en el mensaje (call by copy-restore)

RPC (REMOTE PROCEDURE CALLS)

- ◆ Pase de apuntadores a estructuras complejas (grafos, árboles):
 - ◆ Pasar la estructura completa.
 - ◆ Ir pasando los datos a medida que el servidor lo requiera.
- ◆ Pase de parámetros de tipos definidos por el usuario:
 - ◆ Dividir sucesivamente las estructuras hasta alcanzar tipos de datos básicos.

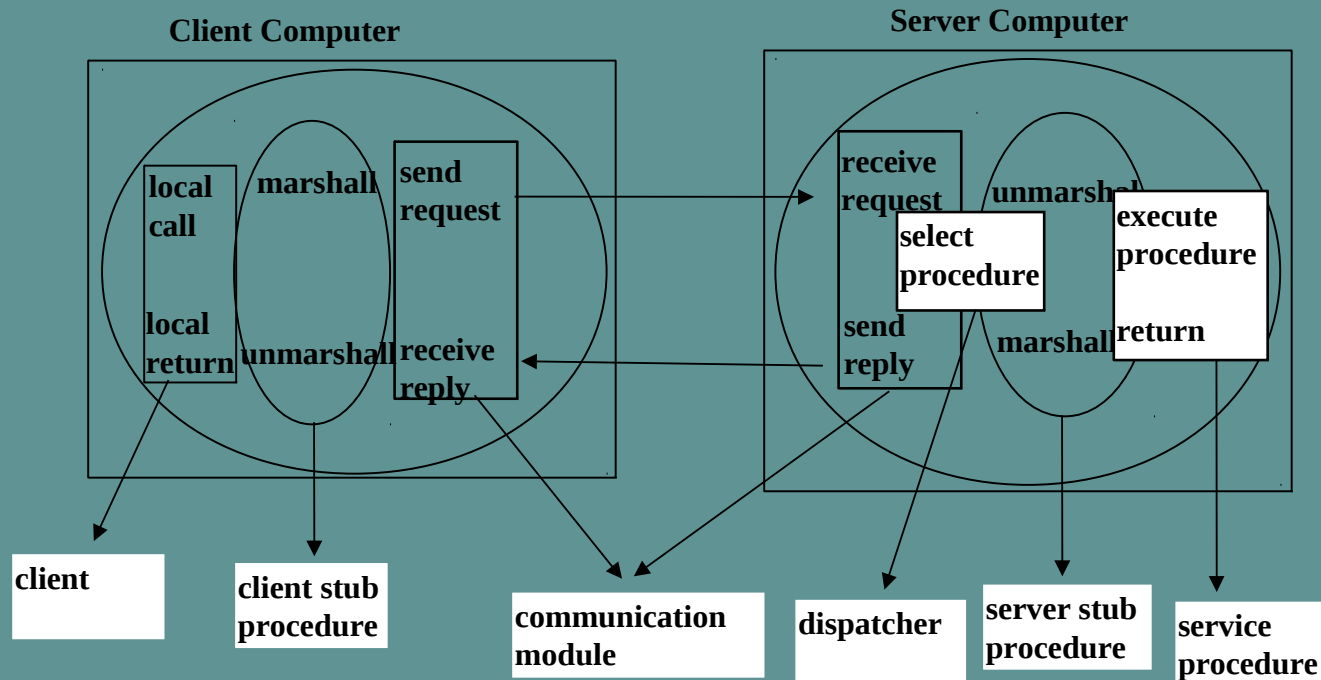


RPC (REMOTE PROCEDURE CALLS)

El software que soporta RPC tiene tres principales tareas:

- ♦ Procesamiento de la interfaz: Integrar el mecanismo RPC con cliente y servidor en un lenguaje de programación convencional.
- ♦ Manejo de comunicaciones: Bajo el protocolo request-reply.
- ♦ “Binding”: Localizar el servidor apropiado para un servicio particular.

RPC (REMOTE PROCEDURE CALLS)



RPC (REMOTE PROCEDURE CALLS)

Procesamiento de la Interfaz

- ♦ Existen lenguajes especializados para definir las interfaces para RPC y proveer las especificaciones necesarias de los servicios:
“Interface Definition Language” (IDL).
- ♦ Algunos ejemplos de IDL: Sun RPC, Courier de Xerox, AIL.

RPC (REMOTE PROCEDURE CALLS)

- ♦ La información que se requiere define las características de los procedimientos provistos por un servidor:
 - ♦ Dirección en la cual los parámetros viajan (in, out).
 - ♦ Los tipos y los tamaños de los parámetros.
 - ♦ Nombre y versión del servidor.
 - ♦ Lista de procedimientos provistos por el servidor.

RPC (REMOTE PROCEDURE CALLS)

Ejemplo de una especificación para un servidor de archivos:

```
# include <header.h>
```

specification of file_server, version 3.1:

```
long read(in char name[MAX_PATH], out char buf[BUF_SIZE],  
          in long bytes, in long position);
```

```
long write(in char name[MAX_PATH], in char buf[BUF_SIZE],  
           in long bytes, in long position);
```

```
long create(in char name[MAX_PATH], in int mode);
```

```
long delete(in char name[MAX_PATH]);
```

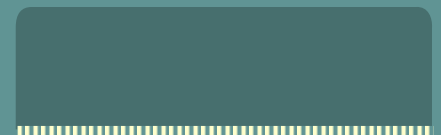
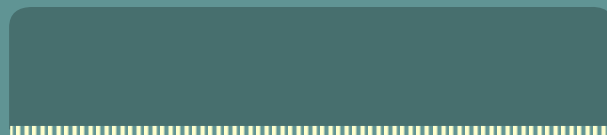
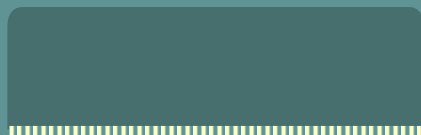
```
end;
```

RPC (REMOTE PROCEDURE CALLS)

Las especificaciones antes descritas son tomadas por el compilador de interfaces cuyas funciones son:

- ♦ Generar un procedimiento STUB del cliente para corresponder a cada procedimiento de la interfaz.
- ♦ Generar un proceso STUB del servidor.
- ♦ Generar las operaciones “marshalling” y “unmarshalling” en cada procedimiento STUB.
- ♦ Generar los procedimientos de servicio de la definición de la interfaz. El programador del servicio provee los cuerpos de esos procedimientos.

¿Puede crearse interfaz entre clientes y servidores escritos en lenguajes diferentes?



RPC (REMOTE PROCEDURE CALLS)

Binding:

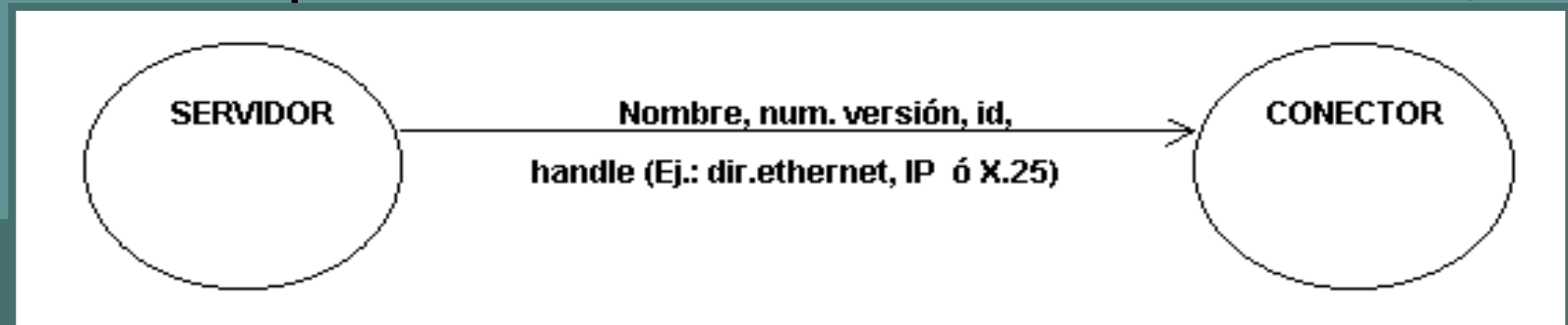
- ♦ Especifica cómo se establece la relación entre un procedimiento remoto y el programa que lo llama.
- ♦ Un “binding” se forma cuando dos aplicaciones han hecho una conexión lógica y están preparadas para intercambiar comandos y datos.

RPC

Binding:

Pasos para la conexión dinámica:

- a.- Especificación formal del Servidor: Sirve como entrada para el generador de "stubs" (cliente-servidores).
- b.- Proceso de registro del servidor: Cuando el servidor inicia su ejecución, exporta su interfaz (envía un mensaje a un programa llamado "conector" para darle a conocer su existencia).



RPC

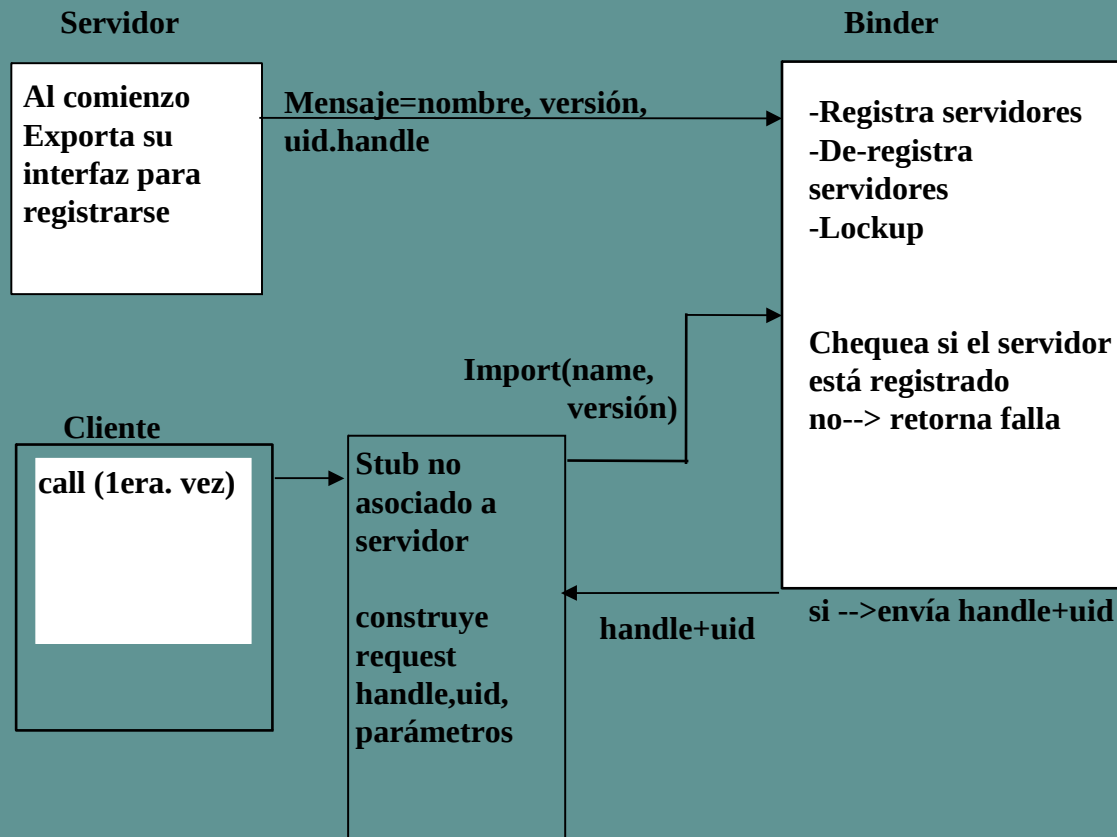
Binding:

Pasos para la conexión dinámica:

c.- Proceso de Conexión (lookup).

d.- Cancelación de un registro = Deregister
(Nombre, versión, handle, id).

RPC (REMOTE PROCEDURE CALLS)



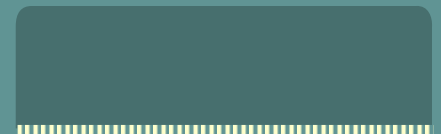
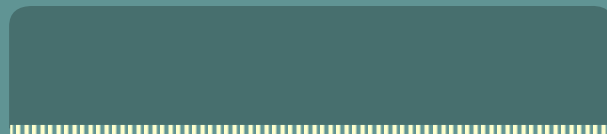
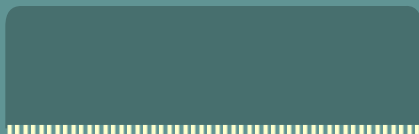
RPC (REMOTE PROCEDURE CALLS)

Ventajas de un “Binder”:

- ♦ Es muy flexible:
 - ♦ Puede ayudar a la autenticación.
 - ♦ Puede manejar servidores múltiples con la misma interfaz.
 - ♦ Puede asignar servidores a clientes considerando la carga de los servidores.
 - ♦ Puede dar soporte a la tolerancia a fallas.

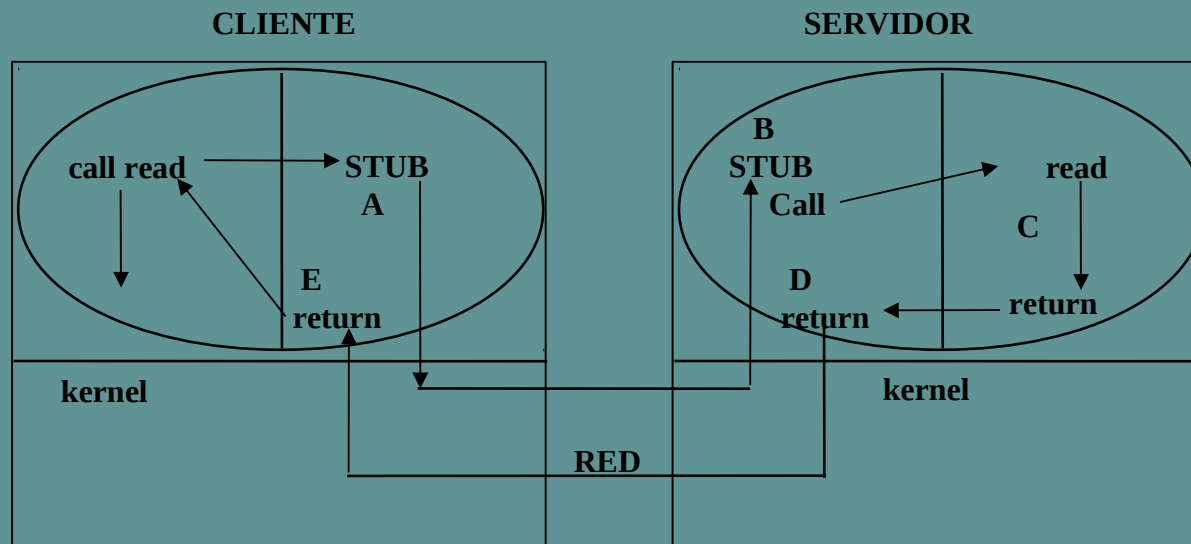
Problemas:

- ♦ Overhead en el import y export de interfaces.
- ♦ Cuello de botella (overhead de replicación).
- ♦ El binder es un servidor ¿cómo localizarlo a el?.



RPC (REMOTE PROCEDURE CALLS)

Semántica RPC ante fallas:



RPC (REMOTE PROCEDURE CALLS)

- ◆ Cliente no puede localizar el servidor:
Servidor está caído - nuevas versiones
 - ◆ Soluciones:
 - ◆ Retornar -1.
 - ◆ Producir una excepción.
- ◆ Mensaje request se pierde:
Expira el timer en A, antes que llegue el reply o ACK.
 - ◆ El kernel retransmite varias veces (at-least-once).
 - ◆ Uso de RPC id, sólo para operaciones ídem potentes.
 - ◆ El kernel reporta el error a la aplicación (at-most-once).

RPC (REMOTE PROCEDURE CALLS)

- ♦ Mensaje reply se pierde:

Expira el timer en D y no se recibe el ACK.

- ♦ Se retransmite el reply (puede recibirse de nuevo el request, con el id se reconoce la retransmisión).

Problema: ¿Se perdió el mensaje realmente?
ó ¿Es lento el servidor? → Puede acarrear problemas cuando la operación no es idempotente (transacción bancaria / Promedio de notas)

Solución: Enumerar los mensajes.

RPC (REMOTE PROCEDURE CALLS)

**“Timeout”
en el cliente**

- ♦ El servidor se cae:
 - ♦ El servidor puede fallar en los puntos:
 1. B: Después de que el STUB recibe la petición, pero antes de hacer la llamada al servicio específico.
 2. C: Durante la ejecución del servicio.
 3. D: Después de ejecutado el servicio, pero antes de enviar la respuesta.

RPC (REMOTE PROCEDURE CALLS)

- ◆ En todos los casos el cliente retransmite el request.
 - ◆ En el primer caso no hay problema.
 - ◆ En el caso 3, solo reenvía el reply.
 - ◆ En el caso 2, habrá problema si no se maneja “rolling back”.
- ◆ El cliente se cae:
 - ◆ Cuando se generan las respuestas no habrá clientes esperándolas → cálculos huérfanos (generan desperdicios de ciclos de CPU).

RPC (REMOTE PROCEDURE CALLS)

- ♦ Se deben eliminar los huérfanos utilizando alguno de los siguientes métodos:
 - ♦ Exterminación: Se elimina explícitamente el huérfano al momento de reiniciarse. Se requiere (log) que el cliente registre cada RPC que va a realizar.
 - ♦ Reencarnación: Se trabaja con épocas. Cuando un cliente se recupera, comienza una nueva época y con un broadcast la da a conocer. Así, todos los cálculos remotos obsoletos son eliminados.

RPC (REMOTE PROCEDURE CALLS)

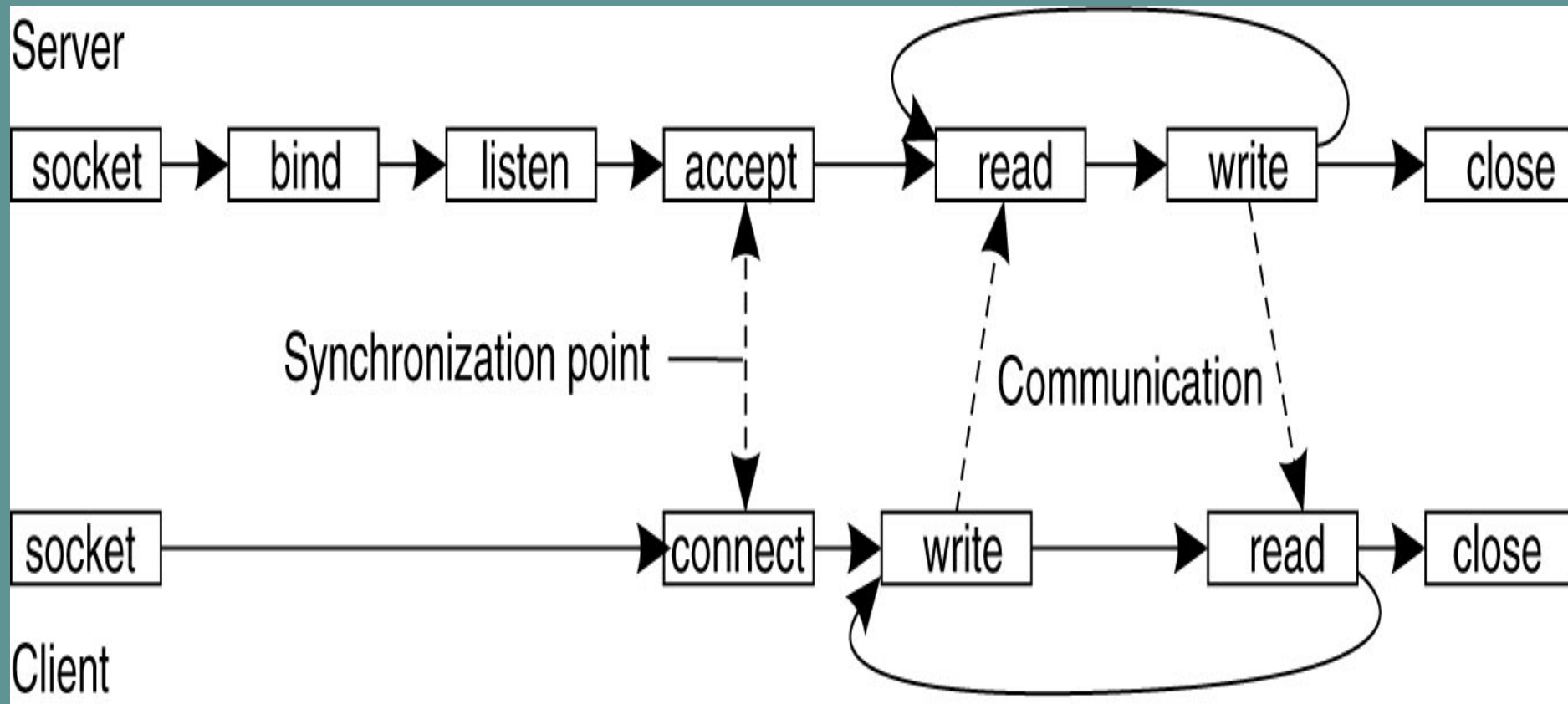
- ♦ Reencarnación sutil: Cuando llega el broadcast de una nueva época, cada máquina chequea si tiene cálculos remotos. Si es así, trata de localizar el propietario, si no lo localiza, se matan los cálculos.
- ♦ Expiración: Se le asigna un tiempo T a cada RPC para su ejecución y si no termina debe solicitar tiempo extra.

Interfaces de Comunicación

◆ Librería de sockets

| Primitive | Meaning |
|------------------|---|
| Socket | Create a new communication end point |
| Bind | Attach a local address to a socket |
| Listen | Announce willingness to accept connections |
| Accept | Block caller until a connection request arrives |
| Connect | Actively attempt to establish a connection |
| Send | Send some data over the connection |
| Receive | Receive some data over the connection |
| Close | Release the connection |

Interfaces de Comunicación:sockets



Interfaces de Comunicación: MPI

| Primitive | Meaning |
|------------------|---|
| MPI_bsend | Append outgoing message to a local send buffer |
| MPI_send | Send a message and wait until copied to local or remote buffer |
| MPI_ssend | Send a message and wait until receipt starts |
| MPI_sendrecv | Send a message and wait for reply |
| MPI_isead | Pass reference to outgoing message, and continue |
| MPI_issend | Pass reference to outgoing message, and wait until receipt starts |
| MPI_recv | Receive a message; block if there is none |
| MPI_irecv | Check if there is an incoming message, but do not block |

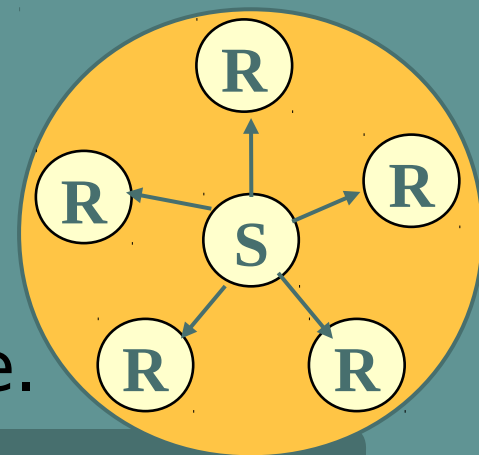
COMUNICACIÓN EN GRUPO

Mecanismo de comunicación alternativo, en el cual un mensaje puede ser enviado a múltiples receptores en una sola operación.

Grupo: colección de procesos que actúan juntos en algún sistema o forma especificada por el usuario.

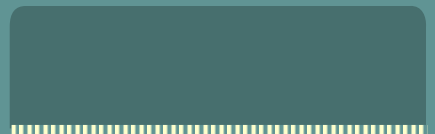
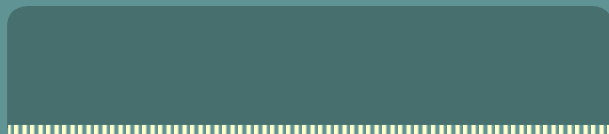
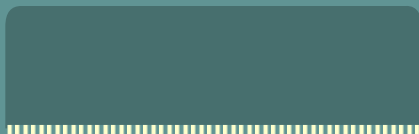
Los grupos son dinámicos.

La implementación de la comunicación en grupo depende mucho del hardware.



COMUNICACIÓN EN GRUPO

- ♦ Se requieren mecanismos para administrar grupos y miembros de grupos.
 - ♦ Multicasting: usando direcciones de red especiales a las cuales múltiples máquinas pueden escuchar.
 - ♦ Broadcasting: Los paquetes que contienen cierta dirección (0, por ejemplo) son entregados a todas la máquinas.
 - ♦ Transmitir los paquetes separados.



COMUNICACIÓN EN GRUPO

- ♦ La comunicación en grupo es útil para construir sistemas con las siguientes características:
 - ♦ Tolerantes a fallas basados en servidores replicados.
 - ♦ Localización de objetos en servicios distribuidos.
 - ♦ Mejor desempeño a través de datos replicados.
 - ♦ Actualización múltiple.

COMUNICACIÓN EN GRUPO

Aspectos de Diseño

- ◆ Grupos cerrados vs. Grupos abiertos.
- ◆ Grupos jerárquicos vs. Grupos de “amigos”.
- ◆ Miembros del grupo.
- ◆ Direccionamiento de grupos.
- ◆ Primitivas send y receive.
- ◆ Atomicidad.
- ◆ Ordenamiento de mensajes.
- ◆ Solapamiento de grupos.

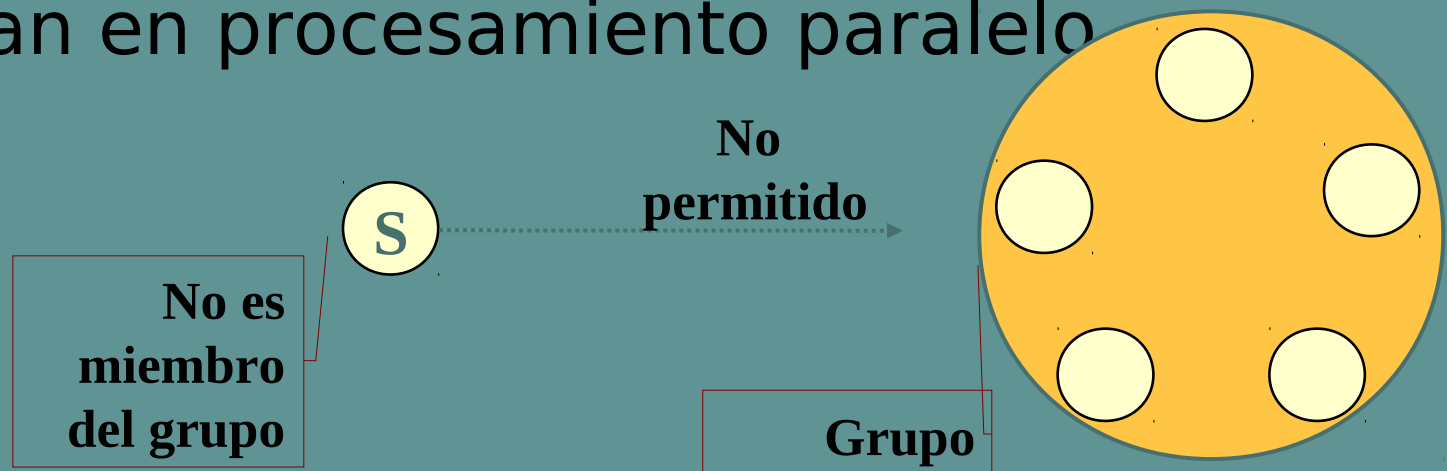
COMUNICACIÓN EN GRUPO

Grupos cerrados vs. Grupos abiertos

¿Quién puede enviar a quiénes?

- ♦ **Grupos cerrados:**

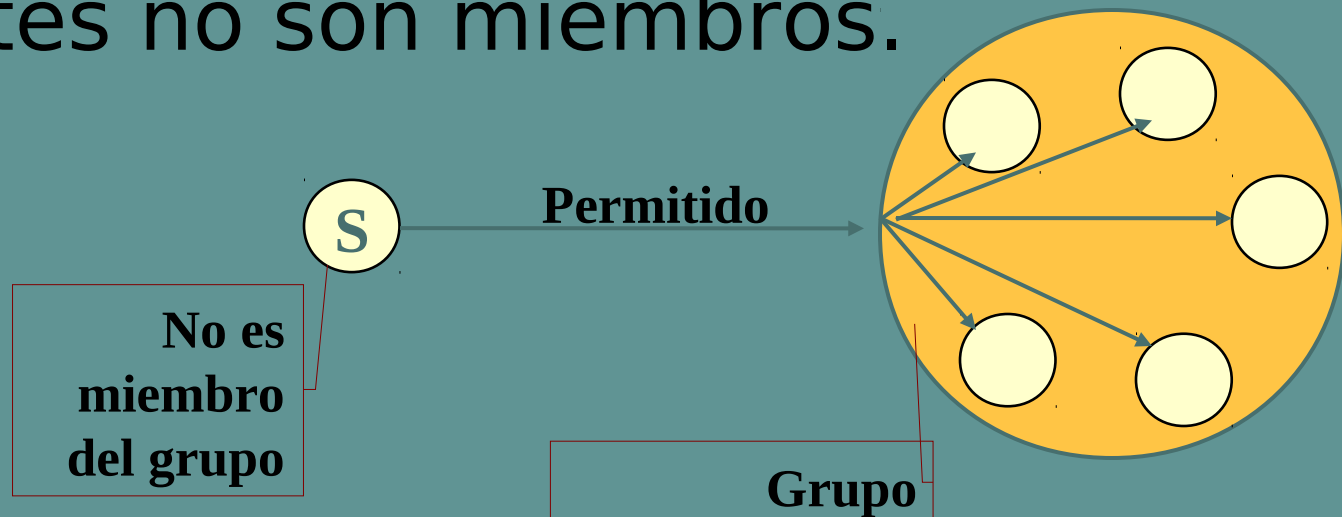
- ♦ sólo los miembros del grupo pueden enviar al grupo.
- ♦ Se usan en procesamiento paralelo



COMUNICACIÓN EN GRUPO

- ♦ **Grupos abiertos:** cualquier proceso en el sistema puede enviar a cualquier grupo.

Ejemplo: grupo de servidores y los clientes no son miembros.

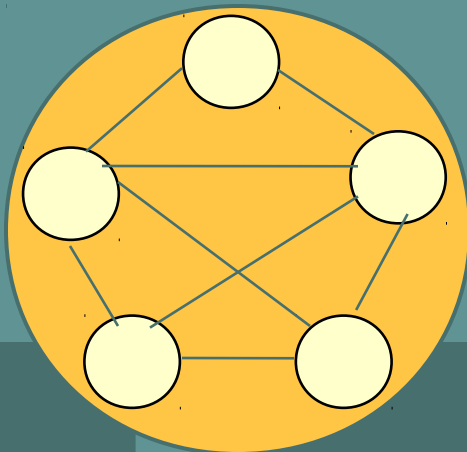


COMUNICACIÓN EN GRUPO

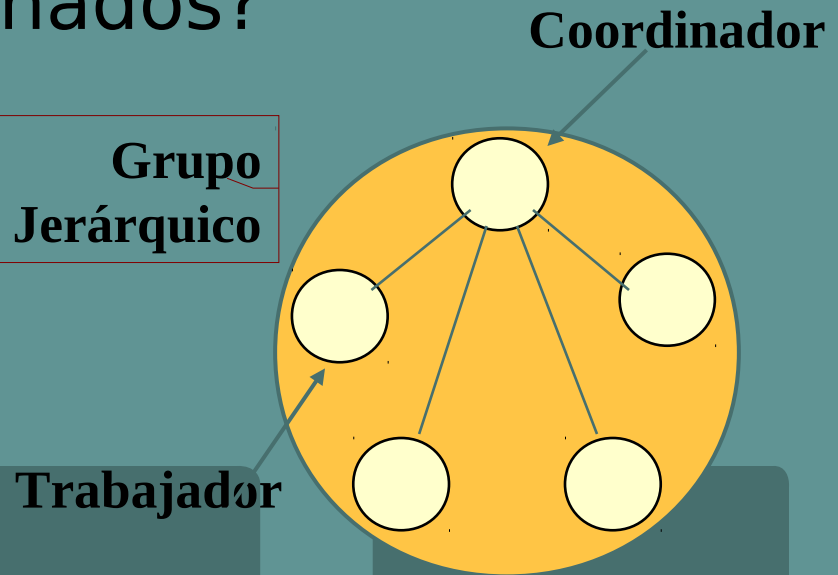
Grupos jerárquicos vs. Grupos de “amigos”

- ♦ Define la estructura interna del grupo
- ♦ Al momento de tomar una decisión, ¿todos los procesos son iguales ó hay coordinadores-subordinados?

Grupo de
Compañeros



Grupo
Jerárquico



COMUNICACIÓN EN GRUPO

- ♦ Grupos de compañeros (“amigos”):
 - ♦ Simétrico
 - ♦ Toma de decisión compleja
 - ♦ No hay un único punto de falla
 - ♦ Hay n puntos de falla
- ♦ Grupos Jerárquicos:
 - ♦ Asimétrico
 - ♦ Toma de decisión sencilla
 - ♦ Hay un único punto de fallo => Coordinador

COMUNICACIÓN EN GRUPO

Miembros del grupo

- ♦ Se requiere de operaciones para crear y borrar grupo, permitir nuevos miembros, permitir abandonar un grupo.
 - ♦ Servidor de grupo.
 - ♦ Administrar los miembros de un grupo de forma distribuida.
 - ♦ Nuevo miembro: envía un mensaje a todos los miembros del grupo para indicar su presencia.
 - ♦ Un miembro abandona el grupo: envía un mensaje “adiós al grupo.

Desventaja: Si un miembro se cae no hay forma de comunicar este abandono. Se debe descubrir experimentalmente.

COMUNICACIÓN EN GRUPO

- ♦ Se requiere sincronizar, con el abandono e integración los mensajes enviados.
- ♦ ¿Que hacer con los miembros, si muchas máquinas se caen y el grupo no puede funcionar?.
 - ♦ Se requiere un protocolo para reconstruir el grupo.
 - ♦ ¿Quién toma la iniciativa?. El protocolo debe soportarlo.

COMUNICACIÓN EN GRUPO

Direccionamiento de grupos:

- ♦ Dar a cada grupo una dirección única: Puede ser implementado con multicast, broadcast o una lista de máquinas (en el kernel) que contienen los procesos pertenecientes al grupo.
- ♦ Requerir que el enviador provea una lista explícita de todos los destinos (direcciones IP). No es transparente.

COMUNICACIÓN EN GRUPO

- **Direccionamiento por predicado:** El mensaje es enviado a todos los miembros del sistema (o del grupo). El mensaje contiene una expresión booleana a ser evaluada. Si evalúa true, el mensaje es aceptado.

COMUNICACIÓN EN GRUPO

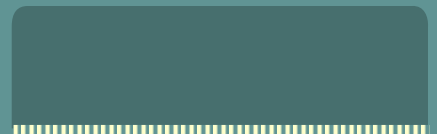
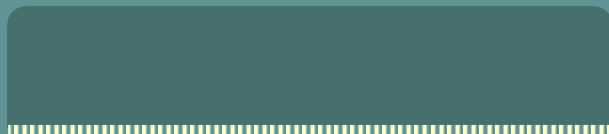
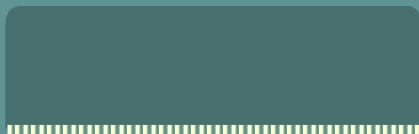
Primitivas send y receive:

- ♦ Proporcionar primitivas send y receive generales. Con un parámetro se distingue si es punto-a-punto o en grupo. El problema es si la comunicación punto-a-punto se hace con RPC.
- ♦ Proporcionar primitivas diferentes:
 - ♦ Group-send
 - ♦ Group-receive

COMUNICACIÓN EN GRUPO

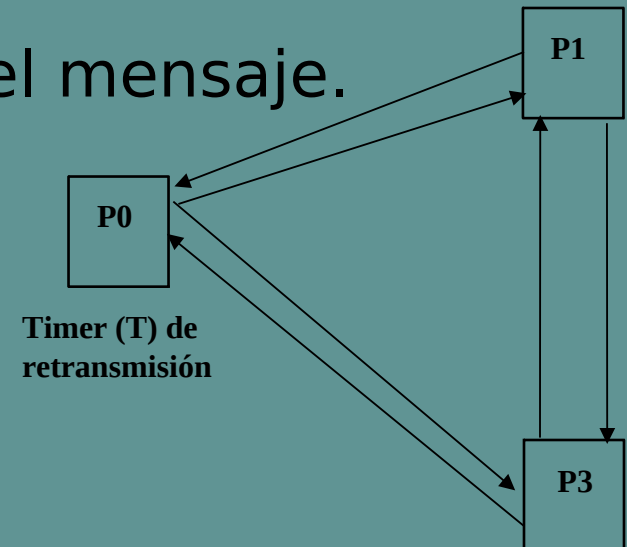
Atomicidad:

- ♦ Propiedad todo-o-nada. El mensaje debe llegar a todos los miembros del grupo o a ninguno.
- ♦ Implementación de multicast atómico:
 - ♦ ACK ante la recepción del mensaje para cada miembro del grupo. No funciona si hay fallas. Por ejemplo, no llega un mensaje a un miembro y el enviador se cae. A quienes no les llegó el mensaje, no se enteran que perdieron algo.



COMUNICACIÓN EN GRUPO

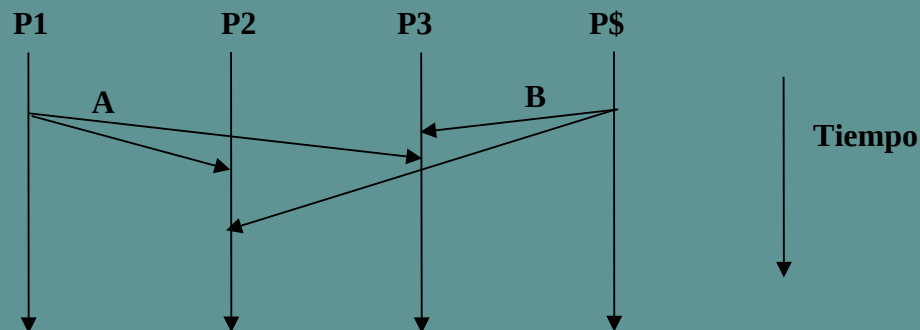
- ♦ ACK ante la recepción del mensaje y los procesos del grupo retransmiten el mensaje al recibirlo por primera vez.
 - ♦ Cada P_i se retransmite solo una vez.
 - ♦ Los sobrevivientes tendrán el mensaje.



COMUNICACIÓN EN GRUPO

Ordenamiento de mensajes:

- Ordenamiento por tiempo global.
- Ordenamiento por tiempo consistente.



COMUNICACIÓN EN GRUPO

Solapamiento de grupos:

El ordenamiento global dentro de cada grupo, no es suficiente.

